МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ М.В. ЛОМОНОСОВА

Факультет
вычислительной математики
и кибернетики

И.А. Волкова, А.А. Вылиток, Л.Е. Карпов

Сборник экзаменационных задач по курсу «Системы программирования» (ООП на Си++ и методы трансляции на основе формальных грамматик и автоматов)

(учебное пособие для студентов II курса)

Москва – 2023

УДК 004.43(075.8)
ББК 32.973-018.1я73
В67

Печатается по решению Редакционно-издательского совета факультета вычислительной математики и кибернетики МГУ имени М.В. Ломоносова

Рецензенты:

И.В. Машечкин – д.ф.-м.н.;
С.Ю. Соловьев – д.ф.-м.н.

Волкова И. А., Вылиток А.А., Карпов Л. Е.

В52 **Сборник экзаменационных задач по курсу «Системы программирования» (ООП на Си++ и методы трансляции на основе формальных грамматик и автоматов):** Учебное пособие для студентов II курса . – М.: Издательский отдел факультета ВМК МГУ имени М.В.Ломоносова; МАКС Пресс, 2023 – 297 с.
ISBN

В сборнике представлены задачи и упражнения по языку Си++, основм трансляции и формальным граамматикам, рекомендуемые студентам 2 курса для подготовки к коллоквиуму и экзамену в рамках курса «Системы программирования». Рассматриваемая версия языка Си++ соответствует стандарту ISO/IEC (1998), за исключением явно оговоренных случаев использования более поздних стандаротов.

В сборнике собраны задачи на использование основных механизмов объектно-ориентированных языков программирования: инкапсуляции, наследования, полиморфизма, а также задачи по методам трансляции на основе формальных грамматик и автомтатов. Задачи предлагались студентам в письменных проверочных работах в 2009 – 2020 гг.

Для студентов факультета ВМК в поддержку основного лекционного курса “Системы программирования”, а также для преподавателей, ведущих практические занятия по этому курсу.

Авторы выражают благодарность преподавателям кафедры алгоритмических языков за участие в обсуждении и составлении задач.

УДК 004.43(075.8)
ББК 32.973-018.1я73

Учебное издание
ВОЛКОВА Ирина Анатольевна
ВЫЛИТОК Алексей Александрович
КАРПОВ Леонид Евгеньевич
СБОРНИК ЭКЗАМЕНАЦИИОННЫХ ЗАДА ??
Учебное пособие для студентов II курса

19992, ГСП-2, Москва, Ленинские Горы, МГУ им. М.В. Ломоносова, 2-й учебный корпус
ISBN 978-5-317-04595-1

ОГЛАВЛЕНИЕ

2008 год
Коллоквиум

Вариант 1 _2008 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Есть ли ошибки в реализации функций `B::g()` и main()? Если есть, **объясните**, в чем они заключаются. Для всех правильных операторов этих функций укажите, из какой области видимости выбираются участвующие в их записи имена, используя операцию разрешения области видимости «**::**».

```
      int x = 0;
      void f (int a, int b){x = a+b;}
class A {
      int x;
public:
      void f (){x = 2;}
};
class B: public A {
public:
      void f (int a){::x = a;}
      void g ();
};
void B::g() {
      f();
      f(1);
      f(5 , 1);
      x = 2;
}
int main () {
      B b;
      f(5);
      f('+', 6);
      return 0;
}
```

2. Есть ли ошибки в приведенном фрагменте программы на C++? Если есть, то **объясните,** в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный канал вывода при работе программы?

```
class X {
public:     virtual int g (double x) { h ();   cout << "X::g" << endl;   return 1;}
            void h ()                { t ();   cout << "X::h" << endl;}
            virtual void t ()        { cout << "X::t" << endl;}
};
class Z: public X {
public:     int g (double y)         { h ();   cout << "Z::g" << endl;   return 3;}
            virtual void h ()        { t (1);   cout  << "Z::h" << endl;  }
            virtual void t (int k)   { cout  << "Z::t" << endl;  }
};
int main( ){
      X a;    Z b;    X * p = &b;
      p -> g(1.5);
      p -> h();
      p -> t(5);
   }
```

3. Является ли тип данных A из задания **1.** абстрактным типом данных (пояснить ответ)? Перегрузить для этого класса префиксную операцию ++ в соответствии с принятым для нее в C++ смыслом.

4. Для объектов класса B из задания **1.** определена функция ret:

B ret (B & x, B & y) { return x; }

Какие конструкторы и деструкторы и в каком порядке будут вызываться при работе следующего фрагмента программы

... ; { B b; b = ret (b, b); } ...

5. Описать прототипы двух перегруженных функций g из некоторой области видимости, для которых будут верны следующие обращения к ним:

g (1);

g (‘+’, ‘+’);
g (2.3);
g (3, “str”);

6. Для каких конструкций C++ существует понятие шаблона? Привести пример реализации какой-либо шаблонной конструкции.

7. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
      class X;
      void F(X & x, int n);
class X {
      public: X() {     try          { F(*this, -2);    cout << 1 << endl; }
                        catch (X)    { cout << 2 << endl; }
                        catch (int) { cout << 3 << endl; } }
              X (X &) { cout << 12 << endl; }
};
class Y: public X {
      public:     Y ()         {cout << 4 << endl;}
                  Y (Y & a)    {cout << 5 << endl;}
                  ~Y ()        {cout << 6 << endl;}
};
void F(X & x, int n) {  try    { if (n < 0) throw x;
                                 if (n > 10) throw 1;
                                       cout << 7 << endl;    }
                        catch (int) { cout << 8 << endl; }
                        catch (X&)  { cout << 9 << endl;   throw; }
}
int main() { try { Y a; }
             catch (...) {  cout << 10 << endl;  }
             cout << 11 << endl;
}
```

8. Есть ли ошибки в тексте приведенной программы? Можно ли исправить описание класса, не вводя дополнительных членов, чтобы программа стала верной? Если да, то как?

```
class A {
      public:
            int y;
            void f () {cout << "f\n";}
};
int A::y;
int main () {
      A::y = 1;
      const A a;
      a.f();
      return 0;
}
```

9. Дать определение функции – друга класса. Привести пример класса, имеющего друга, описание этого друга, и пример его использования.

10. STL: Написать функцию, которая удваивает (добавляет еще один такой же) каждый элемент заданного контейнера-списка lst <int>, а затем распечатывает его элементы в обратном порядке.

Вариант 2 _2008 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Есть ли ошибки в реализации функций `D::h()` и main()? Если есть, **объясните**, в чем они заключаются. Для всех правильных операторов этих функций укажите, из какой области видимости выбираются участвующие в их записи имена, используя операцию разрешения области видимости «**::**».

```
#include <iostream>
using namespace std;
      double a = 0;
      void f (double x){a = x;}
struct B {
```

```
        double a;
        void f (){a = 2;}
};
class D: B {
public:
        void f (int a){::a = a;}
        void h ();
};
void D::h() {
        f(1.2);
        f();
        a = 2;
}
int main () {
        D d;
        f();
        f(6);
        return 0;
}
```

2. Есть ли ошибки в приведенном фрагменте программы на C++? Если есть, то **объясните,** в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный канал вывода при работе программы?

```
class T {
public:     virtual  int f (int x)  { cout  << "T::f" << endl;    return 0; }
            void g ()               { f (1);  cout  << "T::g" << endl; }
            virtual void h ()       { g ();  cout  << "T::h" << endl; }
};
class S: public T {
public:     int f (double y)        { cout  << "S::f" << endl;    return 2;  }
            virtual void g ()       { f (1);    cout  << "S::g" << endl;  }
        virtual void h ()       { g();   cout  << "S::h" << endl;  }
};
int main( ){
        T t;  S s;  T *p = &s;
        p -> f (1.5);
        p -> g ();
        p -> h ();
}
```

3. Является ли тип данных B из задания **1.** абстрактным типом данных (пояснить ответ)? Перегрузить для этого класса постфиксную операцию -- в соответствии с принятым для нее в C++ смыслом.

4. Для объектов класса D из задания **1.** определена функция empty:

void empty (D & a, D b) { }

Какие конструкторы и деструкторы и в каком порядке будут вызываться при работе следующего фрагмента програмы

... ; { D d; empty (d, d); } ...

5. Описать прототипы двух перегруженных функций f из некоторой области видимости, для которых будут верны следующие обращения к ним:

f ();
f (“abc”);
f (2);
f (‘+’, 3);

6. Дать определение абстрактного класса в C++. Привести пример. Можно ли описывать конструкторы, вводить неконстантные члены-данные, заводить объекты абстрактного класса?

7. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
class A {
        public: A () {    cout << 1 << endl;      }
};
class B: public A {
        public: B (int n) { try { if (n == 0) throw *this;
                                  if (n > 11) throw 11; }
                            catch (int)     { cout << 2 << endl; }
                            catch (B&)      { cout << 3 << endl;   throw; }
                            cout << 4 << endl; }
                B (B&)    {cout << 5 << endl;}
                ~B ()     {cout << 6 << endl;}
```

```
};
int main() { try              { B b(0);   B c (3); }
             catch (...)      {  cout << 7 << endl;  }
             cout << 8 << endl;
}
```

9. Есть ли ошибки в тексте приведенной программы? Можно ли исправить описание класса, не вводя дополнительных членов, чтобы программа стала верной? Если да, то как?

```
class X {
      public:
            void g () {cout << "g\n";}
            int h (int n) {cout << "f\n"; return n}
};
int main () {
      int k;
      const X x;
      X::g();
      k = x.h(5);
      return 0;
}
```

9. Описать суть работы оператора dynamic_cast. Привести пример его использования.

10. STL: Написать функцию, которая удаляет каждый второй элемент заданного контейнера-вектора v <char>, а затем распечатывает его элементы в обратном порядке.

Ответы и баллы :::: вариант 1_2008

Максимальная оценка всех задач – 10 баллов (всего 100 баллов)

1.

```
void B::g() {
      f();          // ошибка, эта f не видна
      f(1);         // B::f(1);
      f(5 , 1);     // ошибка эта f не видна
      x = 2;        // ошибка, так как int x private в базовом классе
}
int main () {
      B b;
      f(5);         //ошибка, эта f не видна
      f('+', 6);    //::f('+', 6);
      return 0;
}
```

Критерии: За каждую не найденную ошибку **- 2**
За каждую лишнюю ошибку или неверно написанное с «::» имя
(это строго взаимоисключающие ошибки) **- 2**

2.

```
int main( ){
      X a;    Z b;    X * p = &b;
      p -> g(1.5); // печатается    Z::t
                                    Z::h
                                    Z::g
      p -> h();  //печатается       X::t
                                    X::h
      p -> t(5);  // ошибка, X::t – без параметров
}
```

Критерии: За каждую не найденную ошибку, лишнюю ошибку или
неправильно вызванную функцию **– 2**
(может быть больше 10 баллов, но причины ошибок пересекаются)

3.

Да, т.к. не содержит открытых полей-данных.
Метод класса A:

```
A &  operator ++ () {
        i++;
        return *this;
}
```

Критерии: Не АТД **-3**
Нет ссылки в возвращаемом результате **-1**
Неверно реализована операция **- 7**

4.

- конструктор умолчания A,
- конструктор умолчания B,
- конструктор умолчания A,
- конструктор копирования B,
- деструктор B,
- деструктор A,
- деструктор B,
- деструктор A.

Критерии: За каждую ошибку (пропущенную, лишнюю или неверную печать) **- 2**

5. void g (double a, int b = 0);
void g (int, const char * = 0);
Возможны варианты с double, char...

Критерии: За каждую ошибку **– 5**
Функций не две **-10**

6. Приведен пример шаблона класса либо шаблона функции

Критерии: Нет либо шаблона функций, либо классов – **5**,
ошибки в примере существенные **- 4,** несущественные - **2**.

7. 12
9
2
4
6
11

Критерии: За каждую неверную (лишнюю или пропущенную) печать **-2**

8. Исправления:

```
class A {
        public:
                static int y;
                void f () const {cout << "f\n";}
};
```

Критерии: Ответ нет - **-10**, не добавили static **-5**, не добавили const **-5**.

9. Друг класса – это функция, не являющаяся членом этого класса, но имеющая доступ к private и protected членам класса.

```
class X {          int a;
        public:  friend void fff (X*, int);
};
void fff  ( X *  p,  int  i ) {  p -> a =  i;          }
void f ( ) {          X  obj;   fff (& obj, 10);     }
```

Критерии: Нет определения **– 8**, существенные ошибки в примере **-5.**

10.
```
void g (list <int> &lst) {
      list < int > :: iterator  p = lst.begin ();
      while ( p != lst.end()) {
            lst.insert(p, *p);
            p++;
      }
      list < int > :: reverse_iterator rp = lst.rbegin ();
      while ( rp!=lst.rend ()) {
            cout << *rp << ' ';
            rp++;
      }
      cout << endl;
}
```
Критерии: За каждую существенную ошибку **-3**.

Ответы и баллы :::: вариант 2_2008

Максимальная оценка всех задач – 10 баллов (всего 100 баллов)

1.
```
void D::h() {
      f(1.2);  // D::f
      f();     // ошибка эта f не видна
      a = 2;   // B::a = 2
}
int main () {
      D d;
      f();    //ошибка
      f(6);   //::f
      return 0;
}
```
Критерии: За каждую не найденную ошибку **- 2**
За каждую лишнюю ошибку или неверно написанное с «::» имя (это строго взаимоисключающие ошибки) **- 2**

2.
```
int main( ){
    T t;  S s;  T *p = &s;

    p -> f (1.5); // печатается   T::f
    p -> g ();    // печатается   T::f
                                 T::g
    p -> h();     //печатается    S::f
                                 S::g
                                 S::h
}
```
Критерии: За каждую лишнюю ошибку или неправильно вызванную функцию **– 2**
(может быть больше 10 баллов, но причины ошибок пересекаются)

3.
Нет, т.к. содержит открытые поля-данные.

Метод класса B:

```
B  operator -- ( int t) {
      B b = *this;
      a = a - 1;
      return  b;
}
```

Критерии: АТД **-3**
Есть ссылка в возвращаемом результате **-1**
Неверно реализована операция **- 7**

4.

- конструктор умолчания B,
- конструктор умолчания D,
- конструктор умолчания B,
- конструктор копирования D,
- деструктор D,
- деструктор B,
- деструктор D,
- деструктор B.

Критерии: За каждую ошибку (пропущенную, лишнюю или неверную печать) **- 2**

5. void f (const char * a = 0);
void f (int a, int b = 0);
Возможны варианты с char...
Критерии: За каждую ошибку **– 5**
Функций не две **-10**

6. Класс, в котором есть хотя бы одна чистая виртуальная функция, называется абстрактным классом. Нельзя создавать объекты абстрактного класса, но можно описывать указатели на абстрактный класс, использовать абстрактный класс в качестве базового класса, вводить неконстантные члены-данные, описывать конструкторы.

<Пример абстрактного класса>

Критерии: ошибки в примере (существенные) **-4**
только пример **-5,** только определение – **5**
за каждый неверный ответ на последний вопрос **- 2**.

8. 1
1
5
3
7
6
8

Критерии: За каждую неверную (лишнюю или пропущенную) печать **-2**

8. Исправления:

```
class X {
      public:
            static void g () {cout << "g\n";}
            int f (int n) const {cout << "f\n"; return n;}
};
```

Критерии: Ответ нет - **-10**, не добавили static **-5**, не добавили const **-5**.

9. Оператор **dynamic_cast** реализует приведение **<u>полиморфных типов (указателей или ссылок</u>**) в динамическом режиме. При неудачной попытке приведения типов результатом выполнения dynamic_cast является 0, если в операторе использовались указатели. Если же использовались ссылки, генерируется исключительная ситуация **bad_cast**. Пример:

```
Base *bp, b_ob;
Derived *dp, d_ob;
bp = &d_ob;
dp = dynamic_cast <Derived *> (bp);
```

Критерии: Совсем неверное описание оператора **-8**, нет полиморфного типа **-4**
неверный пример **-5**.

10.

```
void f (vector <char> &v) {
      vector < char > :: iterator  p = v.begin ();
      while ( p != v.end()) {
            p++;
            if (p != v.end()) {v.erase(p);}
      }
      vector < char > :: reverse_iterator rp = v.rbegin ();
      while ( rp != v.rend ()) {
            cout << *rp << ' ';
            rp++;
      }
      cout << endl;
}
```

Критерии: За каждую существенную ошибку **-3**.

2008 год
Экзамен

ФИО_____ № группы_____

1. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
class X;
void  F (X& x, int n);
class X
{ public: X ()    { try { F (* this, -2);          cout <<  1 << endl; }
                    catch (X&)    {                cout <<  2 << endl; }
                    catch (int n) { F (* this, -n); cout <<  3 << endl; }
                  }
          X (X&)  {                                cout <<  4 << endl; }
         ~X ()    {                                cout <<  5 << endl; }
};
void F (X& x, int n) { try { if (n < 0) throw x;
                             if (n > 1) throw n;   cout <<  6 << endl;
                           }
                    catch (int) {                 cout <<  7 << endl;   throw; }
                    catch (X&)  {                 cout <<  8 << endl;   throw; }
}
void main()
{ try { X a; }
  catch (...)  {                                  cout <<  9 << endl;  }
                                                  cout << 10 << endl;
}
```

Ответ: _____

2. Дать определение класса, являющегося другом некоторого другого класса. Привести пример класса, имеющего дружественный класс, описание этого друга, и пример его использования.

3. STL: Написать функцию, которая считает количество положительных элементов заданного контейнера-списка `list<int>`, а затем распечатывает это значение (выдает в стандартный поток `cout`).

4. Дать определение абстрактного класса. Привести небольшой пример, поясняющий использование абстрактных классов в программе на C++.

5. Есть ли ошибки в тексте приведенной программы? Можно ли исправить определение класса и функции, чтобы программа стала верной? Если да, то как?

```
#include <iostream.h>
class A { public: int y; void f (int t) { cout << "f\n"; } };
g ()
{ const A a;
  a.f(A::y);
  return 0;
}
```

6. Дана грамматика G:

$S \to Bc$
$S \to dSc$
$dBc \to dAc$
$dA \to dAd$
$A \to d$

(а) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.
Ответ:

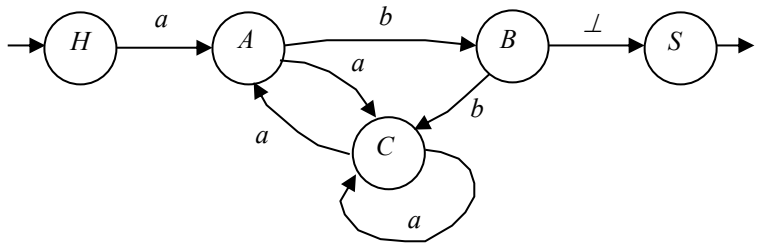
(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\Im$	$L(G) \in \Im$? (да/нет)
контекстно-свободные языки	
контекстно-зависимые языки	
регулярные языки	
языки типа 0	

(в) Определение неукорачивающей грамматики. **Ответ**:

7. Конечный автомат $\mathcal{A}$ задан в виде диаграммы состояний:



(а) Показать, что автомат $\mathcal{A}$ не является детерминированным.

(б) По автомату $\mathcal{A}$ с помощью соответствующего алгоритма построить эквивалентный детерминированный автомат $\mathcal{A}_D$.

(в) По автомату $\mathcal{A}_D$ построить эквивалентную леволинейную грамматику *G*.

8. Дана грамматика:

G_1 : $S \to cAb$
$A \to dS \mid d$

Вставить в G_1 действия вида ⟨*cout* << …⟩ по переводу цепочек языка $L_1 = L(G_1)$ в цепочки языка $L_2 = \{ b(a)^{2k} \mid k \geq 1 \}$ так, чтобы символов *a* в цепочке из L_2 было в два раза больше, чем символов *d*
в соответствующей цепочке из L_1.

Ответ:

9. Записать в постфиксной записи оператор программы на Си++:

```
if (i > 0) x = j; else if (j > 0) x = i; else if (i < j) x = j;
```

Ответ:

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	

10. Какие виды отношений между классами поддерживают большинство объектно-ориентированных языков программирования? Коротко охарактеризуйте их.

ОТВЕТЫ и КРИТЕРИИ

За каждую задачу — максимум 10 баллов.

1. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
class X;
void  F (X& x, int n);
class X
{ public: X ()     { try { F (* this, -2);           cout <<  1 << endl; }
                     catch (X&)    {                 cout <<  2 << endl; }
                     catch (int n) { F (* this, -n); cout <<  3 << endl; }
                   }
          X (X&)   {                                 cout <<  4 << endl; }
         ~X ()     {                                 cout <<  5 << endl; }
};
```

```
void F (X& x, int n) { try { if (n < 0) throw x;
                            if (n > 1) throw n;    cout <<  6 << endl;
                          }
                     catch (int) {                 cout <<  7 << endl;   throw; }
                     catch (X&)  {                 cout <<  8 << endl;   throw; }
}
void main()
{ try { X a; }
  catch (...)  {                                   cout <<  9 << endl;  }
                                                   cout << 10 << endl;
}
```

Ответ: 4 ↵ 8 ↵ 2 ↵ 5 ↵ 5 ↵ 10 ↵

КРИТЕРИИ:

За каждую неверную (лишнюю или пропущенную) печать: **–2** балла.
(Если числа записаны в строчку без указания «перехода на новую строку», через пробел или через запятую, то за это не снижать).

2. Дать определение класса, являющегося другом некоторого другого класса. Привести пример класса, имеющего дружественный класс, описание этого друга, и пример его использования.

Ответ: Дружественный класс — это класс, все методы которого являются друзьями данного класса.

КРИТЕРИИ:

Нет определения — снижение на 5 баллов. Нет примера или пример не разъясняет нужное понятие — снижение на 5 баллов. Пример — только на описание (без использования) дружественного класса — снижение на 3 балла.

3. STL: Написать функцию, которая считает количество положительных элементов заданного контейнера-списка `list<int>`, а затем распечатывает это значение (выдает в стандартный поток `cout`).

Ответ:
```
void f (list <int> & l) { int count = 0;
        list <int> :: iterator p = l.begin ();
        while (p != l.end()) if (* p ++ > 0) count ++;
        cout << count << endl;
}
```

КРИТЕРИИ: За каждую существенную ошибку — снижение на 3 балла.

4. Дать определение абстрактного класса. Привести небольшой пример, поясняющий использование абстрактных классов в программе на C++.

Ответ: *Класс, в котором есть хотя бы одна чистая виртуальная функция, называется абстрактным классом*. Нельзя (1) создавать объекты абстрактного класса, но можно (2) описывать указатели на абстрактный класс, (3) использовать абстрактный класс в качестве базового класса, (4) вводить неконстантные члены-данные, (5) описывать конструкторы.

КРИТЕРИИ: Ошибки в примере (существенные, то есть не показывающее перечисленных свойств абстрактных классов) — снижение на 1 балл за каждое пропущенное в примере свойство. Есть только пример — снижение на 5 баллов, только определение (выделенное шрифтом) — снижение на 5 баллов

5. Есть ли ошибки в тексте приведенной программы? Можно ли исправить определение класса и функции, чтобы программа стала верной? Если да, то как?

```
#include <iostream.h>
class A { public: int y; void f (int t) { cout << "f\n"; } };
g ()
{ const A a;
  a.f(A::y);
  return 0;
}
```

Ответ:

```
#include <iostream.h>
class A { public: int y; void f (int t) { cout << "f\n"; }
                  A () { y = 1; /* здесь любое значение */ }
};
int g ()
{ A a;
  a.f(a.y);
  return 0;
}
```

КРИТЕРИИ: За каждую не найденную или лишнюю ошибку — снижение на 3 балла.

6. Дана грамматика G:

$S \to Bc$
$S \to dSc$
$dBc \to dAc$
$dA \to dAd$
$A \to d$

(а) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.
Ответ: $k = 1$
(третье правило не удовлетворяет типу 2)

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

$L(G) = \{d^{n+k} c^n \mid n \geq 1, k \geq 1\}$ **Ответ:**

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные языки	**да**
контекстно-зависимые языки	**да**
регулярные языки	**нет**
языки типа 0	**да**

(в) Определение неукорачивающей грамматики. **Ответ**:

Грамматика $G = \{V_T, V_N, P, S\}$ — ***неукорачивающая*** , если каждое правило из P имеет вид: $\alpha \to \beta$,
где $\alpha \in (V_T \cup V_N)^+$, $\beta \in (V_T \cup V_N)^+$ и $|\alpha| <= |\beta|$.

Дополнение. Если студент добавил к определению, что неукорачивающая грамматика может содержать, кроме прочих, единственное правило с пустой правой частью $S \to \varepsilon$, **и при этом** S не содержится в правых частях правил, то считать такой ответ верным – баллы не снижать.

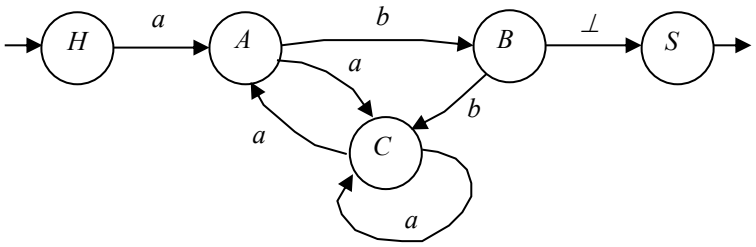
КРИТЕРИИ:

(а) нет ответа; неверный ответ : **–3**
(обоснование не обязательно)

(б) за каждый неверный или отсутствующий ответ: **–1 (т.е. всего до –4 за этот пункт)**
(обоснование и формула языка не обязательны)

(в) определение отсутствует или с ошибками: **–3**

7. Конечный автомат $\mathcal{A}$ задан в виде диаграммы состояний:



(а) Показать, что автомат $\mathcal{A}$ не является детерминированным.

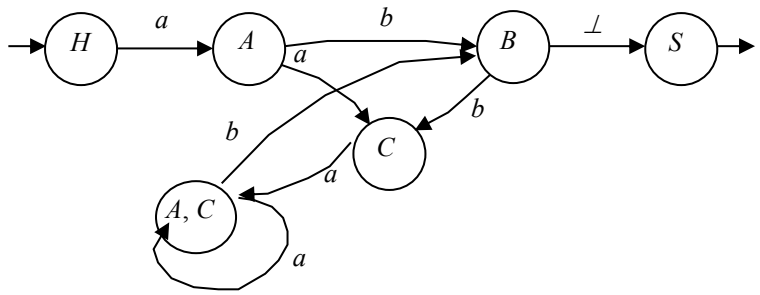
Ответ: Из состояния C по символу a возможен переход в более, чем одно состояние: (C, A)

(б) По автомату $\mathcal{A}$ с помощью соответствующего алгоритма построить эквивалентный детерминированный автомат $\mathcal{A}_D$.
Ответ:

Построим таблицу переходов для $\mathcal{A}_D$:

	a	b	$\perp$
$\{ H \}$	$\{ A \}$	$\varnothing$	$\varnothing$
$\{ A \}$	$\{ C \}$	$\{ B \}$	$\varnothing$
$\{ B \}$	$\varnothing$	$\{ C \}$	$\{ S \}$
$\{ C \}$	$\{A, C \}$	$\varnothing$	$\varnothing$
$\{A, C \}$	$\{A, C \}$	$\{ B \}$	$\varnothing$
$\{ S \}$	$\varnothing$	$\varnothing$	$\varnothing$

$\mathcal{A}_D$:



(в) По автомату $\mathcal{A}_D$ построить эквивалентную леволинейную грамматику G.

Ответ:

G: $S \to B\perp$

$B \to Ab \mid S_{\{A,C\}}\, b$

$S_{\{A,C\}} \to S_{\{A,C\}}\, a \mid Ca$

$C \to Bb \mid Aa$

$A \to a$

Замечание. В качестве промежуточного этапа в грамматике допускается правило вида $H \to \varepsilon$, которое устраняется на последнем шаге построения. Результат должен совпасть с тем, что приведен слева (с точностью до обозначений нетерминалов).

КРИТЕРИИ: (а) нет ответа; неверное обоснование: **–3**

(б) нет ответа; неэквив. авт-т; недетерм. авт-т : **–4**

(в) нет ответа; неэквивалентная автомату из пункта (б) гр-ка: **–3**

эквивалентная, но не леволинейная гр-ка: **–1**

8. Дана грамматика:

G_1 : $S \to cAb$

$A \to dS \mid d$

Вставить в G_1 действия вида ⟨*cout* << …⟩ по переводу цепочек языка $L_1 = L(G_1)$ в цепочки языка $L_2 = \{b(a)^{2k} \mid k \geq 1\}$, так, чтобы символов a в цепочке из L_2 было в два раза больше, чем символов d
в соответствующей цепочке из L_1.

Ответ: Соглашение: запись ⟨a⟩ является сокращением ⟨*cout* << "*a* ";⟩

Грамматика с действиями:

(возможные ответы)

(1) $S \to cAb\ \langle aa \rangle$
$A \to dS \mid d\ \langle b \rangle$

(2) $S \to cAb$
$A \to dS\ \langle aa \rangle \mid d\ \langle baa \rangle$

(3) $S \to cAb\ \langle a \rangle$
$A \to dS \langle a \rangle \mid d\ \langle ba \rangle$

КРИТЕРИИ:

За каждое неправильно вставленное (или пропущенное) действие вида ⟨*cout* << …⟩: **–4**

Если, кроме действий по печати символов ⟨*cout* << …⟩, есть другие вычисления: **0**

9. Записать в постфиксной записи оператор программы на Си++:

```
if (i > 0) x = j; else if (j > 0) x = i; else if (i < j) x = j;
```

Ответ:

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
i	0	>	***11***	!F	x	j	=	***29***	!	j	0	>	***21***	!F	x	i

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
=	***29***	!	i	j	<	***29***	!F	x	j	=								

КРИТЕРИИ:

Неправильное представление о ПОЛИЗ операторов, перестановка операций, замена одних операций на другие, оптимизация переходов, инвертирование условия, пропуск указания на адресность значения в присваивании – снижение до 0 баллов

Ошибка в подсчете адреса переходов – снижение на 4 балла.

Использование перехода по лжи разрешается без пояснений. Разрешается использование перехода по истине, если правильно описана семантика этой операции.

10. Какие виды отношений между классами поддерживают большинство объектно-ориентированных языков программирования? Коротко охарактеризуйте их.

Ответ:

- ассоциация (показывает, что два класса концептуально взаимодействуют друг с другом);
- наследование («частное — общее», «обобщение — специализация», «есть», «is a»);
- агрегация («часть — целое», «содержит», «has a»);
- использование (в функции одного класса используются имена, локальные объекты, функции другого класса);
- инстанцирование (связь между шаблоном класса и классом-результатом генерации по шаблону).

КРИТЕРИИ:

Нет понимания сути отношений между классами — снижение до 0 баллов. За пропуск любого из видов отношений — снижение на 2 балла, за пропуск пояснения к какому-либо виду отношений (когда указано только наименование) — снижение на 1 балл.

2009 год

Коллоквиум

Вариант 1_2009 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Есть ли ошибки в реализации функций *C::g()* и *main()*? Если есть, исправьте найденные ошибки, используя операцию разрешения области видимости "**::**". Для всех операторов этих функций, используя операцию разрешения области видимости "**::**", также укажите, из какой области видимости выбираются участвующие в их записи имена.

```
int x = 0;
int f (int a, int b)        {       return   x = a + b; }
class A { int x;
          public: A (int n = 1)          { x = n; }
                  int f () {      return ::x = x; }
};
class B { int x;
          public:  B (int n = 2)         { x = n; }
};
class C:  public A,   public B  { int x;
          public: int f (int a) { return ::x = x; }
            void g ();
};
void C::g() {       x = f ();
                    f (3);
                    x = f (4, 5);
                    x = 6;
}
int main () {       C c;
                    c.f ();
                    c.f (7);
                    x = f ('8', 9);
                    return -1;
}
```

Ответ:

```
C::x = A::f ();
C::f (3);
C::x = ::f (4, 5);
C::x = 6;

c.A::f ();
c.C::f (7);
::x =::f ('8', 9);
```

Отмечены исправляющие (обязательные) изменения.

Критерии: За каждое не отмеченное или неверное уточнение области видимости (из 7) **-2**. За каждое не найденное исправление (из 3) еще **-3**.

Есть ли ошибки в реализации функции *main()*? Удалите неверные операторы из этой функции (если они есть). Для всех правильных операторов этой функции, используя операцию разрешения области видимости "**::**", также укажите, из какой области видимости вызываются используемые в них функции *f()*.

```
int x = 0;
int f (int a) { return x = a; }
class A { int x;
          public: A (int n = 1)          { x = n; }
                  int f (int a)   { return x = a; }
};
class B { int x;
          public:  B (int n = 2)          { x = n; }
          virtual   int f ()      { return  x = ::x; }
};
class C:  public A, public B { int x;
          public:   int f (int a) { return  x = a; }
                    int f ()      { return  x = ::x; }
};
```

```
int main () { A a; B b; C c;
              A * pa = & a; B * pb = & b; C * pc = & c;
              x =       f (3);
              x = pa -> f ();
              x = pa -> f (4);
     pa = pc; x = pa -> f ();
              x = pa -> f (5);
              x = pb -> f ();
              x = pb -> f (6);
     pb = pc; x = pb -> f ();
              x = pb -> f (7);
              x = pc -> f ();
              x = pc -> f (8);
              return -1;
}
```

Ответ:

```
              x =       f (3); //  ::f (3)
//            x = pa -> f ();
              x = pa -> f (4); // A::f (4)
//   pa = pc; x = pa -> f ();
              x = pa -> f (5); // A::f (5)
              x = pb -> f ();  // B::f ()
//            x = pb -> f (6);
     pb = pc; x = pb -> f ();  // C::f ()
//            x = pb -> f (7);
              x = pc -> f ();  // C::f ()
              x = pc -> f (8); // C::f (8)
```

Комментариями закрыты неправильные вызовы.

Критерии: За каждое неправильное или неуказанное уточнение области видимости (из 7) **-2**, за ненайденную ошибку (из 4) **-3**.

Есть ли ошибки в реализации функций *B::g()* и *main()*? Если есть, исправьте неверные операторы из этой функции, используя операцию разрешения области видимости "**::**".Для всех операторов этих функций укажите, из какой области видимости вызываются используемые в них функций *f()* и *g()*.

```
int x = 0;
int f ()                                    { return x = 1;     }
class A { int x;
          public:A ( int n = 2)             {        x = n;     }
                    int f ()                { return x = 3;     }
                    int f (int a, int b)    { return x = a % b; }
};
class B:  public A { int x;
          public:    int f (int a)          { return x = a;     }
                     int f (int a, int b)   { return x = a + b; }
                     int g (A * a, B * b);
};
int  B::g (A * pa, B * pb)
{             x =       f ();
              x =       f (5);
              x =       f (6, 6);
              x =   A:: f (5);
              return -1;
}
int main ()
{ B a;        x =     a.f ();
              x =     a.f (7);
              return  a.g (& a, & a);
}
```

Ответ:

```
              x =   A:: f ();     // A::f ()
              x =    B:: f (5);    // B::f (5)
              x =    B:: f (6, 6); // B::f (6, 6)
              x =   B:: f (5);    // B::f (5)

              x = a.A:: f ();     // A::f ()
              x = a.B:: f (7);    // B::f (7)
              return  a.B::g (& a, & a);
```

Отмечены исправляющие (обязательные) изменения.

Критерии: За каждое не отмеченное уточнение области видимости (из 7) **-2**. За каждое не найденное исправление (из 3) еще **-3**.

Есть ли ошибки в определениях классов *A*, *B* и *C*, препятствующие правильной компиляции функций *B::g()* и *main()*? Если такие ошибки есть, предложите такие исправления в классах, чтобы реализации указанных функций стали правильными. Для всех операторов этих функций, используя операцию разрешения области видимости "**::**", укажите, из какой области видимости вызываются используемые в них функции *f()* и *g()*.

```
int x = 0;
int f (int a) { return x = a; }
class A { int x;
          public:   A (int n = 1)        { x = n; }
                    int f (int a) { return x = a; }
};
class B { int x;    B (int n = 2)        { x = n; }
          public:   int f (int a) { return x = a; }
                    void g ();
};
class C:  public A, B { int x;
          public:   int f (int a) { return x = a; }
};
void B::g () { x = f (3);
               x = f (4);
               x = f (5);
}
int  main () { A a; B b; C c;
               A * pa = & a; B * pb = & b; C * pc = & c;
               x = pa -> f (7);
      pa = pc; x = pa -> f (8);
               x = pb -> f (7);
               x =       f (8);
      pb = pc; x = pb -> f (7);
               x = pc -> f (8);
               c.g ();
               return -1;
}
```

Ответ:

```
               x = f (3);       // B::f (3)
               x = f (4);        // B::f (4)
               x = f (5);        // B::f (5)

               x = pa -> f (7); // A::f (7)
      pa = pc; x = pa -> f (8); // A::f (8)
               x = pb -> f (7); // B::f (7)
               x =       f (8); //  ::f (8)
      pb = pc; x = pb -> f (7); // B::f (7)
               x = pc -> f (8); // C::f (8)
               c. B::g ();
```

Конструктор класса *B* должен быть сделан открытым методом класса *B* (*public*). Наследование класса *B* классом *C* должно быть сделано открытым (*public:*).

Критерии: За каждое не найденное исправление (из 2) **-4**. За каждое не отмеченное уточнение области видимости (из 10) **-2**.

2. Есть ли ошибки в приведенном фрагменте программы на C++? Если есть, то **объясните,** в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный поток вывода при работе программы?

```
class A {
     public: virtual void f (int x) { h (x); cout << "A::f," << x << endl; }
                     void g ()      { h (0); cout << "A::g"       << endl; }
             virtual void h (int k) {        cout << "A::h," << k << endl; }
};
class B: virtual public A {
     public:         void f (int y) { h (y); cout << "B::f," << y << endl; }
                     void g ()      { h (1); cout << "B::g"       << endl; }
                     void h (int k) {        cout << "B::h," << k << endl; }
};
int main( ){
     A a; B b; A * p = & b;
     p -> f (2);
     p -> g ();
```

```
    p -> h ();
    p -> h (3);
}
```

Ответ: Ошибочным является вызов *p -> h()* (нет функции *h()* без параметров).
Будет выдано в поток: B::h,2 B::f,2 B::h,0 A::g B::h,3

Критерии: За каждую не найденную ошибку, лишнюю ошибку или неправильно вызванную функцию – **2**.

Есть ли ошибки в приведенном фрагменте программы на C++? Если есть, то **объясните,** в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный поток вывода при работе программы?

```
class C {
    public: virtual void f (int x) { h (x); cout << "C::f," << x << endl; }
            virtual void g ()      { h (0); cout << "C::g"       << endl; }
            virtual void h ()      {        cout << "C::h"       << endl; }
            virtual void h (int k) { h ();  cout << "C::h," << k << endl; }
};
class D: public C {
    public: virtual void f (int y) { h (y); cout << "D::f," << y << endl; }
            virtual void g ()      { h (1); cout << "D::g"       << endl; }
            virtual void h ()      {        cout << "D::h"       << endl; }
            virtual void h (int k) { h ();  cout << "D::h," << k << endl; }
};
int main( ){
    C c; D d; C * p = & d;
    p -> f (2);
    p -> g ();
    p -> h ();
    p -> h (3);
}
```

Ответ: Ошибок нет.
Будет выдано в поток: D::h D::h,2 D::f,2 D::h D::h,1
D::g D::h D::h D::h,3

Критерии: За каждую лишнюю ошибку или неправильно вызванную функцию **-2**.

Есть ли ошибки в приведенном фрагменте программы на C++? Если есть, то **объясните,** в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный поток вывода при работе программы?

```
class T {
    public: virtual void f (int x) { h ();  cout << "T::f," << x << endl; }
                    void g ()      { h ();  cout << "T::g"       << endl; }
            virtual void h ()      {        cout << "T::h"       << endl; }
};
class U: virtual public T {
    public:         void f (int y) { h (y); cout << "U::f," << y << endl; }
            virtual void g ()      { h (0); cout << "U::g"       << endl; }
                    void h (int k) {        cout << "U::h," << k << endl; }
};
int main( ){
    T t; U u; T * p = & u;
    p -> f (1);
    p -> g ();
    p -> h ();
    p -> h (2);
}
```

Ответ: Ошибочным является вызов *p -> h(2)* (Функция *h(int)* из производного класса не замещает *h()* базового класса, так как имеет отличающийся набор параметров).
Будет выдано в поток: U::h,1 U::f,1 T::h T::g T::h

Критерии: За каждую не найденную ошибку, лишнюю ошибку или неправильно вызванную функцию – **2**.

Есть ли ошибки в приведенном фрагменте программы на C++? Если есть, то **объясните,** в чем они заключаются. Ошибочные конструкции вычеркнуть из текста программы. Что будет выдано в стандартный поток вывода при работе программы?

```
class R {
     public: virtual void f (int x) { g ();  cout << "R::f," << x << endl; }
                     void g ()      { h ();  cout << "R::g"       << endl; }
             virtual void h ()      {        cout << "R::h"       << endl; }
};
class S: virtual public R {
     public:         void f (int x) { g ();  cout << "S::f," << x << endl; }
                     void g ()      { h ();  cout << "S::g"       << endl; }
                     void h ()      {        cout << "S::h"       << endl; }
};
int main( ){
     R r; S s; R * p = & s;
     p -> f (0);
     p -> g ();
     p -> g (1);
     p -> h ();
     p -> h (2);
}
```

Ответ: Ошибочными являются вызовы *p -> g(1)* и *p -> h(2)* (нет функций с таким набором параметров).
Будет выдано в поток: S::h S::g S::f,0 S::h R::g S::h

Критерии: За каждую не найденную ошибку, лишнюю ошибку или неправильно вызванную функцию **–2**.

3. Описать прототипы двух перегруженных функций *f()* из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
f (0, 1);        f (1, 0);        f (0, "m");        f ("n", 0);
```

Ответ:
```
void f (const char *, int);
void f (int, const char *);
```
Критерии: За каждую ошибку **–5**. Если приводятся прототипы более, чем двух функций **–10**. Пропуск слова *const* в обеих функциях одновременно считать за одну ошибку.

Объект *x* принадлежит классу *X*, для которого определены конструктор преобразования типа *int* в класс *X* и функция преобразования класса *X* в тип *int*. Описать прототипы двух перегруженных функций *f()* из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
f ("p");    f (x, 0);    f (0, 0);     f (x, "q");    f (1, "r");
```

Ответ:
```
void f (int, const char *);    либо    void f (X, const char *);
void f (const char *);
```
Критерии: За каждую ошибку **–5**. Если приводятся прототипы более, чем двух функций **–10**. Пропуск слова *const* в обеих функциях одновременно считать за одну ошибку.

Описать прототипы двух перегруженных функций *f()* из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
f (1000000000000);  f (1);    f ();    f (0, 0);    f ("t");   f (1, "u");
```

Ответ:
```
void f (double d = 0, const char * s = 0);
                              // long int и float засчитывать за правильное
void f (const char *);    либо  void f (const char *, int i = 0);
```
Критерии: За каждую ошибку **–5**. Если приводятся более, чем две функции **–10**. Пропуск слова *const* в обеих функциях одновременно считать за одну ошибку.

Описать прототипы двух перегруженных функций *f()* из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
f ("a");    f (1.0);    f ();    f (0, 0);    f ("b", 1);    f (1, "0.1");
```

Ответ:
```
void f (double d = 0, const char * s = 0);
void f (const char *, int i = 0);
```
либо:
```
void f (double, const char * s = 0);
void f (const char * s = 0, int i = 0);
```
Критерии: За каждую ошибку **–5**. Если приводятся прототипы более, чем двух функций **–10**. Пропуск слова *const* в обеих функциях одновременно считать за одну ошибку.

4. Для объектов из задания **1** определить, тела каких конструкторов и деструкторов (возможно пустые, если они не определены явно) и в каком порядке будут исполнены при работе следующего фрагмента программы:

```
int main () { C c; B b = c; A a = c; }
```

Ответ: A (), B (), C (), B (B &), A (A &), ~A (), ~B (), ~C (), ~B (), ~A ().
Критерии: За каждую ошибку **–3**.

Для объектов из задания **1** определить, тела каких конструкторов и деструкторов (возможно пустые, если они не определены явно) и в каком порядке будут исполнены при работе следующего фрагмента программы:

```
int main () { struct D { A a; C c; }; D d; }
```

Ответ: A (), A (), B (), C (), D (), ~D (), ~C (), ~B (), ~A (), ~A ().
Критерии: За каждую ошибку **–3**.

Для объектов из задания **1** определить, тела каких конструкторов и деструкторов (возможно пустые, если они не определены явно) и в каком порядке будут исполнены при работе следующего фрагмента программы:

```
int main () { A a; class C { B b; A a; public: C (): b(), a (b) {} }; C c; }
```

Ответ: A (), A (), B (), A (A &), C (), ~C (), ~A (), ~B (), ~A (), ~A ().
Критерии: За каждую ошибку **–3**.

Для объектов из задания **1**, строящихся на основе исходных или (если надо) исправленных определений классов *A*, *B* и *C*, определить, тела каких конструкторов и деструкторов (возможно пустые, если они не определены явно) и в каком порядке будут исполнены при работе следующего фрагмента программы:

```
int main () { class D { C c; B b; public: D (): b(c) {} }; D d; }
```

Ответ: A (), B (), C (), B (B &), D (),~D (), ~B (), ~C (), ~B (), ~A ().
Критерии: За каждую ошибку **–3**.

5. Для класса комплексных чисел

```
template<class T> class complex { T r; T i; ... };
```

написать два варианта реализации операции "*", выполняющей умножение одного комплексного числа на другое методом класса и функцией-другом этого класса.

Ответ:

```
template<class T> complex<T> complex<T>::operator*
                        (const complex<T>& x)
{ complex<T> w; w.r = r * x.r - i * x.i;
                w.i = r * x.i + i * x.r; return w; }
template<class T> complex<T>            operator*
                        (const complex<T>& x, const complex<T>& y)
{ complex<T> w; w.r = x.r * y.r - x.i * y.i;
                w.i = x.r * y.i + x.i * y.r;   return w; }
```

Критерий: За отсутствие каждого примера или ошибки в нем: **-5.**

Для класса комплексных чисел

```
template<class T> class complex { T r; T i; ... };
```

написать реализацию операции "==", выполняющей сравнение двух комплексных чисел методом класса и функцией-другом класса.

Ответ:

```
template<class T> bool complex<T>::operator== (const complex<T>& x)
{ return r == x.r && i == x.i; }
template<class T> bool            operator==
                          (const complex<T>& x, const complex<T>& y)
{ return x.r == y.r && x.i == y.i; }
```

Критерий: За отсутствие каждого примера или ошибки в нем: **-5.**

Для класса комплексных чисел

```
template<class T> class complex { T r; T i; ... };
```

написать два варианта реализации префиксной операции увеличения "++", выполняющей увеличение на 1 действительной части комплексного числа методом класса и функцией-другом класса.

Ответ:
```
template<class T> & complex<T> complex<T>::operator++ ()
{ ++ r; return * this; }
template<class T> & complex<T>              operator++ (complex<T>& x)
{ ++ x.r; return x; }
```
Критерий: За отсутствие каждого примера или ошибки в нем: **-5.**

Для класса комплексных чисел

```
template<class T> class complex { T r; T i; ... };
```

написать два варианта реализации постфиксной операции уменьшения "--", выполняющей уменьшение на 1 действительной части комплексного числа методом класса и функцией-другом класса.

Ответ:
```
template<class T> complex<T> complex<T>::operator-- (int p)
{ complex<T> w; w = * this; -- r; return w; }
template<class T> complex<T> operator-- (complex<T>& x, int p)
{ complex<T> w; w = x; -- x.r; return w; }
```
Критерий: За отсутствие каждого примера или ошибки в нем: **-5.**

6. Дать определения абстрактного типа данных и абстрактного класса в языке Си++. Есть ли различия в этих понятиях? Можно ли создавать объекты и описывать конструкторы этих типов?

Ответ:
1. Абстрактный тип данных есть тип данных с полностью скрытым (инкапсулированным) внутренним устройством. Работа с объектами абстрактных типов данных осуществляется с помощью специальных функций (доступных методов). Описывать конструкторы и создавать объекты абстрактных типов данных можно.
2. Абстрактный класс – это класс, содержащий чистые виртуальные функции. Объекты абстрактных классов создавать нельзя, но можно строить объекты классов, производных от абстрактных. В абстрактных классах могут быть конструкторы и деструкторы.

Критерий: За отсутствие каждого из определений: **-5**.

Дать определение полиморфизма. Описать различные виды полиморфизма в Си++. Привести необходимые примеры фрагментов программ.

Ответ:
1. Полиморфизм – использование одинакового интерфейса (имени) для различных действий (объектов).
2. Статический полиморфизм действует на этапе компиляции. Примеры – перегрузка функций и операций:
```
void f (double);      void f (int);
void f (float);       void f (long);  void g () { ... f('a'); ... }
```
3. Динамический полиморфизм реализуется с помощью виртуальных функций и доступа к функциям по указателям, когда на этапе компиляции не удаётся однозначно определить вызываемую функцию, так как значение указателя в это время не известно.
```
class X {public:    virtual void f (int x) { cout << "X" << endl; } };
class Y: public X { public: void f (int y) { cout << "Y" << endl; } };
int main () { X * p = new Y b; ... p -> f (1); }
```
4. Параметрический (типовый) полиморфизм – использование шаблонов функций и классов.
```
template <class T> T max (T x, T y) { ... }
void f () { ... max (1, 2); ... }
template <class T> class Vector { T * p; int size;
                          public: explicit Vector (int);
};
Vector<int> X (20);
```

Критерий: За отсутствие каждого из определений: **-3**. За отсутствие соответствующего примера: **-1** за каждый пропуск**.**

Привести по одному примеру перегрузки бинарных, унарных префиксных и постфиксных операций с помощью функций-друзей и с помощью методов классов (всего 6 примеров). Указать особенности перегрузки операций присваивания. Можно ли изменить приоритет операции при её перегрузке?

Ответ:
1. Бинарная операция (сложение и присваивание):

Друг:
```
complex & operator+= (complex & a, const complex & b)
{ a.re += b.re; a.im += b.im; return a; }
```
Метод:
```
complex & complex::operator+= (const complex & a)
{ re += a.re, im += a.im; return * this; }
```
2. Унарная префиксная операция:

Друг:
```
complex & operator++ (complex & a)
{ ++ a.re; return a; }
```

Метод:
```
complex & complex::operator++ ()
{ ++ re; return * this; }
```

3. Унарная постфиксная операция:

Друг:
```
complex operator++ (complex & a, int b)
{ complex temp = a; ++ a.re; ++ a.im; return temp; }
```

Метод:
```
complex complex::operator++ (int b)
{ complex temp = * this; ++ re; ++ im; return temp; }
```

4. Операции присваивания могут перегружаться только нестатическими членами классов, но не функциями-друзьями.
5. При перегрузке операций не могут меняться их основные свойства – количество необходимых операндов, ассоциативность и приоритет.

Критерий: За отсутствие каждого из примеров: **-1**. За отсутствие описания особенностей операции присваивания: **-2**. Неправильный ответ на вопрос о свойствах операций: **-2**. На последний вопрос лаконичный ответ «нет» засчитывается как правильный.

Какие виды конструкторов могут присутствовать в определениях классов Си++? Каково их назначение? Привести примеры определения и использования конструкторов разных видов.

Ответ:

1. Конструктор умолчания (без параметров):
```
class complex { ... complex () { re = im = 0; } };
```
2. Конструктор преобразования (с одним параметром):
```
class complex { ... complex (const double a) { re = a; im = 0; } };
```
3. Конструктор копирования (с параметром-ссылкой на этот класс):
```
class complex { ... complex (complex & a) { re = re.a; im = re.im; } };
```
4. Обычный конструктор:
```
class complex { ... complex (double r, double i) { re = r; im = i; } };
```

Критерий: За отсутствие какого-либо из видов: **-2**. За отсутствие примера, если сам конструктор упомянут: **-1**.

7. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
struct X; void f(X & x, int n);
int const P = 1; int const Q = 1; int const R = 1;
struct X {    X()    { try    { f(*this, P);   cout <<  1 << endl; }
                      catch (X)             { cout <<  2 << endl; }
                      catch (int)           { cout <<  3 << endl; }
                    }
              X (X &)                       { cout <<  4 << endl; }
              ~X ()                         { cout <<  5 << endl; }
};
struct Y: X { Y ()            { f(*this, Q);
                                  cout <<  6 << endl; }
              Y (Y &)                       { cout <<  7 << endl; }
              ~Y ()                         { cout <<  8 << endl; }
};
void f(X & x, int n) { try    { if (n < 0) throw x;
                                if (n > 0) throw 1;
                                  cout <<  9 << endl;
                              }
                       catch (int)          { cout << 10 << endl; }
                       catch (X& a)         { cout << 11 << endl;
                                              f(a, R);
                                              cout << 12 << endl;
                                              throw;
                                            }
}
int main() {          try { Y a; }
                      catch (...)           { cout << 13 << endl;
                                              return 0;
                                            }
                                              cout << 14 << endl;
                                              return 0;
}
```

Ответ: В зависимости от значений констант P, Q и R:

```
P = -1, Q = -1, R = 1:  4 11 10 12  2  5  4 11 10 12  5 13  5
P =  0, Q = -1, R = 1:  9  1  4 11 10 12  5 13  5
P = -1, Q =  1, R = 0:  4 11  9 12  2  5 10  6  8  5 14
P =  1, Q = -1, R = 0: 10  1  4 11  9 12  5 13  5
```

Критерии: За каждую неверную (лишнюю или пропущенную) печать **-2**.

8. Есть ли синтаксические ошибки в тексте приведенной программы? Можно ли исправить описание класса, не вводя дополнительных членов и не убирая имеющиеся, чтобы программа стала верной? Если да, то как?

```
class A { static int i;
          static void f () { g (); cout << "f ()" << endl; }
                 void g () { if (i >= 0) i = -1, f ();
                                 cout << "g ()" << endl; }
};
int A::i = 1;
int main () {
     A::i = 1;
     A a;
     a.f();
     a.i = 0;
}
```

Ответ: 1. Заменить слово *"class"* словом *"struct"*, либо вставить после открывающей фигурной скобки определения класса слово *"public:"*.
2. Сделать метод *"g()"* статическим.

Критерий: За пропуск ошибок доступа к закрытым членам класса: **-5**. За пропуск ошибки вызова нестатического метода: **-5.** За указание ошибки там, где ее нет: **-5.**

Есть ли синтаксические ошибки в тексте приведенной программы? Можно ли исправить описание класса, не вводя дополнительных членов, чтобы программа стала верной? Если да, то как?

```
class A { static int i;
                 void f () { if (i >= 0) i = -1, g ();
                                 cout << "f ()" << endl; }
                 void g () { f (); cout << "g ()" << endl; }
};
int A::i = 1;
int main () {
     A::i = 1;
     const A a;
     a.f();
     a.i = 0;
}
```

Ответ: 1. Заменить слово *"class"* словом *"struct"*, либо вставить после открывающей фигурной скобки определения класса слово *"public:"*.
2. В определениях методов *"f()"* и *"g()"* вставить после пустого списка формальных параметров слово *"const"*.

Критерий: За пропуск ошибок доступа к закрытым членам класса: **-5**. За пропуск одной ошибки вызова неконстантного метода *"f()"*: **-6**. За пропуск одной ошибки вызова неконстантного метода *"g()"*: **-5.** За пропуск обеих таких ошибок в методах *"f()"* и *"g()"* **-8**. За указание ошибки там, где ее нет: **-5.**

Есть ли синтаксические ошибки в тексте приведенной программы? Можно ли исправить описание класса, не вводя дополнительных членов, чтобы программа стала верной? Если да, то как?

```
class A { static int i;
                 void f () const { if (i < 0) g (i);
                                      cout << "f ()" << endl; }
          void g (int & n) { i = n; f (); cout << "g ()" << endl; }
};
int A::i = 1;
int main () {
     const A a;
     a.g (2);
}
```

Ответ: 1. Заменить слово *"class"* словом *"struct"*, либо вставить после открывающей фигурной скобки определения класса слово *"public:"*.
2. В определении метода *"g()"* вставить перед типом формального параметра слово *"const"*.
3. В определении метода *"g()"* вставить после списка формальных параметров слово *"const"*.

Критерий: За пропуск ошибок доступа к закрытым членам класса: **-5**. За пропуск одной ошибки вызова метода *"g()"*: **-5.** За пропуск обеих таких ошибок: **-8**. За указание ошибки там, где ее нет: **-5.**

Есть ли синтаксические ошибки в тексте приведенной программы? Можно ли исправить описание класса, не вводя дополнительных членов, чтобы программа стала верной? Если да, то как?

```
class A { int i;
          void f (int & n) { if (i ==   n) cout << "f (& n)" << endl; }
          void f (int * n) { if (i == * n) cout << "f (* n)" << endl; }
};
int main () {
     const A a;
     a.f (a.i);
     a.f (& a.i);
}
```

Ответ:
1. Заменить слово *"class"* словом *"struct"*, либо вставить после открывающей фигурной скобки определения класса слово *"public:"*.
2. В обоих определениях методов *"f()"* вставить перед типом формального параметра слово *"const"*.
3. В обоих определениях метода *"f()"* вставить после списка формальных параметров слово *"const"*.

Критерий: За пропуск ошибок доступа к закрытым членам класса: **-5**. За пропуск одной ошибки вызова метода *"f()"*: **-5.** За пропуск обеих таких ошибок: **-8**. За указание ошибки там, где ее нет: **-5.**

9. Описать последовательность отождествления типов параметров перегружаемых функций для случая функций с одним параметром. Приведите примеры для каждого из шагов последовательности.

Ответ:
1. Точное отождествление (с учетом вариантов синтаксической записи):
```
void f (double);
void f (int);     void g ()     { ... f (1); ... } // f (int)
```
2. Безопасные расширения типов (*char*, *short*, *enum* --> *int (unsigned int)*; *bool* --> *int*; *float* --> *double*; *double* --> *long double*):
```
void f (float);
void f (long double); void g () {...f (1.0); ... } // f (long double )
```
3. Стандартные преобразования:
```
void f (char *);  class A { ... };
void f (A);       void g ()     { ... f (0); ... } // f (char *)
```
4. Пользовательские преобразования:
```
struct S { ... S (int);  operator int ();... };
void e (int);     void e (char *);
void f (S);       void f (char *);
                  void g (S &a) { ... e (a); ...   // e (int)
                                      f (1); ...   // f (S)
                                }
```

Критерий: За пропуск какого-либо одного из шагов или непонимание его сущности: **-4.** За пропуск других шагов или непонимание их сущности: **-2** за каждый шаг. Отсутствие примеров при указании самого шага: **-1** за каждый пропуск.

Описать особенности процесса отождествления типов параметров перегружаемых функций с несколькими параметрами. Приведите по одному примеру перегрузки функции с несколькими параметрами, в которых

- не удается найти ни одной подходящей функции,
- находится одна подходящая функция,
- находится несколько подходящих функций.

Ответ:
1. Отбор функций по количеству формальных параметров (с учетом параметров, имеющих значения по умолчанию, и многоточий)
2. Создание множеств пригодных функций при анализе каждого из параметров в отдельности
3. Параметры функций, соответствующие многоточию в списке формальных параметров, считаются подходящими во всех случаях.
4. Вычисление пересечения построенных множеств
5. Если в полученное пересечение входит ровно одна функция – она и есть искомый результат, в противном случае выдается сообщение об ошибке

Пример:
```
class X { public: X (long); X (char *); ...};
void f (X, long);   void f (X, double);
void g () {...   f (0, 0);   //  нет пригодной функции
                 f (1L, 1L); //  f(X, long)
                 f ("a", 1); //  неоднозначность при вызове
...}
```

Критерий: За пропуск любого шага или непонимание его сущности: **-2.** Отсутствие примера: **-2**.

Данные каких типов могут подвергаться динамическому и статическому приведению? В чём проявляется отличие выполнения динамического приведения типов для указателей и ссылок? Какие свойства указателей и ссылок влияют на это отличие? Приведите примеры фрагментов программы с использованием подобных операций.

Ответ: Динамическое и статическое приведение типов возможно для объектов родственных полиморфных классов, относящихся к одной иерархии классов. Статическое приведение возможно также для свободных указателей (*void **), которые могут преобразовываться в значения любых типов указателей, и *для преобразований между арифметическими типами* (отсутствие этого добавления не наказывается). Указатель может иметь нулевое значение, поэтому при динамическом приведении указателя в случае возникновения ошибки может возвращаться это нулевое значение, которое затем может быть проверено в программе. Ошибка при приведении ссылки всегда приводит к возбуждению исключительной ситуации *bad_cast*, так как никакого выделенного значения для ссылок не существует. Проверка правильности динамического приведения ссылок всегда выполняется перехватом исключительной ситуации.

Пример:
```
class A { ... };
class B: public A { ... };
class C: public A { ... };
void f (A * p, A & r)
{ if (B * pp = dynamic_cast<B *> (p))
        { ... // использование указателя pp
        }
  else { ... // указатель pp не принадлежит нужному типу
        }
B & pr = dynamic_cast<B &> (r);
   ...        // использование ссылки pr
}
void g ()
{ try { f (new B, * new B); // правильный вызов
        f (new C, * new C); // выход в перехватчик (C – из другой
                            // иерархии, основанной на том же
                            // базовом классе)
      }
  catch (bad_cast)
       { ... // обработка исключительной ситуации
       }
}
```

Критерий: За неверное понимание сути операций приведения типов: **-4**. За неверное понимание приведения типа в одном из случаев: **-4.** За неверное понимание сути динамического приведения типа и для указателей и для ссылок: **-6.** Отсутствие одного из примеров (здесь приведен обобщенный пример, эквивалентный сразу двум простым) при демонстрации правильного понимания сути: **-2.**

Описать последовательность действий при возбуждении исключительной ситуации. Привести пример фрагмента программы.

Ответ:
1. Создание временного объекта (копии исключительной ситуации)
2. Выход из ***try***-блока с освобождением памяти для локальных объектов (свёртка стека)
3. Выполнение составного оператора перехватчика (***catch***-блока) – статическая ловушка
4. Если статическая ловушка не сработала, выход в объемлющий блок с поиском подходящего обработчика (динамическая ловушка)
5. Если ни одна ловушка не сработала, выполнение функции *terminate()*.

Пример:
```
class A { char * p; int s; ... };
char & A::operator [] (int i)
{ if (i <  0) throw "A: размер < 0";
  if (i >= s) throw i;
  return p [i];
}
void g (int i)
{ ... try { ... A a (i), b (7);     ... b [i] = a [i - 1]; ... }
       catch (int n)               { ... }
       catch (const char * diagn) { ... }
}
```

Критерий: За пропуск любого шага или непонимание его сущности: **-2.** Отсутствие примера: **-2.**

10. Написать функцию *g()* с тремя параметрами: непустой контейнер-вектор типа *vector<int>*, непустой контейнер-список типа *list<int>*, целое число – шаг по первому контейнеру. Функция должна, последовательно проходя по списку от начала к концу, перезаписывать на место очередного его элемента соответствующий очередному шагу элемент вектора (сам вектор при этом не изменяется), а затем распечатывать элементы списка в обратном порядке. Функция возвращает количество изменённых элементов списка.

Решение:
```
typedef vector<int> V;
typedef   list<int> L;
int g (const V & vect, L & lst, int step)
{ V::  const_iterator vp = vect.begin ();  int t = 0;
```

```
  L::         iterator lp =  lst.begin ();
  do { * lp = * vp,    ++ t;
       ++lp;     vp += step;
     } while (vect.end () > vp && lp != lst.end ());
  L:: reverse_iterator rp = lst.rbegin ();
  while (rp != lst.rend ())
     { cout << * rp << ' ';
       ++ rp;
     }
  cout << endl;
  return t;
}
```

Критерий: За каждую существенную ошибку: **-3**.

Написать функцию *g()* с тремя параметрами: непустой и неизменяемый контейнер-список типа *list<long int>*, непустой контейнер-вектор типа *vector<long int>*, целое число – шаг по второму контейнеру. Функция должна копировать отрицательные элементы списка с шагом, равным 1, в уже имеющийся контейнер-вектор, от его начала к концу с шагом, равным третьему параметру, а затем распечатывать элементы вектора в прямом порядке. Функция возвращает количество измененных элементов вектора.

Решение:

```
typedef vector<long int> VL;
typedef   list<long int> LL;
int g (VL & vect, const LL & lst, int step)
{ VL::       iterator vp = vect.begin ();  int t = 0;
  LL:: const_iterator lp =  lst.begin ();
  do { if (* lp < 0) * vp = * lp,    ++ t;
       ++lp;     vp += step;
     } while (vect.end () > vp && lp != lst.end ());
  VL:: const_iterator rp = vect.begin ();
  while (rp != vect.end ())
     { cout << * rp << ' ';
       ++ rp;
     }
  cout << endl;
  return t;
}
```

Критерий: За каждую существенную ошибку: **-3**.

Написать функцию *g()* с тремя параметрами: непустой и неизменяемый контейнер-вектор типа *vector<double>*, непустой контейнер-список типа *list<double>*, целое число – шаг по первому контейнеру. Функция должна сравнивать элементы списка, выбираемыми от его начала с шагом, равным 1, с элементами вектора, выбираемыми от начала с шагом, равным третьему параметру. Если обнаруживается несовпадение очередной выбранной пары, то в список в текущем месте вставляется отсутствующий элемент. Изменённый список распечатывается в обратном порядке. Функция возвращает количество элементов, вставленных в список.

Решение:

```
typedef vector<double> V;
typedef   list<double> L;
int g (const V & vect, L & lst, int step)
{ V::  const_iterator vp = vect.begin ();
  L::         iterator lp =  lst.begin ();  int t = 0;
  do { if (* lp != * vp) lst.insert (lp, * vp), ++ t;
            ++lp;     vp += step;
     } while (vect.end () > vp && lp != lst.end ());
  L:: reverse_iterator rp = lst.rbegin ();
  while (rp != lst.rend ())
     { cout << * rp << ' ';
       ++ rp;
     }
  cout << endl;
  return t;
}
```

Критерий: За каждую существенную ошибку: **-3**.

Написать функцию *g()* с тремя параметрами: непустой и неизменяемый контейнер-вектор типа *vector<float>*, непустой контейнер-список типа *list<float>*, целое число – шаг по первому контейнеру. Функция должна исследовать элементы списка, выбираемые от его конца с шагом, равным 1, и элементы вектора, выбираемые от его начала с шагом, равным третьему параметру. Если обнаруживаются пары элементы разных знаков, то у текущего элемента списка должен меняться знак. Изменённый список распечатывается в прямом порядке. Функция возвращает общее количество неотрицательных элементов списка.

Решение:

```
typedef vector<float> V;
typedef   list<float> L;
int g (const V & vect, L & lst, int step)
{ V::  const_iterator vp = vect.begin ();
  L::reverse_iterator lp = lst.rbegin (); int t = 0;
  do { if (* lp * * vp < 0) * lp = - * lp;
          ++ lp;    vp += step;
     } while (vect.end () > vp && lp != lst.rend ());
  L:: iterator rp = lst.begin ();
  while (rp != lst.end ())
     { cout << * rp << ' ';
       if (* rp >= 0) t ++;
       ++ rp;
     }
  cout << endl;
  return t;
}
```

Критерий: За каждую существенную ошибку: **-3**.

2009 год
Экзамен

Вариант 1.
За каждую задачу — максимум 10 баллов.

1. Дана грамматика G:

$$S \to Aa \mid cSad \mid \varepsilon$$
$$cAad \to cBad$$
$$B \to c$$
$$cB \to cBc$$

(а) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.
Ответ: $k = \mathbf{0}$
(есть $S \to \varepsilon$ и S встречается в правой части правила $S \to cSad$)

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\Im$	$L(G) \in \Im$? (да/нет)
контекстно-свободные языки	**да**
контекстно-зависимые языки	**да**
языки типа 0	**да**
регулярные языки	**нет**

(в) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

Ответ: $L(G) = \{c^{n+k}\,(ad)^n \mid n \geq 1,\ k \geq 0\} \cup \{\varepsilon\}$

КРИТЕРИИ:

(а) нет ответа; неверный ответ : **–3**
(обоснование не обязательно)

(б) за каждый неверный или отсутствующий ответ: **–1 (т.е. всего до –4 за этот пункт)**
(обоснование не обязательно)

(в) описание отсутствует или с ошибками: **–3**

2. (а) Дать определение *недостижимого* символа.
(б) Эквивалентны ли данные КС-грамматики G_1 и G_2? Ответ обосновать.

G_1 :
$$S \to BAa$$
$$A \to Aa \mid a \mid \varepsilon$$
$$B \to b$$

G_2 :
$$S \to E \mid BAa$$
$$D \to dDd \mid c$$
$$E \to EaD \mid Ed$$
$$A \to Aa \mid ADE \mid a \mid \varepsilon$$
$$B \to DbE \mid b$$

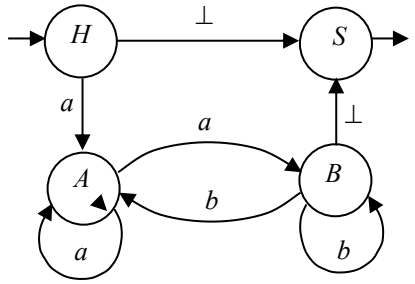
Ответ:

(**а**) символ $x \in T \cup N$ называется *недостижимым* в грамматике $G = \langle T, N, P, S \rangle$, если он не появляется ни в одной сентенциальной форме этой грамматики.

(**б**) **Да**, эквивалентны. Грамматика G_1 является результатом применения к грамматике G_2 алгоритма приведения КС-грамматики. Другое обоснование: $L = \{ba^n \mid n \geq 1\} = L(G_1) = L(G_2)$.

КРИТЕРИИ: (а) нет ответа; определение с ошибками: **–5**
(б) нет ответа; неверный ответ; нет обоснования; неверное обоснование: **–5**

3. По леволинейной грамматике G была построена диаграмма состояний $\mathcal{A}$:



(а) Восстановить по диаграмме $\mathcal{A}$ грамматику G:

(б) Детерминирован ли разбор по данной диаграмме ? Объясните ответ.

(в) **Если** разбор по диаграмме состояний $\mathcal{A}$ недетерминированный (т.е. конечный автомат $\mathcal{A}$ — недетерминированный), **то** с помощью соответствующего алгоритма построить эквивалентный детерминированный автомат $\mathcal{A}_D$, **иначе** — написать для $\mathcal{A}$ программу-анализатор на языке C++.

Ответ: (**а**) G: $S \to \perp \mid B\perp$
$B \to Bb \mid Aa$
$A \to Aa \mid Bb \mid a$

(**б**) Разбор недетерминирован, т.к. автомат $\mathcal{A}$ не является детерминированным: из состояния A возможен переход в более чем одно состояние (по символу a). [или: из состояния B возможен переход в более чем одно состояние (по символу b)]

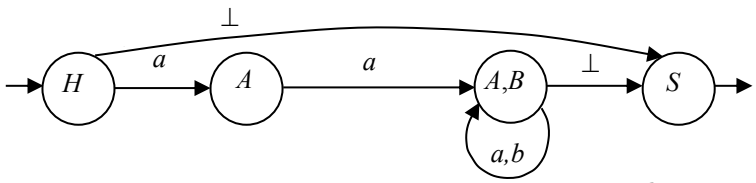
(**в**)

Построим таблицу переходов для $\mathcal{A}_D$:

	a	b	$\perp$
$\{ H \}$	$\{ A \}$	$\varnothing$	$\{ S \}$
$\{ A \}$	$\{ A, B \}$	$\varnothing$	$\varnothing$
$\{ A, B \}$	$\{ A, B \}$	$\{ A, B \}$	$\{ S \}$
$\{ S \}$	$\varnothing$	$\varnothing$	$\varnothing$

Начальное состояние: $\{ H \}$. Заключительное состояние: $\{ S \}$.

$\mathcal{A}_D$:



Допускаются другие способы построения, дающие в итоге такой же автомат $\mathcal{A}_D$ с точностью до обозначений вершин.

КРИТЕРИИ: (а) нет ответа; грамматика построена, но неэквивалентная или не соответствующая диаграмме : **–3**

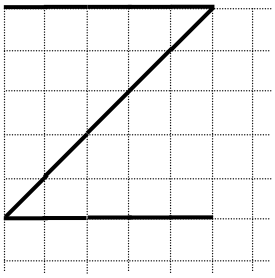
(б) нет ответа; неверный ответ; ошибочное объяснение : **–3**

(в) нет ответа или программа на C++ приведена;
неэквивалентный или недетерм. автомат построен : **–4**

4. Имеется клетчатый лист бумаги, бесконечный вправо и вниз. В левом верхнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита $\{a, b\}$: a — переместить перо на одну клетку вправо;

b — переместить перо по диагонали на одну клетку влево-вниз.

Например, по последовательности *aaaaabbbbbaaaaa* интерпретатор построит букву «Z» размера 5×5, изображенную на рисунке.



Построить грамматику с терминальным алфавитом $\{a, b\}$, описывающую буквы «Z» любого размера $n \times n$, $n \geq 1$. В грамматике должно быть **не более** 7 правил вывода (считая альтернативы).

Ответ: Описание буквы «Z» размера $n \times n$ — это цепочка $a^n b^n a^n$. Следовательно, требуется построить грамматику, порождающую язык $\{ a^n b^n a^n \mid n \geq 1 \}$: $S \to ABSa \mid Aba$

$$BA \to AB$$
$$Bb \to bb$$
$$Ab \to ab$$
$$Aa \to aa$$

КРИТЕРИИ: порождается пустая цепочка: **–5**

имеется лишняя (непустая) или недостающая цепочка: **–10**

за каждое лишнее правило: **–3** (количество правил в условии дано с запасом)

Замечание. Грамматика $S \to ABSa \mid ABa$
$BA \to AB$
$B \to b$
$A \to a$ ошибочная, так как порождает лишние цепочки, напр.: *ababaa*.

5. **(а)** Сформулируйте достаточное условие применимости метода рекурсивного спуска с учетом возможной ε-альтернативы (т.н. «канонический вид» КС-грамматики).

(б) Выполняется ли это условие для грамматики G ? Обоснуйте ответ.
G: $S \to aAS \mid bB \mid \varepsilon$
$A \to aAca \mid bBb$
$B \to cB \mid \varepsilon$

Ответ: (**а**) Достаточное условие: каждая группа правил с одинаковой левой частью имеет один из перечисленных ниже видов и выполняются дополнительные условия:

(1) либо $X \to \alpha$,
где $\alpha \in (T \cup N)^*$ и это единственное правило вывода для этого нетерминала;

(2) либо $X \to a_1\alpha_1 \mid a_2\alpha_2 \mid \ldots \mid a_n\alpha_n$,
где $a_i \in T$ для всех $i = 1, 2, \ldots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$, т. е. если для нетерминала X правил вывода несколько, то они должны начинаться с терминалов, причем все эти терминалы должны быть попарно различными;

(3) либо $X \to a_1\alpha_1 \mid a_2\alpha_2 \mid \ldots \mid a_n\alpha_n \mid \varepsilon$,
где $a_i \in T$ для всех $i = 1, 2, \ldots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$, и $first(X) \cap follow(X) = \varnothing$.

Замечание. Допускаются обозначения V_T , V_N для обозначения алфавитов терминальных и нетерминальных символов, и обозначение V для их объединения. $V = T \cup N$ $(V = V_T \cup V_N)$

(**б**) **Да**, т.к. все непустые альтернативы начинаются с попарно различных терминалов, $first(S) \cap follow(S) = \varnothing$, и $first(B) \cap follow(B) = \varnothing$.

[$first(S) = \{a, b\}$, $follow(S) = \varnothing$; $first(B) = \{c\}$, $follow(B) = \{b\}$]

КРИТЕРИИ: (а) нет ответа, ошибки в формулировке: **–5**

(б) нет ответа; неверный ответ; нет обоснования; ошибочное обоснование: **–5**

6. Описать грамматику, порождающую язык $L_{вх} = \{ d^n a^m a^n \mid m,n \geq 0 \}$, и вставить в нее действия вида ⟨*cout* << 'символ' ;⟩ так, чтобы для любых целых $m, n \geq 0$ в процессе анализа рекурсивным спуском входной цепочки $d^n a^m a^n$ печаталась бы цепочка $cb^n a^m$.

Ответ. Соглашение : запись ⟨*a*⟩ является сокращением ⟨*cout* << *'a '*;⟩

$L_{вх} = \{ d^n a^{m+n} \mid m,n \geq 0 \} = \{ d^n a^n a^m \mid m,n \geq 0 \}$ (входной язык). Грамматика для $L_{вх}$, к которой применим метода рек. спуска выглядит так:

$$S \to AB$$
$$A \to dAa \mid \varepsilon$$
$$B \to aB \mid \varepsilon$$

$L_ц = \{ cb^n a^m \mid m,n \geq 0 \}$ (целевой язык).

Грамматика с действиями:

$$S \to AB \qquad \textit{или :} \quad S \to \langle c \rangle AB$$
$$A \to dAa \langle b \rangle \mid \varepsilon \langle c \rangle \qquad A \to d \langle b \rangle Aa \mid \varepsilon$$
$$B \to a \langle a \rangle B \mid \varepsilon \qquad B \to a \langle a \rangle B \mid \varepsilon$$

КРИТЕРИИ:

Грамматика не порождает $L_{вх}$, или не генерирует $L_ц$: **0**

Порождает $L_{вх}$ и формально генерирует соответствующие цепочки из $L_ц$, но метод рек. спуска неприменим: **-5**

7. Дайте определение системы программирования. Перечислите основные компоненты классической системы программирования на базе ОС UNIX (ЯП Си). Кратко опишите назначение каждого из основных компонент.

Ответ:

Система программирования — это комплекс программных инструментов и библиотек, который поддерживает **весь** технологический цикл создания программного продукта (ПП).

[Как вариант: **Система программирования** — это комплекс программных средств (инструментов, библиотек), предназначенных для поддержки программного продукта на протяжении **всего** жизненного цикла этого программного продукта.]

Основные компоненты системы программирования: 1. **Транслятор** (проверяет синтаксическую правильность программ и переводит их с языка программирования на машинный язык, что и позволяет выполнить их на ЭВМ). 2. **Макрогенератор** или макропроцессор (работает непосредственно перед транслятором, используется для получения макрорасширения исходной программы). 3. **Редактор связей** или компоновщик (предназначен для связывания между собой (по внешним данным) объектных файлов, порождаемых компилятором, а также файлов библиотек, входящих в состав СП). 4. **Редактор текстов** (используется для составления и редактирования программ на языке программирования).
5. **Отладчик** (используется для проверочных запусков программ, локализации и исправления ошибок). 6. **Библиотеки** (облегчают работу программиста, предоставляя готовые фрагменты решения тех или иных подзадач программы, используются на этапе трансляции и исполнения).

КРИТЕРИИ:
- неверное определение СП: **–5**;
- есть только определение СП: **1** балл;
- отсутствие каждого компонента (кроме макрогенератора): **–3**;
- за каждое неверное (отсутствующее) описание: **–2**;
- каждая мелкая погрешность: **–1**.

Замечание. Макрогенератор может рассматриваться как часть транслятора или пакетного редактора текстов, поэтому допустимо его отсутствие в перечне компонентов . Вместо трансляторов могут упоминаться компиляторы и ассемблеры.

8. Каковы два основных способа подключения библиотек функций? Чем они отличаются? Назовите основные достоинства этих способов.

Ответ: Библиотеки функций могут подключаться статически и динамически. ДБ в отличие от СтБ подключаются к программе не во время компиляции программы с помощью редактора связей, а непосредственно в ходе её выполнения. На этапе компоновки программы редактор связей, встречая вызовы функций ДБ, вместо выполнения процедуры связывания формирует таблицу точек вызова функций ДБ для последующей операции динамического связывания (то есть компоновщик не помещает в целевую программу тела функций ДБ). Таким образом, отложенный процесс полной компоновки завершится уже на этапе выполнения целевой программы. Преимущества ДБ:

- не требуется включать в программу объектный код часто используемых функций, что существенно сокращает объем кода;

- различные программы, выполняемые в некоторой ОС, могут пользоваться кодом одной и той же библиотеки, содержащейся в ОП;
- изменения и улучшения функций библиотек сводится к обновлению только содержимого динамически подключаемых библиотек, а уже существующие тексты программ не требуют перекомпиляции (правда, этот же факт может оказаться недостатком, если при модификации функций каким-то образом меняется логика их работы, поэтому использование динамических библиотек накладывает определенные обязательства как на разработчика программы, так и на создателя библиотеки).

КРИТЕРИИ: - есть только ответ на первый вопрос: **1** балл;
- ответ не на вопрос: **0** баллов;
- неверный (или отсутствует) ответ на второй вопрос: **–5**;
- за каждое не упомянутое преимущество: –2;
- отсутствие преимуществ статических библиотек — не снижать (можно отметить относительную простоту реализации; устойчивость к модификациям — все равно все перекомпилируется и какие-то несоответствия со старым вариантом могут обнаружиться на этапе трансляции).

9. Записать результат перевода в ПОЛИЗ (постфиксную запись) фрагмента программы на языке Си:

```
b = i = 0;  do {S =(a < i*(a + b) ? b++ : ++i);} while (i < 100);
```

Ответ:

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
&b	&i	0	=	=	&S	a	i	a	b	+	*	<	20	!F	&b	++

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
22	!	&i	##	=	i	100	<	30	!F	6	!						

Соглашения: 1. Адреса ПОЛИЗа можно обозначать подчеркиванием (a = &a).
2. Префиксный ++ обозначен в ПОЛИЗе как ## .
3. Различение префиксных и постфиксных ++ разрешается на уровне интерпретации.
4. "Мусор" в стеке после выполнения присваивания игнорируется.

КРИТЕРИИ: - пропуск адреса в первом операнде операции(ий) присваивания: –3;
- пропуск адреса в операнде операции(ий) ++ : –3;
- неверный ПОЛИЗ первого оператора: –6;
- неверный ПОЛИЗ цикла (любые, не указанные выше ошибки): 0 баллов.

Замечание. Допустимы следующие вариации ПОЛИЗа в ответах:
(1) разрешается разворачивать унарные операции ++ и -- :
префиксные — ++x ==> &x, x, 1, +, = ;
постфиксные — x++ ==> &y, x, =, &x, x, 1, +, = ,y , где y — новый операнд;
(2) может использоваться операция «;» для выбрасывания «мусора» из стека, при трактовке присваивания как операции, оставляющей результат присваивания на вершине стека.
Соответственно изменяются адреса переходов в связи с (1) и/или (2).

10. Чем характерны машинно-зависимые оптимизирующие преобразования? Перечислите их основные типы, дав краткое пояснение. На каком этапе трансляции производится большая часть этих преобразований?

Ответ: При выполнении машинно-зависимой оптимизации компилятор принимает во внимание те или иные составляющие архитектуры вычислительной системы (особенности аппаратных и программных средств), на которой результирующая программа будет выполняться.
Основные типы м-з пр.: **1. Оптимальное распределение регистров процессора.**
2. Оптимизация передачи параметров в процедуры и функции (через регистры).
3. Оптимизация кода для процессоров, допускающих распараллеливание вычислений.
Пример: $a+b+c+d+e+f; \Rightarrow$
для одного потока обработки данных: $((((a+b)+c)+d)+e)+f;$
для двух потоков обработки данных: $((a+b)+c)+((d+e)+f);$
для трех потоков обработки данных: $(a+b)+(c+d)+(e+f);$
Машинно-зависимые преобразования производятся в основном над **результирующей объектной программой.**

КРИТЕРИИ: - нет ответа на первый вопрос: –3;
- только ответ на 1-й вопрос или только на последний: 1 балл;
- отсутствие каждого типа: –2;
- неверный (отсутствует) ответ на последний вопрос: –3;

Вариант 2.
За каждую задачу — максимум 10 баллов.

1. Дана грамматика G:

$$S \to A \mid \varepsilon$$
$$A \to cdAa \mid cB$$
$$cdcBa \to cdcDa$$
$$Da \to Daa$$
$$D \to a$$

(а) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.
Ответ: k = **1**
(третье и четвертое правила не удовл. типу 2)

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
регулярные языки	**нет**
контекстно-зависимые языки	**да**
контекстно-свободные языки	**да**
языки типа 0	**да**

(в) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

Ответ: $L(G) = \{(cd)^n\, c\, a^{n+k} \mid n \geq 1,\ k \geq 1\} \cup \{\varepsilon\}$

КРИТЕРИИ:

(а) нет ответа; неверный ответ : **–3**
(обоснование не обязательно)

(б) за каждый неверный или отсутствующий ответ: **–1 (т.е. всего до –4 за этот пункт)**
(обоснование не обязательно)

(в) описание отсутствует или с ошибками: **–3**

2. (а) Дать определение *бесплодного* символа.
(б) Эквивалентны ли данные КС-грамматики G_1 и G_2?
Ответ обосновать.

G_1 :
$$S \to BAaC \mid BC$$
$$A \to Aa \mid a \mid \varepsilon$$
$$B \to b$$

G_2 :
$$S \to E \mid BAa \mid AA$$
$$D \to bDc \mid a$$
$$E \to EaD \mid Ed$$
$$A \to AaE \mid ADE \mid \varepsilon$$
$$B \to DbE \mid bE$$

Ответ: (**а**) символ $A \in N$ называется *бесплодным* в грамматике $G = \langle T, N, P, S \rangle$, если множество $\{\alpha \in T^* \mid A \Rightarrow \alpha\}$ пусто.

(**б**) **Нет**. Грамматика G_1 после применения алгоритма приведения не будет содержать ни одного правила (S — бесплодный символ), поэтому $L(G_1)=\varnothing$. Результатом применения алгоритма приведения к грамматике G_2 будет грамматика :

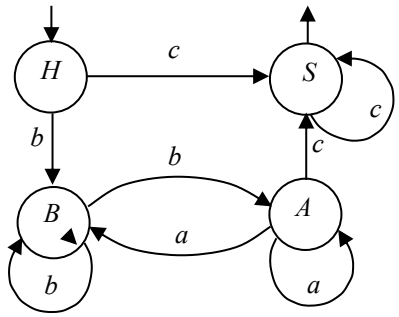
$$S \to AA$$
$$A \to \varepsilon.$$

Следовательно, $L(G_2)=\{\varepsilon\}$. Таким образом, $L(G_1) \neq L(G_2)$, т.к. $\varepsilon \in L(G_2)$, $\varepsilon \notin L(G_1)$.

КРИТЕРИИ: (а) нет ответа; определение с ошибками: **–5**
(б) нет ответа; неверный ответ; нет обоснования; неверное обоснование: **–5**

Замечание. Для обоснования неэквивалентности достаточно привести одну различающую цепочку**.**
Ответ **почти эквивалентны** считать верным.

3. По леволинейной грамматике G была построена диаграмма состояний $\mathcal{A}$:



(а) Восстановить по диаграмме $\mathcal{A}$ грамматику G:

(б) Детерминирован ли разбор по данной диаграмме ? Объясните ответ.

(в) **Если** разбор по диаграмме состояний $\mathcal{A}$ недетерминированный (т.е. конечный автомат $\mathcal{A}$ — недетерминированный), **то** с помощью соответствующего алгоритма построить эквивалентный детерминированный автомат $\mathcal{A}_D$, **иначе** — написать для $\mathcal{A}$ программу-анализатор на языке C++.

Ответ: (**а**) G: $S \to c \mid Sc \mid Ac$
$A \to Bb \mid Aa$
$B \to Aa \mid Bb \mid b$

(**б**) Разбор недетерминирован, т.к. автомат $\mathcal{A}$ не является детерминированным: из состояния A возможен переход в более чем одно состояние (по символу a). [или: т.к. из состояния B возможен переход в более чем одно состояние (по символу b)]

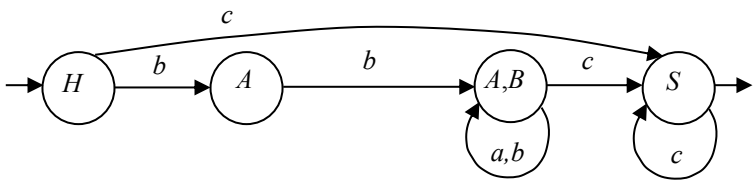
(**в**)

Построим таблицу переходов для $\mathcal{A}_D$:

	a	b	c
$\{ H \}$	$\varnothing$	$\{ B \}$	$\{ S \}$
$\{ B \}$	$\varnothing$	$\{ A, B \}$	$\varnothing$
$\{ A, B \}$	$\{ A, B \}$	$\{ A, B \}$	$\{ S \}$
$\{ S \}$	$\varnothing$	$\varnothing$	$\{ S \}$

Начальное состояние: $\{ H \}$. Заключительное состояние: $\{ S \}$.

$\mathcal{A}_D$:



Допускаются другие способы построения, дающие в итоге такой же автомат $\mathcal{A}_D$ с точностью до обозначений вершин.

КРИТЕРИИ: (а) нет ответа; грамматика построена, но неэквивалентная или не соответствующая диаграмме : **–3**

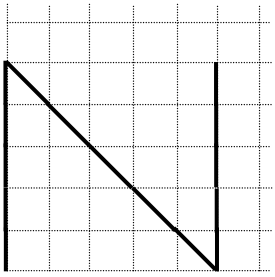
(б) нет ответа; неверный ответ; ошибочное объяснение : **–3**

(в) нет ответа или программа на C++ приведена;
неэквивалентный или недетерм. автомат построен : **–4**

4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита $\{c, f\}$: c — переместить перо на одну клетку вверх;

f — переместить перо по диагонали на одну клетку вправо-вниз.

Например, по последовательности *ccccfffffccccc* интерпретатор построит букву «N» размера 5×5, изображенную на рисунке.



Построить грамматику с терминальным алфавитом $\{c, f\}$, описывающую буквы «N» любого размера $n \times n$, $n \geq 1$. В грамматике должно быть **не более** 7 правил вывода (считая альтернативы).

Ответ: Описание буквы «N» размера $n \times n$ — это цепочка $c^n f^n c^n$. Следовательно, требуется построить грамматику, порождающую язык $\{ c^n f^n c^n \mid n \geq 1\}$: $S \to CFSc \mid Cfc$

$$FC \to CF$$
$$Ff \to ff$$
$$Cf \to cf$$
$$Cc \to cc$$

КРИТЕРИИ: порождается пустая цепочка: **–5**

имеется лишняя (непустая) или недостающая цепочка: **–10**

за каждое лишнее правило: **–3** (количество правил в условии дано с запасом)

Замечание. Грамматика $S \to CFSc \mid CFc$
$FC \to CF$
$F \to f$
$C \to c$ ошибочная, так как порождает лишние цепочки, напр.: $cfcfcc$.

5. (а) Сформулируйте достаточное условие применимости метода рекурсивного спуска с учетом возможной ε-альтернативы (т.н. «канонический вид» КС-грамматики).

(б) Выполняется ли это условие для грамматики G ? Обоснуйте ответ.

G: $S \to dDS \mid cBA \mid \varepsilon$
$D \to dD \mid aAc \mid b$
$A \to aAca \mid bB$
$B \to cB \mid \varepsilon$

Ответ: (а) Достаточное условие: каждая группа правил с одинаковой левой частью имеет один из перечисленных ниже видов и выполняются дополнительные условия:

(1) либо $X \to \alpha$,
где $\alpha \in (T \cup N)^*$ и это единственное правило вывода для этого нетерминала;

(2) либо $X \to a_1\alpha_1 \mid a_2\alpha_2 \mid \ldots \mid a_n\alpha_n$,
где $a_i \in T$ для всех $i = 1, 2, \ldots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$, т. е. если для нетерминала X правил вывода несколько, то они должны начинаться с терминалов, причем все эти терминалы должны быть попарно различными;

(3) либо $X \to a_1\alpha_1 \mid a_2\alpha_2 \mid \ldots \mid a_n\alpha_n \mid \varepsilon$,
где $a_i \in T$ для всех $i = 1, 2, \ldots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$, и $first(X) \cap follow(X) = \varnothing$.

Замечание. Допускаются обозначения V_T, V_N для обозначения алфавитов терминальных и нетерминальных символов, и обозначение V для их объединения. $V = T \cup N$ ($V = V_T \cup V_N$)

(б) **нет**, не выполняется: т.к. для правила $B \to cB \mid \varepsilon$ **не** выполняется условие $first(B) \cap follow(B) = \varnothing$.

[$first(B) = \{c\}$, $follow(B) = \{a,b,c\} \Rightarrow first(B) \cap follow(B) = \{c\} \neq \varnothing$]

КРИТЕРИИ: (а) нет ответа, ошибки в формулировке: **–5**

(б) нет ответа; неверный ответ; нет обоснования ; ошибочное обоснование: **–5**

6. Описать грамматику, порождающую язык $L_{вх} = \{ c^k b^m \mid m \geq k \geq 0 \}$, и вставить в нее действия вида ⟨*cout* << 'символ' ;⟩ так, чтобы для любых целых $m \geq k \geq 0$ в процессе анализа рекурсивным спуском входной цепочки $c^k b^m$ печаталась бы цепочка $ad^k b^{m-k}$.

Ответ. Соглашение : запись ⟨*a*⟩ является сокращением ⟨*cout* << *'a '*;⟩

$L_{вх} = \{ c^k b^m \mid m \geq k \geq 0 \} = \{ c^k b^k b^{m-k} \mid m \geq k \geq 0 \}$ (входной язык). Грамматика для $L_{вх}$, к которой применим метод рек. спуска, выглядит так:

$$S \to AB$$
$$A \to cAb \mid \varepsilon$$
$$B \to bB \mid \varepsilon \ .$$

$L_ц = \{ ad^k b^{m-k} \mid m \geq k \geq 0 \}$ (целевой язык).

Грамматика с действиями:

$S \to AB$ *или :* $S \to \langle a \rangle AB$

$A \to cAb \langle d \rangle \mid \varepsilon \langle a \rangle$ $A \to c \langle d \rangle Ab \mid \varepsilon$

$B \to b \langle b \rangle B \mid \varepsilon$ $B \to b \langle b \rangle B \mid \varepsilon$

КРИТЕРИИ:

Грамматика не порождает $L_{вх}$, или не генерирует $L_ц$: **0**

Порождает $L_{вх}$ и формально генерирует соответствующие цепочки из $L_ц$,
но метод рек. спуска неприменим: **-5**

7. Перечислите этапы технологического цикла создания программного продукта. Кратко охарактеризуйте каждый этап, указав назначение и результат его прохождения.

Ответ:

ЭТАП	***ХАРАКТЕРИСТИКА***
Анализ (определение) требований **Постановка задачи**	***внешние спецификации ПП***
Проектирование: - структуры ПП **- отдельных модулей**	***схема иерархии и внешняя спецификация отдельных модулей;*** ***выбор алгоритма работы каждого модуля и способ внутреннего представления данных***
Написание текста программ (кодирование) Компоновка или интеграция программного комплекса	***алгоритм записывается на каком-либо алгоритмическом языке***
Верификация, **тестирование;** **отладка**	***проверка программы на правильность, выявление наличия ошибок;*** ***локализация и исправление ошибок***
Документирование Внедрение Тиражирование **Окончательная сдача ПП**	***написание текстовой документации, инструкций для пользователей и т.п.***

Сопровождение (повторяющее все предыдущие этапы) и **эксплуатация ПП**	***попытка приспособить ПП к измененным целям и исправление не выявленных на более ранних этапах ошибок***

КРИТЕРИИ: - за каждый пропущенный (крупный) этап, кроме последнего: –2;
- за каждое неверное или отсутствующее пояснение: –1;
- за отсутствие последнего этапа: –1.

8. Перечислите основные типы библиотек, используемых в современных системах программирования (не путать со способами подключения!). Дайте их краткую характеристику.

Ответ: 1. Библиотеки функций (или подпрограмм). 2. Библиотеки классов. 3. Библиотеки компонент.

Библиотеки функций во многом определяют возможности системы программирования в целом. Различают *библиотеки для языков программирования* (например, функции ввода-вывода, работа со строками) и *библиотеки для решения задач в конкретной проблемной области* (например, функции, реализующие алгоритмы линейной алгебры). Библиотеки функций представляют собой откомпилированные объектные модули.

Библиотеки классов - для СП, базирующихся на ООЯП. Все их классы должны быть написаны на том же ЯП, на котором пишется программа, куда интегрируются библиотечные классы.

В библиотеке классов различают *конкретные классы*, *абстрактные классы, шаблоны классов*.

Библиотеки классов включаются в программу на этапе компиляции и компилируются со всей программой вместе.

Библиотеки компонент - это библиотека готовых откомпилированных программных модулей, предназначенных для использования в качестве составной части программы, и которыми можно манипулировать во время разработки программы. Компоненты бывают локальные (находящиеся на той же ЭВМ, где создается ПП) и распределенные (расположенные на сервере и доступные по сети ЭВМ).

Примеры технологий, использующих библ. компонент: CORBA, COM, DCOM, ActiveX, Java Beans

КРИТЕРИИ: - есть только перечисление типов библиотек: 1 балл;
- за каждую отсутствующую (неверную) характеристику: –3.

Замечание. За отсутствие упоминаний о распределенных объектах и компонентах не наказывать.

9. Записать результат перевода в ПОЛИЗ (постфиксную запись) фрагмента программы на языке Си:

```
for( i = 0; i < n; i++ ){ a = b - c - d; ++b; }
```

Ответ:

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&i	0	=	i	n	<	26	!F	15	!	&i	++	4	!	&a	b	c

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
-	d	-	=	&b	##	11	!											

Соглашения: 1. Адреса ПОЛИЗа можно обозначать подчеркиванием (a = &a).
2. Префиксный ++ обозначен в ПОЛИЗе как ## .
3. Различение префиксных и постфиксных ++ разрешается на уровне интерпретации.
4. "Мусор" в стеке после выполнения присваивания игнорируется.

КРИТЕРИИ: - пропуск адреса в первом операнде операции(ий) присваивания: –3;
- пропуск адреса в операнде операции(ий) ++ : –3;
- неверный ПОЛИЗ первого оператора: –6;
- неверный ПОЛИЗ цикла (любые, не указанные выше ошибки): 0 баллов.

Замечание. Допустимы следующие вариации ПОЛИЗа в ответах:

(1) разрешается разворачивать унарные операции ++ и -- :
префиксные — ++x ==> &x, x, 1, +, = ;
постфиксные — x++ ==> &y, x, =, &x, x, 1, +, = ,y , где y — новый операнд;

(2) может использоваться операция «;» для выбрасывания «мусора» из стека, при трактовке присваивания как операции, оставляющей результат присваивания на вершине стека.

Соответственно изменяются адреса переходов в связи с (1) и/или (2).

10. Перечислите основные типы машинно-независимых оптимизирующих преобразований, конкретизируя, что под ними понимается. На каком этапе трансляции производится большая часть этих преобразований?

Ответ: **1. Удаление недостижимого кода**.

2. Оптимизация линейных участков программы.

а) Удаление бесполезных присваиваний.

б) Исключение избыточных вычислений.

в) Свертка объектного кода (выполнение во время компиляции тех операций исходной программы, для которых значения операндов уже известны).

г) Перестановка операций (для дальнейшей свертки или оптимизации вычислений).

д) Арифметические преобразования (на основе алгебраических и логических тождеств).

е) Оптимизация вычисления логических выражений.

3. Подстановка кода функции вместо ее вызова в объектный код.

4. Оптимизация циклов.

а) Вынесение инвариантных вычислений из циклов.

б) Замена операций с индуктивными (образующими арифметическую прогрессию) переменными (как правило, умножения на сложение).

в) Слияние и развертывание циклов.

Машинно-независимые преобразования исходной программы производятся **в форме ее внутреннего представления.**

КРИТЕРИИ:
- только ответ на последний вопрос: 1 балл;
- отсутствие каждого типа (арабская цифра): –3;
- за каждую неточность внутри: –1 (не больше -3 для одной арабской цифры);
- неверный (отсутствует) ответ на последний вопрос: –3.

Вариант 3.
За каждую задачу — максимум 10 баллов.

1. Дана грамматика

G: $S \to Bc \mid Sa \mid \varepsilon$
$B \to Ad$
$A \to d$
$A \to bB$

(а) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа $k+1$.
Ответ: $k = 2$ ($C \to bB$ не удовлетвор. определению леволинейной грамматики, $B \to Cd$ не удовл. опре-делению праволинейной)

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные языки	**да**
контекстно-зависимые языки	**да**
регулярные языки	**нет**
языки типа 0	**да**

(в) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

Ответ $L(G) = \{ b^n d^{n+2} ca^k \mid n \geq 0, k \geq 0 \} \cup \{ a^k \mid k \geq 0 \}$

КРИТЕРИИ:

(а) нет ответа; неверный ответ : **–3**
(обоснование не обязательно)

(б) за каждый неверный или отсутствующий ответ: **–1 (т.е. всего до –4 за этот пункт)**
(обоснование и формула языка не обязательны)

(в) описание отсутствует или с ошибками: **–3**

2. (а) Дать определение *приведенной* КС-грамматики.
(б) Эквивалентны ли данные КС-грамматики G_1 и G_2?
Ответ обосновать.

G_1 : $S \to BA$
$A \to bAa \mid a \mid \varepsilon$
$B \to b \mid \varepsilon$

G_2 : $S \to E \mid BA$
$D \to dDd \mid c$
$E \to ED \mid Ed$
$A \to bAa \mid ADE \mid a \mid \varepsilon$
$B \to DbE \mid b$

Ответ: (**а**) КС-грамматика G называется *приведенной,* если в ней нет недостижимых и бесплодных символов.

(**б**) Нет. Грамматика G_2 после приведения:

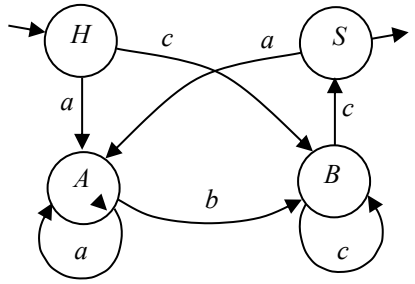
$S \to BA$
$A \to bAa \mid a \mid \varepsilon$
$B \to b$

Следовательно, $L(G_1) \neq L(G_2)$, т.к. $\varepsilon \notin L(G_2)$, $\varepsilon \in L(G_1)$.

КРИТЕРИИ: (а) нет ответа; определение с ошибками: **–5**
(б) нет ответа; неверный ответ; нет обоснования; неверное обоснование: **–5**

Замечание. Для обоснования неэквивалентности достаточно привести одну различающую цепочку**.**
Ответ **почти эквивалентны** считать верным.

3. По леволинейной грамматике G была построена диаграмма состояний $\mathcal{A}$:



(а) Восстановить по диаграмме $\mathcal{A}$ грамматику G:

(б) Детерминирован ли разбор по данной диаграмме ? Объясните ответ.

(в) **Если** разбор по диаграмме состояний $\mathcal{A}$ недетерминированный (т.е. конечный автомат $\mathcal{A}$ — недетерминированный), **то** с помощью соответствующего алгоритма построить эквивалентный детерминированный автомат $\mathcal{A}_D$, **иначе** — написать для $\mathcal{A}$ программу-анализатор на языке C++.

Ответ: (**а**) G: $S \to Bc$

$B \to Bc \mid Ab \mid c$

$A \to Aa \mid Sa \mid a$

(**б**) Разбор недетерминирован, т.к. автомат $\mathcal{A}$ не является детерминированным: из состояния B возможен переход в более чем одно состояние по символу c.

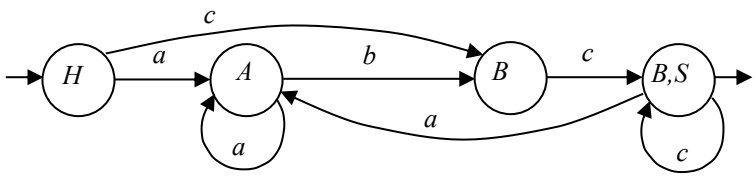
(**в**)

Построим таблицу переходов для $\mathcal{A}_D$:

	a	b	c
$\{ H \}$	$\{ A \}$	$\varnothing$	$\{ B \}$
$\{ A \}$	$\{ A \}$	$\{ B \}$	$\varnothing$
$\{ B \}$	$\varnothing$	$\varnothing$	$\{B, S \}$
$\{B, S \}$	$\{ A \}$	$\varnothing$	$\{B, S \}$

Начальное состояние: $\{ H \}$. Заключительное состояние: $\{ B, S \}$.

$\mathcal{A}_D$:



Допускаются другие способы построения, дающие в итоге такой же автомат $\mathcal{A}_D$ с точностью до обозначений вершин.

КРИТЕРИИ: (а) нет ответа; грамматика построена, но неэквивалентная или не соответствующая диаграмме : **–3**

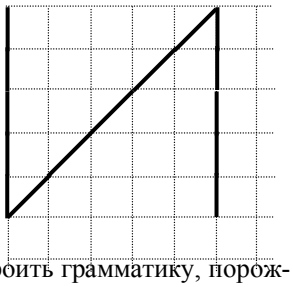
(б) нет ответа; неверный ответ; ошибочное объяснение : **–3**

(в) нет ответа или программа на C++ приведена;
неэквивалентный или недетерм. автомат построен : **–4**

4. Имеется клетчатый лист бумаги, бесконечный вправо и вниз. В левом верхнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита { *e* , *d* }: *e* — переместить перо на одну клетку вниз;

d — переместить перо по диагонали на одну клетку вправо-вверх.

Например, по последовательности *eeeeeddddeeeee* интерпретатор построит букву «И» размера 5×5, изображенную на рисунке.



Построить грамматику с терминальным алфавитом {*e*, *d*}, описывающую буквы «И» любого размера *n*×*n*, *n* ≥1. В грамматике должно быть **не более** 7 правил вывода (считая альтернативы).

Ответ: Описание буквы «И» размера *n*×*n* — это цепочка $e^n d^n e^n$. Следовательно, требуется построить грамматику, порождающую язык $\{ e^n d^n e^n \mid n \geq 1 \}$: $S \to EDSe \mid Ede$

$$DE \to ED$$
$$Dd \to dd$$
$$Ed \to ed$$
$$Ee \to ee$$

КРИТЕРИИ: порождается пустая цепочка: **–5**

имеется лишняя (непустая) или недостающая цепочка: **–10**

за каждое лишнее правило: **–3** (количество правил в условии дано с запасом)

Замечание. Грамматика $S \to EDSe \mid EDe$
$DE \to ED$
$D \to d$
$E \to e$ ошибочная, так как порождает лишние цепочки, напр.: *ededee*.

5. (а) Сформулируйте достаточное условие применимости метода рекурсивного спуска с учетом возможной ε-альтернативы (т.н. «канонический вид» КС-грамматики).

(б) Выполняется ли это условие для грамматики *G* ? Обоснуйте ответ.
G: $S \to aAeS \mid bB \mid \varepsilon$
$A \to aAd \mid bC \mid aC$
$C \to cC \mid \varepsilon$
$B \to eS \mid d$

Ответ: (а) Достаточное условие: каждая группа правил с одинаковой левой частью имеет один из перечисленных ниже видов и выполняются дополнительные условия:

(1) либо $X \to \alpha$,
где $\alpha \in (T \cup N)^*$ и это единственное правило вывода для этого нетерминала;

(2) либо $X \to a_1\alpha_1 \mid a_2\alpha_2 \mid \ldots \mid a_n\alpha_n$,
где $a_i \in T$ для всех $i = 1, 2, \ldots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$, т. е. если для нетерминала *X* правил вывода несколько, то они должны начинаться с терминалов, причем все эти терминалы должны быть попарно различными;

(3) либо $X \to a_1\alpha_1 \mid a_2\alpha_2 \mid \ldots \mid a_n\alpha_n \mid \varepsilon$,
где $a_i \in T$ для всех $i = 1, 2, \ldots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$, и $first(X) \cap follow(X) = \varnothing$.

Замечание. Допускаются обозначения V_T , V_N для обозначения алфавитов терминальных и нетерминальных символов, и обозначение *V* для их объединения. $V = T \cup N$ ($V = V_T \cup V_N$)

(б) нет, не выполняется: для правила $A \to aAd \mid bC \mid aC$, имеющего вид (2), нарушено дополнительное условие, т.к. $first(aAd) \cap first(aC) = \{a\} \neq \varnothing$ (две альтернативы начинаются одинаковыми терминалами).

КРИТЕРИИ: (а) нет ответа, ошибки в формулировке: **–5**

(б) нет ответа; неверный ответ; нет обоснования ; ошибочное обоснование: **–5**

6. Описать грамматику, порождающую язык $L_{вх} = \{ c^n d^m \mid m \geq k \geq 0 \}$, и вставить в нее действия вида ⟨*cout* << 'символ' ;⟩ так, чтобы для любых целых $m \geq n \geq 0$ в процессе анализа рекурсивным спуском входной цепочки $c^n d^m$ печаталась бы цепочка $b^n a d^{m-n}$.

Ответ. Соглашение : запись ⟨*a*⟩ является сокращением ⟨*cout* << *'a '*;⟩

$L_{вх} = \{ c^n d^m \mid m \geq n \geq 0 \} = \{ c^n d^n d^{m-n} \mid m \geq n \geq 0 \}$ (входной язык). Грамматика для $L_{вх}$, к которой применим метод рек. спуска, выглядит так:

$$S \to AB$$
$$A \to cAd \mid \varepsilon$$
$$B \to dB \mid \varepsilon$$

$L_ц = \{ b^n a d^{m-n} \mid m \geq n \geq 0 \}$ (целевой язык).

Грамматика с действиями:

$S \to A\langle a\rangle B$	*или :* $S \to A\langle a\rangle B$	*или :* $S \to AB$
$A \to cAd\langle b\rangle \mid \varepsilon$	$A \to c\langle b\rangle Ad \mid \varepsilon$	$A \to c\langle b\rangle Ad \mid \varepsilon\langle a\rangle$
$B \to d\langle d\rangle B \mid \varepsilon$	$B \to d\langle d\rangle B \mid \varepsilon$	$B \to d\langle d\rangle B \mid \varepsilon$

КРИТЕРИИ:

Грамматика не порождает $L_{вх}$, или не генерирует $L_ц$: **0**

Порождает $L_{вх}$ и формально генерирует соответствующие цепочки из $L_ц$,
но метод рек. спуска неприменим: **-5**

7. Что такое интегрированная среда разработки? Каковы преимущества (приведите три причины) использования ИСР по сравнению с классической СП?

Ответ: Современная **ИСР** — комплекс интегрированных программных средств, поддерживающих полный жизненный цикл ПП. Основные преимущества:

- ИСР объединяет в себе возможности текстовых редакторов исходных текстов программ, отладчиков и командный язык компиляции. Программист, не прерывая работу с ИСР, может в удобной форме воспользоваться услугами любого, входящего в ее состав компонента;
- ИСР обеспечивает многооконный интерфейс с поддержкой режима "буксировки" фрагментов текста мышкой (drag&drop);
- в текстовом редакторе ИСР благодаря интеграции с компилятором реализованы а) синтаксическая подсветка элементов языка (выделение цветом разных лексем), б) дополнение кода, интерактивная подсказка, в) всплывающие подсказки об атрибутах идентификаторов, если на них установить курсор, г) отображение ошибок, обнаруженных на этапе компиляции, в тексте программы;
- благодаря интеграции с отладчиком в текстовом редакторе ИСР реализованы, например, а) отображение контрольных точек останова при отладке, б) отображение текущего значения объекта, при наведении курсора на идентификатор, в) выполнение программы до строки, в которой в редакторе стоит курсор, г) выделение выполняемой в данный момент строки;
- часто ИСР поддерживает репозиторий прогр. проекта, являющийся основой ИСР, который обеспечивает хранение информации о проекте на всех этапах его жизненного цикла, версий проекта и его отдельных компонентов, синхронизацию поступления информации от различных разработчиков при групповой разработке, контроль данных о проекте на полноту и непротиворечивость;
- как правило, ИСР содержит графические средства анализа и проектирования ПП;

.....

КРИТЕРИИ: - есть только определение ИСР: **1 балл**;
- безошибочно указано только одно из достоинств ИСР: **–6**;
- безошибочно указаны только два из достоинств ИСР: **–3.**

Замечание. Допускается «описательное» определение ИСР через свойства, например: «объединяет в себе возможности текстовых редакторов и командного языка компилятора, а также некоторых других компонентов системы программирования, например, редактора связей и отладчика».

8. Перечислите основные критерии проектирования стандартных библиотек. Кратко прокомментируйте каждый из критериев.

Ответ:

- **Общезначимость** содержимого (предоставлять программисту нетривиальные средства, на которые он может рассчитывать, заботясь о переносимости программ, например, средства работы со списками, функции сортировки, потоки ввода/вывода;)

- **Эффективность (**предоставлять функции, которые не могут быть написаны оптимально для всех вычислительных систем на данном языке программирования, например, функции вычисления квадратного корня `sqrt()` или пересылок блоков памяти `memmove()`;)

- **Безопасность** (использование библиотеки не должно провоцировать ошибки, а наоборот, снижать их вероятность)

- **Завершенность** (библиотека должна обеспечить достаточную функциональность, чтобы ни у кого не возникало желания что-то заменить или доопределить)

- **Сочетаемость с базовыми типами данных (**предоставлять основу для расширения собственных возможностей, в частности, соглашения и средства поддержки, позволяющие обеспечить операции для данных, имеющих определяемые пользователями типы, в том же стиле, в котором обеспечиваются операции для встроенных типов (например, ввод/вывод);**)**

- Может служить **фундаментом** для создания других библиотек

- **Поддержка свойств языка**, (например, управление памятью и предоставление информации об объектах во время выполнения программ);

- **Предоставление информации** о зависящих от реализации аспектах языка, например, о максимальных размерах целых значений;

КРИТЕРИИ: - указаны и верно прокомментированы не менее 5 критериев: 10 баллов;
- если менее 5, то отсутствие (ошибочность) любого критерия: –2 (всего до –10);
- отсутствие каждого соответствующего критерию краткого комментария: –1.

9. Записать результат перевода в ПОЛИЗ (постфиксную запись) фрагмента программы на языке Си:

```
while (a < b) {S += a < 0 ? -a : b++; b = b - a - 1;}
```

Ответ:

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	b	<	28	!F	&S	a	0	<	16	!F	a	@	18	!	&b	++

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
+=	&b	b	a	-	1	-	=	1	!									≀

Соглашения: 1. Адреса ПОЛИЗа можно обозначать подчеркиванием (a = &a).
2. Префиксный ++ обозначен в ПОЛИЗе как ## , унарный минус — @.
3. Различение префиксных и постфиксных ++ разрешается на уровне интерпретации.
4. "Мусор" в стеке после выполнения присваивания игнорируется.

КРИТЕРИИ: - пропуск адреса в первом операнде операции(ий) присваивания: –3;
- пропуск адреса в операнде операции(ий) ++ : –3;
- неверный ПОЛИЗ первого оператора: –6;
- неверный ПОЛИЗ цикла (любые, не указанные выше ошибки): 0 баллов.

Замечание. Допустимы следующие вариации ПОЛИЗа в ответах:

(1) разрешается разворачивать унарные операции ++ и -- :
префиксные — ++x ==> &x, x, 1, +, = ;
постфиксные — x++ ==> &y, x, =, &x, x, 1, +, = ,y , где y — новый операнд;

(2) может использоваться операция «;» для выбрасывания «мусора» из стека, при трактовке присваивания как операции, оставляющей результат присваивания на вершине стека.

Соответственно изменяются адреса переходов в связи с (1) и/или (2).

10. Какие участки программы называют линейными? Что такое профилировщик? Кратко опишите его назначение.

Ответ: Линейные участки кода - фрагменты программы, все операторы которых выполняются в порядке их следования в тексте (нет передачи управления).

Профилировщик - компонент СП, который строит профиль программы (где выделены линейные участки кода) с указанием того, какой процент времени они выполняются тот или иной фрагмент кода.
Используется для более эффективной оптимизации программы, т.к. обычно 90% времени выполнения программы тратится на 10% ее кода, зная которые можно не оптимизировать оставшиеся 90% и избежать возможных ошибок при оптимизации, а также при отладке программ. (Процентное соотношение указано приблизительно).

КРИТЕРИИ: - неверный (отсутствует) ответ на первый вопрос: –4;
- неверный (отсутствует) ответ на второй вопрос: –4;
- неверный (отсутствует) ответ на третий вопрос: –4;

Замечание. Все три ответа могут быть сформулированы "своими словами". .

Вариант 4.
За каждую задачу — максимум 10 баллов.

1. Дана грамматика

G: $S \to Bcd$
$S \to \varepsilon$
$B \to Ca$
$C \to a$
$C \to Bbc$

(а) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.
Ответ: k = **3** (удовл. определению леволинейной грамматики)

(б) Каким из перечисленных классов принадлежит язык $L(G)$?
Ответ:

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные языки	**да**
контекстно-зависимые языки	**да**
языки типа 0	**да**
регулярные языки	**да**

(в) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

Ответ: $L(G) = \{\, aa\,(bca)^n\, cd \mid n \geq 0 \,\} \cup \{\, \varepsilon \}$

КРИТЕРИИ:

(а) нет ответа; неверный ответ : **–3**
(обоснование не обязательно)

(б) за каждый неверный или отсутствующий ответ: **–1 (т.е. всего до –4 за этот пункт)**
(обоснование и формула языка не обязательны)

(в) описание отсутствует или с ошибками: **–3**

2. (а) Можно ли любой КС-язык описать неукорачивающей КС-грамматикой ? Поясните ответ.

(б) Эквивалентны ли данные КС-грамматики G_1 и G_2 ? Ответ обосновать.

G_1 : $S \to AS \mid a$
$A \to b$

G_2 : $S \to ADS \mid a$
$A \to CC \mid bC$
$D \to DC \mid \varepsilon$
$C \to CDC \mid D$
$E \to Eb$

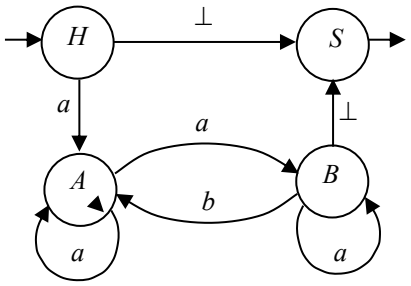
Ответ: (а) Да, можно. Существует алгоритм удаления правил с пустой правой частью для КС-грамматик, который по любой КС-грамматике строит эквивалентную неукорачивающую КС-грамматику.

(б) Эквивалентны. Грамматика G_1 получается из G_2 применением алгоритма удаления пустых правых частей.

КРИТЕРИИ: (а) нет ответа; неверный ответ; нет обоснования; неверное обоснование: **–5** ;
(б) нет ответа; нет обоснования; неверное обоснование: **–5**.

Замечание. Ссылку на утверждение «для любой КС-грамматики существует эквивалентная неукорачивающая КС-грамматика» тоже можно считать обоснованием.

3. По леволинейной грамматике G была построена диаграмма состояний $\mathcal{A}$:



(а) Восстановить по диаграмме $\mathcal{A}$ грамматику G:

(б) Детерминирован ли разбор по данной диаграмме ? Объясните ответ.

(в) **Если** разбор по диаграмме состояний $\mathcal{A}$ недетерминированный (т.е. конечный автомат $\mathcal{A}$ — недетерминированный), **то** с помощью соответствующего алгоритма построить эквивалентный детерминированный автомат $\mathcal{A}_D$, **иначе** — написать для $\mathcal{A}$ программу-анализатор на языке C++.

Ответ: (**а**) G: $S \to \perp \mid B\perp$
$B \to Ba \mid Aa$
$A \to Aa \mid Bb \mid a$

(**б**) Разбор недетерминирован, т.к. автомат $\mathcal{A}$ не является детерминированным: из состояния A возможен переход в более чем одно состояние (по символу a).

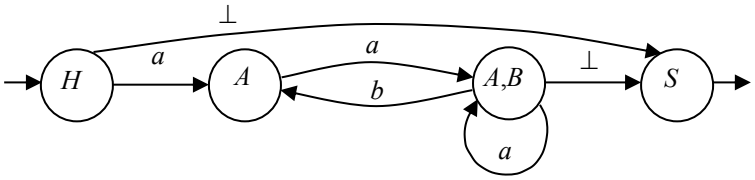
(**в**)

Построим таблицу переходов для $\mathcal{A}_D$:

	a	b	$\perp$
$\{ H \}$	$\{ A \}$	$\varnothing$	$\{ S \}$
$\{ A \}$	$\{ A, B \}$	$\varnothing$	$\varnothing$
$\{ A, B \}$	$\{ A, B \}$	$\{ A \}$	$\{ S \}$
$\{ S \}$	$\varnothing$	$\varnothing$	$\varnothing$

Начальное состояние: $\{ H \}$. Заключительное состояние: $\{ S \}$.

$\mathcal{A}_D$:



Допускаются другие способы построения, дающие в итоге такой же автомат $\mathcal{A}_D$ с точностью до обозначений вершин.

КРИТЕРИИ: (а) нет ответа; грамматика построена, но неэквивалентная или не соответствующая диаграмме : **–3**

(б) нет ответа; неверный ответ; ошибки в обосновании : **–3**

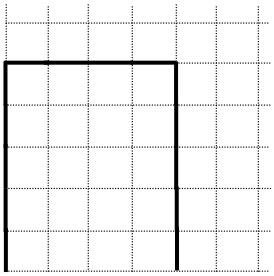
(в) нет ответа или программа на C++ приведена;
неэквивалентный или недетерм. автомат построен : **–4**

4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*a*, *c*, *e* }:

c — переместить перо на одну клетку вверх;

a — переместить перо на одну клетку вправо;

e — переместить перо на одну клетку вниз.



Например, по последовательности *ccccc a a a a eeeee* интерпретатор построит букву «п» размера 5×4, изображенную на рисунке.

Построить грамматику с терминальным алфавитом {*a*, *c*, *e*}, описывающую буквы «п» любого размера $n \times m$, $n,m \geq 1$. В грамматике должно быть **не более** 5 правил вывода (считая альтернативы).

Ответ: Описание буквы «п» размера $n \times m$ — это цепочка $c^n a^m e^n$. Следовательно, требуется построить грамматику, порождающую язык $\{ c^n a^m e^n \mid n,m \geq 1\}$: $S \to cSe \mid cAe$

$A \to aA \mid a$

КРИТЕРИИ: порождается пустая цепочка: **–5**

имеется лишняя (непустая) или недостающая цепочка: **–10**

за каждое лишнее правило: **–3** (количество правил в условии дано с запасом)

5. (а) По процедурам анализатора, действующего методом рекурсивного спуска, восстановить КС-грамматику.

```
char c;  void gc(){cin>>c;}; void S(); void A(); void B();
int main(){ try{ gc();S(); if(c!='\n') throw; cout<<"OK"<<endl; return 0;}
            catch(…) {  cout<<"unexpected: "<<c<<endl; return 1;}};
void S(){if (c=='b'){ gc(); S();A();} else if(c=='a'){ gc();} else throw;}
void B(){if (c=='e'){ gc();B();}
void A(){if (c=='a'){ gc();B();}
    else if(c=='c'){ gc();A(); if(c=='b') gc(); else throw;} else throw;}}
```

(б) Удовлетворяет ли эта грамматика достаточному условию применимости метода рекурсивного спуска с учетом возможной ε-альтернативы (т.н. «канонический вид» КС-грамматики)?
Ответ обосновать.

Ответ: (**а**) $S \to bSA \mid a$
$B \to eB \mid \varepsilon$
$A \to aB \mid cAb$

(**б**) **Да**, удовлетворяет: $first(bSA) = \{ b\}$, $first(a) = \{ a\}$, $first(bSA) \cap first(a) = \varnothing$;
$first(B) = \{e\}$, $follow(B) = \{a,b,c\}$, $first(B) \cap follow(B) = \varnothing$;
$first(aB) = \{a\}$, $first(cAb) = \{c\}$, $first(aB) \cap first(cAb) = \varnothing$.

КРИТЕРИИ: (а) нет ответа, неверно восстановленная грамматика: **–5**

(б) нет ответа; ответ не обоснован: **–5**

ответ верный, за ошибки в обосновании: до **–5**

6. Дана грамматика:

G_1 : $S \to cAb$
$A \to dS \mid cd$

Вставить в G_1 действия вида ⟨*cout* << 'символ' ;⟩ по переводу цепочек языка $L_1 = L(G_1)$ в цепочки языка $L_2 = \{b(a)^{k+1} \mid k \geq 1\}$, так, чтобы символов a в цепочке из L_2 было на один больше, чем символов d в соответствующей цепочке из L_1.

Ответ. Соглашение: запись ⟨a⟩ является сокращением ⟨*cout* << 'a';⟩
Грамматика с действиями: $S \to cAb$
$A \to dS\langle a \rangle \mid c\langle b \rangle d\langle a \rangle\langle a \rangle$
Другой ответ: $S \to cAb\langle a \rangle$
$A \to dS \mid c\langle b \rangle d\ \langle a \rangle$
Также возможна перестановка ⟨b⟩ непосредственно за символ d в обоих вариантах ответа.

Любая другая расстановка действий либо не дает подходящую цепочку из $L_2 = \{ b(a)^{k+1} \mid k \geq 1 \}$, либо не соответствует методу рекурсивного спуска.

КРИТЕРИИ:

За каждое неправильно вставленное (или пропущенное) действие вида ⟨*cout* << 'символ' ;⟩: **–4**

Если, кроме действий по печати символов ⟨*cout* << 'символ' ;⟩, есть другие вычисления: **0**

7. Приведите схему работы компилятора, указав входные и выходные данные для каждого этапа.

Ответ: исх.пр-ма ⇒ **Фаза анализа** [ЛА ⇒ посл.лексем + таблицы ⇒ СА + КУ ⇒ внутр.предст.] + **Фаза синтеза** [(*) ⇒ ГЕН] ⇒ объектная программа, **либо**
исх.пр-ма ⇒ **Фаза анализа** [ЛА + СА + КУ ⇒ внутр.предст. + таблицы] + **Фаза синтеза** [(*) ⇒ ГЕН] ⇒ объектная программа
(*) – подготовка к генерации объектной программы — распределение памяти, оптимизация.

КРИТЕРИИ: - за пропуск любого этапа: –3;
- за пропуск любых входных/выходных данных: –2;
- не указана фаза синтеза (остальное есть): 5 баллов.

8. Перечислите основные типы библиотек, что они из себя представляют? Укажите способы их подключения.

Ответ: 1. Библиотеки функций (или подпрограмм). 2. Библиотеки классов. 3. Библиотеки компонент.

Библиотеки функций представляют собой откомпилированные объектные модули. Подключаются статически во время работы редактора связей и динамически во время выполнения программы с помощью загрузчика.

Библиотеки классов - для СП, базирующихся на ООЯП. Все их классы должны быть написаны на том же ЯП, на котором пишется программа, куда интегрируются библиотечные классы.

В библиотеке классов различают *конкретные классы*, *абстрактные классы, шаблоны классов.*

Библиотеки классов включаются в программу на этапе компиляции и компилируются со всей программой вместе.

Библиотеки компонент - это библиотека готовых откомпилированных программных модулей, предназначенных для использования в качестве составной части программы, и которыми можно манипулировать во время разработки программы. Компоненты бывают локальные (находящиеся на той же ЭВМ, где создается ПП) и распределенные (расположенные на сервере и доступные по сети ЭВМ).
Примеры технологий, использующих библ. компонент: CORBA, COM, DCOM, ActiveX, Java Beans.

КРИТЕРИИ: - есть только перечисление типов библиотек: 1 балл;
- за отсутствие (неверное представление) какого-либо типа библиотек: –3;
- за каждый не указанный способ подключения: –2.

Замечание. За отсутствие упоминаний о распределенных объектах и компонентах не наказывать.

9. Записать результат перевода в ПОЛИЗ (постфиксную запись) фрагмента программы на языке Си:

```
if ( a < b ){ S = S - a - c; a++; b = -b; } else b = c = b - a;
```

Ответ:

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	a	b	<	21	!F	&S	S	a	-	c	-	=	&a	++	&b	b	@

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
=	28	!	&b	&c	b	a	-	=	=									

Соглашения: 1. Адреса ПОЛИЗа можно обозначать подчеркиванием (a = &a).
2. Префиксный ++ обозначен в ПОЛИЗе как ##, унарный минус — @ .
3. Различение префиксных и постфиксных ++ разрешается на уровне интерпретации.
4. "Мусор" в стеке после выполнения присваивания игнорируется.

КРИТЕРИИ: - пропуск адреса в первом операнде операции(ий) присваивания: –3;
- пропуск адреса в операнде операции(ий) ++ : –3;
- неверный ПОЛИЗ первого оператора: –6;
- неверный ПОЛИЗ цикла (любые, не указанные выше ошибки): 0 баллов.

Замечание. Допустимы следующие вариации ПОЛИЗа в ответах:
(1) разрешается разворачивать унарные операции ++ и -- :
префиксные — ++x ==> &x, x, 1, +, = ;
постфиксные — x++ ==> &y, x, =, &x, x, 1, +, = ,y , где y — новый операнд;
(2) может использоваться операция «;» для выбрасывания «мусора» из стека, при трактовке присваивания как операции, оставляющей результат присваивания на вершине стека.
Соответственно изменяются адреса переходов в связи с (1) и/или (2).

10. Перечислите основные методы выделения динамической памяти, дав их краткую характеристику и указав основные проблемы, возникающие при их использовании.

Ответ: Выделяемые области динамической памяти можно разделить на области памяти, **выделяемые пользователем (**явно) и **непосредственно компилятором** (неявно).

Явное выделение блоков фиксированного размера. При этом блоки памяти связываются в список, над которым достаточно легко выполнять операции их выделения и освобождения. Плюсы - если программа полностью использует блок памяти, то никаких дополнительных расходов не нужно. С каждым блоком связан указатель на его начало, и надо хранить только информацию о том, занят блок или нет.

Явное выделение блоков переменного размера. Здесь возникают проблемы. Например, фрагментация памяти, т.к. нельзя выделить блок, больший имеющихся свободных (Тогда необходимо приостанавливать выполнение программы и начинать реорганизацию кучи). Конечно, эта проблема существует и в предыдущем случае, но там она не представляет никакой сложности.

Один из методов выделения блоков переменного размера называется ***методом первого подходящего***.

Неявное выделение/освобождение памяти. Неявно выделяемые блоки памяти также могут быть фиксированного или переменного размера.

При неявном динамическом выделении и освобождении блоков памяти, выделяемые блоки обычно имеют следующую структуру:

- размер блока (если блок фиксированного размера, то соответственно эта информация в нём не хранится.);
- счётчик ссылок, пометка (обычно есть либо одно, либо другое):

Счетчик ссылок подсчитывает количество указателей в программе, которые ссылаются на этот блок (если происходит присваивание указателей, например, $p = q$, то соответственно, один счётчик (для блока, на который указывал указатель p) уменьшается на единицу, а другой увеличивается), если счётчик равен нулю, то блок не используется и его можно освободить. Существует проблема циклических ссылок, когда счётчики всегда >0.

Пометка фиксирует, задействован данный блок или не задействован. Т.е. у программы есть хотя бы один указатель, ссылающийся на этот блок. В некоторый момент начинает работать «сборщик мусора». Он помечает

все блоки как недостижимые, а потом начинает анализ текущих указателей программы, что является очень трудоемким процессом. Блоки, на которые ничего не указывает, считаются свободными и их можно перегруппировать и т.п. по желанию разработчика компилятора.

- указатели на блоки
- то, что досталось пользователю, заказавшему этот блок.

Допускается более краткий ответ с указанием среди проблем в случае фиксированного размера — неэффективное использование памяти и ее быстрое исчерпание, а случае переменного размера — поиск пригодного блока и реорганизация свободного пространства при отсутствии такого блока. Описывать подробно алгоритмы контроля свободного пространства не обязательно.

КРИТЕРИИ: - есть только перечисление основных методов выделения памяти: 1 балл;
- за каждую неверную (отсутствующую) характеристику: –3;
- за каждую не указанную проблему использования: –2.

2010 год
Коллоквиум

Вариант 1 _2010 Ф.И.О._____№ группы_____

1. ООП. Дать определение инкапсуляции. Что такое АТД? Каким образом АТД реализуется в C++?

2. Описать класс A таким образом, чтобы все конструкции функции main () были верными, а на экран выдалось **50 300** .

```
int main () {
        A a1 (5),  a2 = 3;
        a1  *=  10;
        a2  *=  a1  *= 2;
        cout << a1.get() << a2.get() << endl;
        return 0;
}
```

3. Если в реализации функций `B::g()` и main () есть ошибки, **объясните**, в чем они заключаются. К именам **каждой** правильной конструкции **этих функций** добавьте префикс, указывающий с помощью операции «**::**», из какой области видимости они выбираются.

```
      int x = 0;
      void f (int a, int b){ x = a+b;}
class A {
      int x;
public:
      void f (){x = 2;}
      void f (char a1, char b1){ x = a1 - b1;}
};
class B: public A {
public:
      void f ( int a ) { ::x = a; }
      void g () {
            f();
            f(0);
            f(5.3 , 1);
            x = 1;
      }
};
int main () {
      B b;
      f(2);
      f( 3 , 'a');
      return 0;
}
```

Ответы - вариант 1_2010

Максимальная оценка каждой задачи – 10 баллов

1. Инкапсуляция - механизм, связывающий вместе код и данные, которыми он манипулирует, и одновременно защищающий их от произвольного доступа со стороны другого кода, внешнего по отношению к рассматриваемому. Доступ к коду и данным жестко контролируется интерфейсом.

АТД - тип данных с полностью скрытой (инкапсулированной) структурой, а работа с переменными такого типа происходит только через специальные, предназначенные для этого функции. В C++ АТД реализуется с помощью классов (структур), в которых **все члены-данные** расположены в private области.

Критерии: За неверное понятие инкапсуляции – **-5,**
Неверное понятие АТД - **-3**
Неверный ответ о реализации АТД в C++ - **-4 .** В целом до 0.

- ```
  class A {
          int x;
  public:
          A ( int y ) { x = y; }
          int get () { return x; }
          int  operator *=  ( int y ) { return  x =  x*y; }
  };
  ```

Критерии: За каждый недостающий метод - **- 4**
За каждую ошибку реализации метода (неверный вывод…) - **- 2**

3.

```
void B::g() {
      f();          // ошибка, эта f не видна
      f(0);         // B::f(0);
      f(5.3 , 1);         // ошибка эта f не видна
      x = 1;        // ошибка, так как int x private в базовом классе
}
int main () {
      B b;
      f(2);         //ошибка, эта f не видна
      f( 3 , 'a');        //::f( 3 , 'a');
      return 0;
}
```

Критерии: За каждую не найденную ошибку **- 2**
За каждую лишнюю ошибку или неверно написанное с «::» имя
(это строго взаимоисключающие ошибки) **- 2**

Вариант 2 _2010 Ф.И.О._____№ группы_____

1. ООП. Что такое наследование? Какие способы наследования приняты в C++? В чем отличие статуса private и public членов базового класса в производном классе при public наследовании?

2. Описать класс B таким образом, чтобы все конструкции функции main () были верными, а на экран выдалось **10 20 30** .

```
int main () {
      B b1,  b2 = b1, b3 (b2);
      cout << b1.get() << b2.get() << cout b3.get () << endl;
      return 0;
}
```
```

3. Если в реализации функций `D::h()` и main () есть ошибки, **объясните**, в чем они заключаются.
К именам **каждой** правильной конструкции **этих функций** добавьте префикс, указывающий с помощью операции «**::**», из какой области видимости имена выбираются.

```
        double a = 0;
        void f (double x = 2){a = x;}
        void f () { a = 1; }
struct B {
        double a;
        void f (){a = 2;}
};
class D: B {
public:
        void f (int a){::a = a;}
        void h () {
                f( 'r' );
                f();
                a = 2;
        }
};
int main () {
        D d;
        f();
        f(6);
        return 0;
}
```

Ответы - вариант 2_2010

Максимальная оценка каждой задачи – 10 баллов

1. Наследование - механизм, с помощью которого один объект (производного класса) приобретает свойства другого объекта (родительского, базового класса). Наследование позволяет какому-либо объекту наследовать от своего родителя общие атрибуты, а для себя определять только те характеристики, которые делают его уникальным внутри класса.
В C++ наследование бывает public, private и protected. Private-члены базового класса **видны**, но **недоступны** в производном классе, public-члены – и **видны**, и **доступны**.

Критерии: За неверное понятие наследования – **-5,**
За каждый недостающий способ наследования - **-2**
Неверный ответ о статусе private и public членов - **-4**
(**-2** за видимость и **-2** за доступность**).** В целом до 0.

- ```
  class B {
          int x;
  public:
          B () { x = 10; }
          B (const B&  y) { x = y.x + 10 }
          int get () { return x; }
  };
  ```

Критерии: За каждый недостающий метод - **- 4**
За каждую ошибку реализации метода (неверный вывод…) - **-2**

3.
```
void D::h() {
        f( 'r');  // D::f
        f();      // ошибка эта f не видна
        a = 2;    // B::a = 2
}
int main () {
        D d;
        f();    //ошибка, неоднозначный выбор f
```
```

```
        f(6);    //::f
        return 0;
    }
```

Критерии: За каждую не найденную ошибку **- 2**
За каждую лишнюю ошибку или неверно написанное с «::» имя
(это строго взаимоисключающие ошибки) **- 2**

Вариант 3_2010 Ф.И.О._____№ группы_____

1. ООП. Что такое полиморфизм? Какие виды полиморфизма реализованы в C++, с помощью каких механизмов? Когда происходит выбор конкретной языковой конструкции для каждого вида полиморфизма?

2. Описать класс C таким образом, чтобы все конструкции функции main () были верными, а на экран выдалось **14 10 12** .

```
int main () {
        C c1 (7),  c2 = 5, c3 (c1 + c2);
        cout << c1.get() << c2.get() << c3.get () << endl;
        return 0;
}
```

3. Если в реализации функций `S::g()` и main () есть ошибки, **объясните**, в чем они заключаются.
К именам **каждой** правильной конструкции **этих функций** добавьте префикс, указывающий с помощью операции «**::**», из какой области видимости имена выбираются.

```
      float y = 0;
      void f (float a ){y = a;}
class T {
      int y;
public:
      void f (){y = 2;}
};
class S: public T {
public:
      void f (float n, float m){::y = n * m;}
      void f (char c1, char c2) {::y = c1 + c2;}
      void g () {
            f();
            f(1);
            f(-1 , 1);
            x = 2;
      }
};
int main () {
      S b;
      f(5);
      f('+', 6);
      return 0;
}
```

Ответы - вариант 3_2010

Максимальная оценка каждой задачи – 10 баллов

1. Полиморфизм - механизм, позволяющий использовать один и тот же интерфейс для общего класса действий.

Различаются статический (на этапе компиляции, реализуется с помощью перегрузки функций/операций), динамический (во время выполнения программы, реализуется с помощью виртуальных функций) и параметрический (на этапе компиляции, с использованием механизма шаблонов) полиморфизм.

Критерии: За неверное понятие полиморфизма – **-5,**

За каждый недостающий вид полиморфизма - **-2**
За каждый неверный ответ о этапе разрешения полиморфизма - **-2**
В целом до 0.

2.
```
class C {
        int x;
public:
        C (int y) { x = 2 * y; }
        int get () { return x; }
        C& operator + ( C c) {
                C t ( x + c.x );
                return  t;
        }
};
```

Критерии: За каждый недостающий метод - **- 4**
За каждую ошибку реализации метода (неверный вывод…) - **- 2**

3.
```
void S::g() {
        f();          // ошибка, эта f не видна
        f(1);         // ошибка, эта f не видна
        f(-1 , 1);    // ошибка, неоднозначный выбор f
        x = 2;        // ошибка, так как int x private в базовом классе
}

int main () {
        S b;
        f(5);         // ::f(5);
        f('+', 6);    // ошибка, эта f не видна
        return 0;
}
```

Критерии: За каждую не найденную ошибку **- 2**
За каждую лишнюю ошибку или неверно написанное с «::» имя
(это строго взаимоисключающие ошибки) **- 2**

Вариант 4 _2010 Ф.И.О._____№ группы_____

1. ООП. Перечислите основные постулаты (механизмы) ООП. Каким образом реализация каждого постулата упрощает процесс создания сложного ПП?

2. Описать класс D таким образом, чтобы все конструкции функции main () были верными, а на экран выдалось **30 10 20** .

```
int main () {
        D d1,  d2 (1), d3 (2);
        d1 = d2 + d3;
        cout << d1.get () << d2.get () << d3.get () << endl;
        return 0;
}
```

3. Если в реализации функций `X::g()` и main () есть ошибки, **объясните**, в чем они заключаются.

К именам **каждой** правильной конструкции **этих функций** добавьте префикс, указывающий с помощью операции «**::**», из какой области видимости имена выбираются.

```
void f(){ putchar ('f');}
class Y {
public:
        void f () {}
}
class X : public Y{
        double a;
        X ( int  k = 0) { a = k; }
public:
        X (double r ) { a = r; }
        void f (int x) { a = x;}
        void f (int  x,  int  y = 5) { a =  x – y;}
        void g ()  {
                f ();
                f (3);
                f (1 ,  2 );
        }
};
int main () {
        Y y;
        X a ;
        X b (2.5);
        b.f ();
        return 0;
}
```

Ответы - вариант 4_2010

Максимальная оценка каждой задачи – 10 баллов

1. Постулаты ООП – инкапсуляция, наследования и полиморфизм.

Механизма инкапсуляции позволяет некоторые детали реализации класса оставлять скрытыми для пользователя (т.е. инкапсулировать их в классе), что упрощает работу с объектами класса.

Механизм наследования позволяет описывать новый класс не обязательно, начиная с нуля, а использовать в создаваемом (производном) классе готовые методы и данные базового класса, что существенно упрощает работу программиста.

Полиморфизм позволяет программисту помнить и применять один интерфейс для однотипных действий, вместо нескольких, что также упрощает работу.

Критерии: За каждый верный ответ - **+3**.
Все верные ответы - **+10**.

2.

```
class D {
        int x;
public:
        D () {x = 0;}
        D (int y) { x =  y; }
        int get () { return x * 10; }
        D operator + ( D d) {
                D  t ( x + d.x );
                return  t;
        }
};
```

Критерии: За каждый недостающий метод - **- 4**
За каждую ошибку реализации метода (неверный вывод…) - **- 2**

3.
```
void X::g ()  {
        f ();      // ошибка,  эта f  не видна
        f (3);     // ошибка, неоднозначный выбор f
        f (1 ,  2 );          // X::f (1, 2);
}
int main () {
        X a ;     // ошибка, обращение к закрытому конструктору класса
        f ();      // ::f ();
        f ('a', 'b');         // ошибка, нет такой внешней f
        return 0;
}
```

Критерии: За каждую не найденную ошибку **- 2**
За каждую лишнюю ошибку или неверно написанное с «::» имя (это строго взаимоисключающие ошибки) **- 2**

2010 год
Экзамен

Вариант 1_2010 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to AB$
$A \to a \mid \varepsilon$
$B \to CbD \mid C$
$C \to E \mid EE$
$D \to C \mid \varepsilon$
$EE \to EEE$
$E \to c$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные языки	
контекстно-зависимые языки	
языки типа 0	
регулярные языки	

(в) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.

2. Дать определение эквивалентных грамматик. Приведите примеры эквивалентных и почти эквивалентных грамматик. Могут ли неэквивалентные грамматики порождать один и тот же язык?

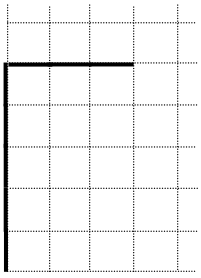
3. Построить по заданной грамматике G конечный автомат (в виде ДС). Если автомат оказался детерминированным, построить анализатор на Си, иначе – преобразовать автомат в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА, и по полученному ДКА построить леволинейную грамматику.

G: $H \to bA \mid bB \mid aC$
$A \to bC$
$B \to aC$
$C \to aA \mid bB \mid \perp$

4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита $\{c, a, e\}$: c — переместить перо на одну клетку вверх;

a — переместить перо на одну клетку вправо;

e — переместить перо на одну клетку вниз.



Например, по последовательности $ccccaaa$ интерпретатор построит букву «Г» размера 5×3, изображенную на рисунке.

Построить грамматику с терминальным алфавитом $\{c, a\}$, описывающую буквы «Г» любого размера $n \times m$, $n,m \geq 1$. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

В построенную грамматику добавить действия вида ⟨***cout*** `<< 'символ';`⟩ так, чтобы описание буквы «Г» размера $n \times m$ переводилось в описание буквы «П» того же размера.

5. Восстановить грамматику по заданному анализатору на C++. Сформулировать достаточное условие применимости метода рекурсивного спуска для грамматик без эпсилон-правил. Соответствует ли полученная грамматика

данному условию? Обосновать свой ответ. Корректен ли данный анализатор? Обосновать свой ответ (доказать корректность или найти примеры некорректного поведения анализатора).

```
void S () { if (c == 'b') { c = GetS (); A ();
                            if (c == 'a') { c = GetS (); C (); D (); }
                            else Error ();
                            if (c == '⊥') return;
                          }
            Error ();
          }
void A () { if (c == 'a') { c = GetS (); A (); }
            else B ();
}
void B () { if (c == 'b') { c = GetS (); B (); }
            else C ();
}
void C () { if (c == 'a') { c = GetS (); C (); }
            else D ();
}
void D () { if (c == 'b') { c = GetS (); D (); }
            else A ();
}
```

6. Дать определение системы программирования. Перечислить основные компоненты систем программирования и их основное назначение.

7. Перечислить не менее 5 различных видов внутреннего представления программ, используемых при компиляции.

8. С какими компонентами интегрированных систем программирования взаимодействуют входящие в них символьные отладчики? Какую помощь и для чего оказывают отладчикам эти компоненты?

9. Дать определение приведённых грамматик. Преобразовать контекстно-свободную грамматику G к приведённому виду, выписывая результаты применения каждого шага используемого алгоритма.

G: $S \to ACb \mid Aa$
$A \to Aa \mid Ca \mid a$
$B \to Aa \mid CDb \mid Bb$
$C \to Ab \mid b$
$D \to Aa \mid Cb \mid a$

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ПОЛИЗ	&x	#+	4	+	y	5	—	20	*	<=	30	!F	&a	x	8	—	7

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
*	y	x	@	2	+	/	—	=	;	!	1	&b	&x	&y	+#	=	=	;

Вариант 1_2010 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

За каждую задачу — максимум 10 баллов

1. Дана грамматика G:

$S \to AB$
$A \to a \mid \varepsilon$
$B \to CbD \mid C$
$C \to E \mid EE$
$D \to C \mid \varepsilon$
$EE \to EEE$
$E \to c$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы). **Ответ**: $L(G) = \{ a^i c^n b^j c^m \mid i, j \in \{0,1\}, n \geq 1, m \geq 0\}$

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные языки	**да (-1)**
контекстно-зависимые языки	**да (-1)**
языки типа 0	**да (-1)**
регулярные языки	**да (-3)**

(в) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа $k+1$.
Ответ: $k = 0$ (есть укорачивающее правило $A \to \varepsilon$ и не КС-правило $EE \to EEE$)
КРИТЕРИИ: (а) описание отсутствует или с ошибками: **–4**
(б) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице (обоснование не обязательно)
(в) нет ответа; неверный ответ: **–4** (обоснование не обязательно)

2. Дать определение эквивалентных грамматик. Приведите примеры эквивалентных и почти эквивалентных грамматик. Могут ли неэквивалентные грамматики порождать один и тот же язык?

Ответ:
(**а**) Грамматики G_1 и G_2 называются эквивалентными, если их языки совпадают, $L(G_1) = L(G_2)$. (**б**) Например, из трёх разных грамматик:
$G_1 = (\{0,1\}, \{A,S\}, P_1, S)$ и $G_2 = (\{0,1\}, \{S\}, \{S \to 0S1 \mid 01\}, S)$
P_1: $S \to 0A1$ $G_3 = (\{0,1\}, \{S\}, \{S \to 0S1 \mid \varepsilon\}, S)$
$0A \to 00A1$
$A \to \varepsilon$
G_1 и G_2 эквивалентны, $L(G_1) = L(G_2) = \{0^n 1^n \mid n > 0\}$.
G_1 (G_2) и G_3 почти эквивалентны, $L(G_3) = \{0^n 1^n \mid n \geq 0\}$
(**в**) **Нет**, это невозможно по приведённому определению.
КРИТЕРИИ: (а) нет ответа; определение с ошибками: **–5**
(б) за каждый из двух отсутствующих примеров: **–3**
(в) нет ответа; неверный ответ: **–5**

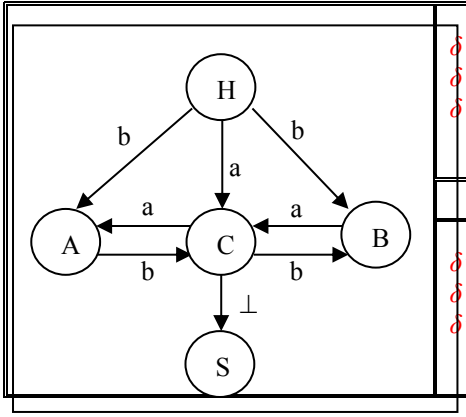
3. Построить по заданной грамматике G конечный автомат (в виде ДС). Если автомат оказался детерминированным, построить анализатор на Си, иначе – преобразовать автомат в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА, и по полученному ДКА построить леволинейную грамматику.

G: $H \to bA \mid bB \mid aC$
$A \to bC$
$B \to aC$
$C \to aA \mid bB \mid \perp$

Ответ: Автомат недетерминирован

G': $S \to C\perp$
$A \to Ca$
$B \to Cb$
$C \to Ab \mid Ba \mid Da \mid Db \mid a$
$D \to b$

Функция переходов исходного автомата



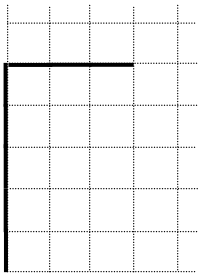
$\delta\,(H, a) = C$ $\delta\,(H, b) = A$ $\delta\,(H, b) = B$	$\delta\,(A, b) = C$ $\delta\,(B, a) = C$	$\delta\,(C, a) = A$ $\delta\,(C, b) = B$ $\delta\,(C, \perp) = S$
Функция переходов детерминированного автомата (D ≡ AB)		
$\delta'\,(H, a) = C$ $\delta'\,(H, b) = D$ $\delta'\,(A, b) = C$	$\delta'\,(C, a) = A$ $\delta'\,(C, b) = B$ $\delta'\,(C, \perp) = S$	$\delta'\,(B, a) = C$ $\delta'\,(D, a) = C$ $\delta'\,(D, b) = C$

КРИТЕРИИ: (**а**) построен неэквивалентный или недетерминированный автомат: **–5**
(**б**) леволинейная грамматика построена, но неэквивалентная или не соответствующая новой диаграмме: **–5**
(**в**) нет ответа или приведена программа анализатора: **0**

4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*c*, *a*, *e* }: *c* — переместить перо на одну клетку вверх;

a — переместить перо на одну клетку вправо;

e — переместить перо на одну клетку вниз.



Например, по последовательности *ccccaaa* интерпретатор построит букву «Г» размера 5×3, изображенную на рисунке.

Построить грамматику с терминальным алфавитом {*c*, *a*}, описывающую буквы «Г» любого размера *n*×*m*, *n,m* ≥1. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

В построенную грамматику добавить действия вида ⟨***cout*** **<< 'символ';**⟩ так, чтобы описание буквы «Г» размера *n*×*m* переводилось в описание буквы «П» того же размера.

Ответ: Описание буквы «Г» размера *n*×*m* — это цепочка $c^n a^m$. Следовательно, требуется построить грамматику, порождающую язык $\{c^n a^m \mid n,m \geq 1\}$: $S \to cS \mid cA$

$A \to aA \mid a$

Соглашение запись ⟨*a*⟩ является сокращением ⟨*cout* << "*a*";⟩
Грамматика с действиями: $S \to c\langle c\rangle S\langle e\rangle \mid c\langle c\rangle A\langle e\rangle$

$A \to a\langle a\rangle A \mid a\langle a\rangle$

КРИТЕРИИ: порождается пустая цепочка: **–5**
имеется лишняя (непустая) или недостающая цепочка: **–10**
за каждое лишнее правило: **–3**

5. Восстановить грамматику по заданному анализатору на C++. Сформулировать достаточное условие применимости метода рекурсивного спуска для грамматик без эпсилон-правил. Соответствует ли полученная грамматика данному условию? Обосновать свой ответ. Корректен ли данный анализатор? Обосновать свой ответ (доказать корректность или найти примеры некорректного поведения анализатора).

```
void S () { if (c == 'b') { c = GetS (); A ();
                            if (c == 'a') { c = GetS (); C (); D (); } else Error ();
                            if (c == '⊥') return;
                          }
            Error ();
          }
void A () { if (c == 'a') { c = GetS (); A (); } else B (); }
void B () { if (c == 'b') { c = GetS (); B (); } else C (); }
void C () { if (c == 'a') { c = GetS (); C (); } else D (); }
```

```
void D () { if (c == 'b') { c = GetS (); D (); } else A (); }
```

Ответ: Восстановленная грамматика имеет вид:

$G = (\{a, b, \perp\}, \{A, B, C, D, S\}, P, S)$ P: $S \to bAaCD\perp$

$A \to aA \mid B$ $B \to bB \mid C$

$C \to aC \mid D$ $D \to bD \mid A$

Анализатор некорректен: он, например, зацикливается на цепочке "b⊥".

Достаточное условие применимости метода рекурсивного спуска для грамматик без эпсилон-правил:

(1) либо $A \to \alpha$, где $\alpha \in (T \cup N)^*$ и это единственное правило вывода для A;

(2) либо $A \to a_1\alpha_1 \mid a_2\alpha_2 \mid \ldots \mid a_n\alpha_n$, где $a_i \in T$ для $i = 1,2,\ldots,n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$.

Грамматика G достаточным условиям не соответствует: не все правила для нетерминальных символов A, B, C и D начинаются с терминальных символов.

КРИТЕРИИ: Неправильно восстановлена грамматика: **-5**.
Неверная формулировка достаточного условия применимости: **-4**.
Неверный ответ на вопрос о соответствии грамматики сформулированному условию: **-2**.
Нет обоснования соответствия/несоответствия грамматики: **-1**.
Неверный ответ на вопрос о корректности анализатора: **-2**.
Нет обоснования корректности (не найден пример "плохой" цепочки): **-1**.

6. Дать определение системы программирования. Перечислить основные компоненты систем программирования и их основное назначение.

Ответ: системой программирования называется комплекс программных средств, предназначенных для поддержки программного продукта на протяжении всего жизненного цикла этого продукта. Основные компоненты современных развитых систем программирования:

- средства интеграции компонентов системы программирования (**-1**)
- редакторы текстов (**-2**)
- трансляторы: интерпретаторы, компиляторы и ассемблеры, вместе с макрогенераторами (**-2**)
- библиотеки (**-2**)
- редакторы связей (**-2**)
- средства конфигурирования (**-1**)
- отладчики (**-2**)
- средства тестирования (**-1**)
- профилировщики (**-1**)
- справочные системы (**-1**)

КРИТЕРИИ: За пропуск компонентов снижать на **1-2** балла.

7. Перечислить не менее 5 различных видов внутреннего представления программ, используемых при компиляции.

Ответ:

- Связные списочные структуры, представляющие синтаксическое дерево
- Многоадресный код с явно именуемыми результатами (тетрады)
- Многоадресный код с неявно именуемыми результатами (триады)
- Инфиксная запись
- Префиксная запись
- Постфиксная запись
- Язык ассемблера

КРИТЕРИИ: За каждый пропущенный из необходимых 5 видов: **-2**.

8. С какими компонентами интегрированных систем программирования взаимодействуют входящие в них символьные отладчики? Какую помощь и для чего оказывают отладчикам эти компоненты?

Ответ: С текстовыми редакторами, компиляторами и редакторами связей. Текстовые редакторы обеспечивают доступ к текстовой структуре программе (для расстановки точек останова и пошагового исполнения программы). Компиляторы и редакторы связей обеспечивают доступ к информационным таблицам имен, адресов, к описаниям областей памяти (для выдачи информации в терминах входной программы).

КРИТЕРИИ: За пропуск какого-либо компонента: **-3** (за каждый пропуск).
За пропуск объяснения какого-либо взаимодействия компонентов: **-2** (за каждый).

9. Дать определение приведённых грамматик. Преобразовать контекстно-свободную грамматику G к приведённому виду, выписывая результаты применения каждого шага используемого алгоритма.

G: $S \to ACb \mid Aa$
$A \to Aa \mid Ca \mid a$
$B \to Aa \mid CDb \mid Bb$
$C \to Ab \mid b$
$D \to Aa \mid Cb \mid a$

Ответ: Контекстно-свободная грамматика называется приведённой, если в ней нет бесполезных (бесплодных и недостижимых) символов.
1. Удаление бесплодных символов. Строится множество N, из элементов которого за некоторое число шагов выводятся терминальные символы:
   1. $N_0 = \varnothing$
   2. $N_1 = \{A, C, D\}$, $N_1 \neq N_0$
   3. $N_2 = \{A, B, C, D, S\}$, $N_2 \neq N_1$
   4. $N_3 = \{A, B, C, D, S\}$, $N_3 = N_2$

Бесплодных символов в грамматике G не обнаружено.
2. Удаление недостижимых символов. Строится множество V, в которое помещаются символы, за некоторое число шагов выводимые из символа S:
   1. $V_0 = \{S\}$
   2. $V_1 = \{S, A, C\}$, $V_1 \neq V_0$
   3. $V_2 = \{S, A, C\}$, $V_2 = V_1$

Символы B и D из алфавита грамматики G являются недостижимыми, их и правила их содержащие можно удалить из грамматики:
$G_{прив}$: $S \to ACb \mid Aa$ $A \to Aa \mid Ca \mid a$ $C \to Ab \mid b$

КРИТЕРИИ: Нет определения или оно неверно: **-5**.
За ошибку при преобразовании: **-7**.

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&x	#+	4	+	y	5	—	20	*	<=	30	!F	&a	x	8	—	7

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
*	y	x	@	2	+	/	—	=	;	!	1	&b	&x	&y	+#	=	=	;

Ответ:
M: *if (! (x ++ + 4 <= (y – 5) * 20)) goto L;*
*a = (x – 8) * 7 – y / (– x + 2);*
goto M;
L: *b = (x = ++ y);*

*while (x ++ + 4 <= (y – 5) * 20) a = (x – 8) * 7 – y / (– x + 2); b = (x = ++ y);*

КРИТЕРИИ: за неверный ответ на первый вопрос: **-7**.
за неверный ответ на второй вопрос, если первый вариант верен: **-3**, иначе: **-10**

Вариант 2_2010 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to aA \mid A$
$A \to BbB \mid Bb \mid B$
$B \to C \mid CC$
$CC \to CCC$
$C \to c$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

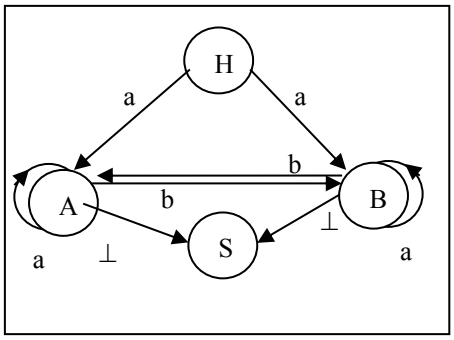
(б) Каким из перечисленных классов грамматик принадлежит G?

Класс П	$G \in$ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

(в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком типа k+1.

2. Дать определение порождающей грамматики. Какие ограничения на использование терминальных и нетерминальных символов в правилах грамматики наложены этим определением? Предложите какую-либо грамматику для языка `L = {α∈(a,b)⁺}` и укажите на этом примере все компоненты, упоминаемые в определении грамматики.

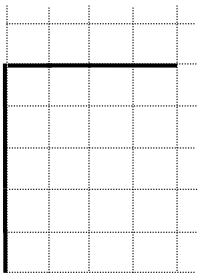
3. Дана диаграмма состояний конечного автомата. Построить по этой диаграмме соответствующую леволинейную грамматику. Если заданный автомат детерминирован, построить анализатор на Си, иначе – преобразовать автомат в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА, и по полученному ДКА построить праволинейную грамматику.



4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита $\{c, a, e\}$: c — переместить перо на одну клетку вверх;

a — переместить перо на одну клетку вправо;

e — переместить перо на одну клетку вниз.



Например, по последовательности $ccccaaaa$ интерпретатор построит букву «Г» размера 5×4, изображенную на рисунке.

Построить грамматику с терминальным алфавитом $\{c, a\}$, описывающую буквы «Г» любого размера $n \times m$, $n,m \geq 1$. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

В построенную грамматику добавить действия вида ⟨***cout*** `<< 'символ';`⟩ так, чтобы описание буквы «Г» размера $n \times m$ переводилось в описание буквы «П» размера $n \times (m+1)$.

5. Восстановить грамматику по заданному анализатору на С++. Сформулировать достаточное условие применимости метода рекурсивного спуска для грамматик с эпсилон-правилами. Соответствует ли полученная грамматика данному

условию? Обосновать свой ответ. Корректен ли данный анализатор? Обосновать свой ответ (доказать корректность или найти примеры некорректного поведения анализатора).

```
void S () { if (c == '1') { c = GetL ();  A();
                            if (c != '0') Error ();
                            c = GetL ();  C(); D();
                            if (c != '1') Error ();
                            c = GetL ();
                          }
            else Error ();
          }
void A () {      if (c == '0') { c = GetL ();  A(); }
            else if (c == '1') { c = GetL ();  B(); }
            else Error ();
          }
void B () {      if (c == '0') { c = GetL ();  C(); }
            else if (c == '1') { c = GetL ();  B(); }
            else Error ();
          }
void C () {      if (c == '0') { c = GetL ();  C(); }
            else if (c == '1') { c = GetL ();  D(); }
            else Error ();
          }
void D () {      if (c == '1') { c = GetL ();  D(); } }
```

6. Описать основные задачи ассемблеров, компиляторов и интерпретаторов. В чём сходство и различие этих компонентов систем программирования?

7. Нарисовать схему однопроходного компилятора. Каким образом в таких компиляторах взаимодействуют лексический и синтаксический анализаторы? Какой из анализаторов является ведущим в таком взаимодействии?

8. Описать основные подходы, использующие поведенческие и структурные тесты программ. В чем их основные различия?

9. Дать определения бесплодных и недостижимых символов алфавита контекстно-свободной грамматики. Преобразовать контекстно-свободную грамматику G к приведённому виду, выписывая результаты применения каждого шага используемого алгоритма.

G: $S \to Ab \mid Ba$
$A \to Ca$
$B \to Bb \mid Da$
$C \to Aa \mid Bb \mid a \mid b$
$D \to Db \mid Ba$

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&a	x	@	11	*	y	x	@	/	—	+=	;	&x	#+	5	&y	+#

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	
—	x	+	1	—	<=	28	!F	1	!	&y	&x	y	8	+	=	7	+	=	;	}

Вариант 2_2010 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

За каждую задачу — максимум 10 баллов

1. Дана грамматика G:

$S \to aA \mid A$
$A \to BbB \mid Bb \mid B$
$B \to C \mid CC$
$CC \to CCC$
$C \to c$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).
Ответ: $L(G) = \{ a^i c^n b^j c^m \mid i, j \in \{0,1\}, n \geq 1, m \geq 0\}$

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

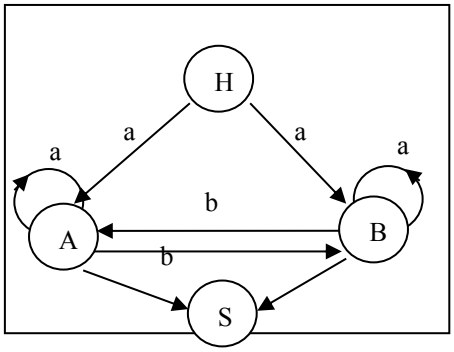
Класс П	$G \in$ П? (да/нет)
регулярные	**нет (-3)**
контекстно-зависимые	**да (-1)**
контекстно-свободные	**нет (-3)**
грамматики типа 0	**да (-1)**
неукорачивающие	**да (-1)**

(в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком k+1.
Ответ: k = **3** (для заданного языка существует порождающая его регулярная грамматика)
КРИТЕРИИ: (а) описание отсутствует или с ошибками: **–4**
(б) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице (обоснование не обязательно)
(в) нет ответа; неверный ответ: **–4** (обоснование не обязательно)

2. Дать определение порождающей грамматики. Какие ограничения на использование терминальных и нетерминальных символов в правилах грамматики наложены этим определением? Предложите какую-либо грамматику для языка `L = {α∈(a,b)⁺}` и укажите на этом примере все компоненты, упоминаемые в определении грамматики.

Ответ: (**а**) Порождающая грамматика G – это четверка (T, N, P, S), где
T – алфавит терминальных символов (терминалов),
N – алфавит нетерминальных символов (нетерминалов), множества N и T не пересекаются друг с другом, то есть $N \cap T = \varnothing$,
P – конечное множество правил – подмножество множества $(T \cup N)^+ \times (T \cup N)^*$;
S – начальный символ (цель) грамматики, $S \in N$.
(**б**) Этим определением требуется, чтобы в левой части любого правила грамматики присутствовал хотя бы один нетерминальный символ. Других формальных ограничений нет.
(**в**) Например, в грамматике G = ({a,b}, {A,S}, {S → AS | A; A → a | b}, S):
Множество терминальных символов T – {a,b}
Множество нетерминальных символов N – {A,S}
Множество правил P – {S → AS | A; A → a | b}
Начальный символ (цель) грамматики S
КРИТЕРИИ: (**а**) нет ответа; определение с ошибками: **–5**
(**б**) нет ответа; неверный ответ: **–3**
(**в**) нет примера грамматики или неправильный пример: **–4**
за каждый неверный или отсутствующий компонент определения: **–1** (до **–4**)

3. Дана диаграмма состояний конечного автомата. Построить по этой диаграмме соответствующую леволинейную грамматику. Если заданный автомат детерминирован, построить анализатор на Си, иначе – преобразовать автомат в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА, и по полученному ДКА построить праволинейную грамматику.



Ответ: *Гл:* $S \to A\perp \mid B\perp$
$A \to Aa \mid Bb \mid a$
$B \to Ab \mid Ba \mid a$
Автомат недетерминирован

⊥ ⊥

Функция переходов исходного автомата

$\delta(H, a) = A$ $\delta(H, a) = B$	$\delta(A, a) = A$ $\delta(A, b) = B$ $\delta(A, \perp) = S$	$\delta(B, a) = B$ $\delta(B, b) = A$ $\delta(B, \perp) = S$

Функция переходов детерминированного автомата (C ≡ AB)

$\delta'(H, a) = C$	$\delta'(C, a) = C$	$\delta'(C, b) = C$	$\delta'(C, \perp) = S$

Glев: $S \to C\perp$
$C \to Ca \mid Cb \mid a$

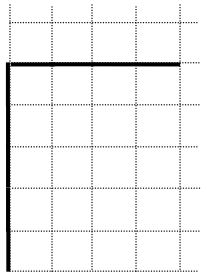
Gправ: $H \to aC$
$C \to aC \mid bC \mid \perp$

КРИТЕРИИ: **(а)** нет или неправильная леволинейная грамматика исходного автомата: **–5**
(б) построен неэквивалентный или недетерминированный автомат: **–5**
(в) праволинейная грамматика построена, но неэквивалентная или не соответствующая новой диаграмме: **–5**
(г) нет ответа или приведена программа анализатора: **0**

4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*c*, *a*, *e*}: *c* — переместить перо на одну клетку вверх;

a — переместить перо на одну клетку вправо;

e — переместить перо на одну клетку вниз.



Например, по последовательности *ccccaaaa* интерпретатор построит букву «Г» размера 5×4, изображенную на рисунке.

Построить грамматику с терминальным алфавитом {*c*, *a*}, описывающую буквы «Г» любого размера *n*×*m*, *n*,*m* ≥1. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

В построенную грамматику добавить действия вида ⟨***cout*** **<< 'символ';**⟩ так, чтобы описание буквы «Г» размера *n*×*m* переводилось в описание буквы «П» размера *n* × (*m*+1).

Ответ: Описание буквы «Г» размера *n*×*m* — это цепочка $c^n a^m$. Следовательно, требуется построить грамматику, порождающую язык $\{c^n a^m \mid n,m \ge 1\}$: $S \to cS \mid cA$ $A \to aA \mid a$

Соглашение: запись ⟨*a*⟩ является сокращением ⟨*cout* << "*a*";⟩

Грамматика с действиями: $S \to c\langle c\rangle S\langle e\rangle \mid c\langle c\rangle A\langle e\rangle$ $A \to a\langle a\rangle A \mid a\langle a\rangle\langle a\rangle$

КРИТЕРИИ: порождается пустая цепочка: **–5**
имеется лишняя (непустая) или недостающая цепочка: **–10**
за каждое лишнее правило: **–3**

5. Восстановить грамматику по заданному анализатору на C++. Сформулировать достаточное условие применимости метода рекурсивного спуска для грамматик с эпсилон-правилами. Соответствует ли полученная грамматика данному условию? Обосновать свой ответ. Корректен ли данный анализатор? Обосновать свой ответ (доказать корректность или найти примеры некорректного поведения анализатора).

```
void S () { if (c == '1') { c = GetL ();  A();
                            if (c != '0') Error ();
                            c = GetL ();  C(); D();
                            if (c != '1') Error ();
                            c = GetL ();
                          }
            else Error ();
          }
void A () {      if (c == '0') { c = GetL ();  A(); }
            else if (c == '1') { c = GetL ();  B(); }  else Error ();    }
void B () {      if (c == '0') { c = GetL ();  C(); }
            else if (c == '1') { c = GetL ();  B(); }  else Error ();    }
void C () {      if (c == '0') { c = GetL ();  C(); }
            else if (c == '1') { c = GetL ();  D(); }  else Error ();    }
```

```
void D () {      if (c == '1') { c = GetL ();  D(); } }
```

Ответ: Восстановленная грамматика имеет вид:

$G = (\{0, 1\}, \{A, B, C, D, S\}, P, S)$ P: $S \to 1A0CD1$

$A \to 0A \mid 1B$

$B \to 0C \mid 1B$

$C \to 0C \mid 1D$

$D \to 1D \mid \varepsilon$

Анализатор некорректен: любая правильная цепочка должна заканчиваться символом `'1'`, а этот символ может быть зря прочитан процедурой `"D()"`, что приведёт к неправильной трактовке всех правильных цепочек, например, анализатор выдаёт ошибку на правильной цепочке `"10101011"`.

Достаточное условие применимости метода рекурсивного спуска для грамматик с эпсилон-правилами:

a) либо $X \to \alpha$, где $\alpha \in (T \cup N)^*$ и это единственное правило вывода для X;

b) либо $X \to a_1\alpha_1 \mid \ldots \mid a_n\alpha_n$ где $a_i \in T$ для $i = 1, \ldots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$;

c) либо $X \to a_1\alpha_1 \mid \ldots \mid a_n\alpha_n \mid \varepsilon$ где $a_i \in T$ для $i = 1, \ldots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$,
и $first(X) \cap follow(X) = \varnothing$.

Грамматика G показанным условиям не соответствует: $first(D) \cap follow(D) = \{1\} \neq \varnothing$.

КРИТЕРИИ: Неправильно восстановлена грамматика: **-5**.
Неверная формулировка достаточного условия применимости: **-5**.
Неверный ответ на вопрос о соответствии грамматики сформулированному условию: **-3**.
Нет обоснования соответствия/несоответствия грамматики: **-2**.
Неверный ответ на вопрос о корректности анализатора: **-3**.
Нет обоснования корректности (нет рассуждений о конечных символах цепочек, не найден пример неправильно трактуемой цепочки и т. д.): **-2**.

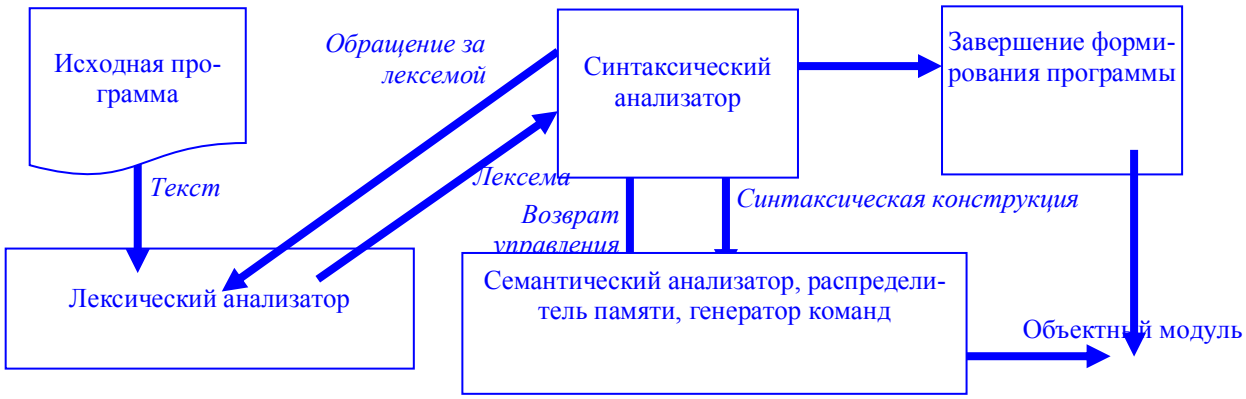
6. Описать основные задачи ассемблеров, компиляторов и интерпретаторов. В чём сходство и различие этих компонентов систем программирования?

Ответ: Ассемблер – для трансляции программ, написанных на языках типа 1:1, компилятор – для трансляции программ, написанных на языках высокого уровня, интерпретатор – не для трансляции программ, а для получения результата с помощью алгоритма, закодированного в программе.

КРИТЕРИИ: За отсутствие комментариев к какому-либо виду трансляторов: **-4**.

7. Нарисовать схему однопроходного компилятора. Каким образом в таких компиляторах взаимодействуют лексический и синтаксический анализаторы? Какой из анализаторов является ведущим в таком взаимодействии?

Ответ:



Лексический и синтаксический анализаторы взаимодействуют как сопрограммы: ЛА формирует лексемы по исходной программе по запросам СА. Работа ведется под управлением СА.

КРИТЕРИИ: (**а**) За отсутствие схемы или ошибки в ней: **-5**.
(**б**) За отсутствие описания способа взаимодействия: **-3**.
(**в**) Нет ответа или ошибка: **-3**.

8. Описать основные подходы, использующие поведенческие и структурные тесты программ. В чем их основные различия?

Ответ: Поведенческое тестирование основано на технических требованиях, спецификациях. Структурное тестирование проверяет правильность исполнения каждого структурного фрагмента программы (ветви условного оператора, использование каждого элемента данных и т. д.). Для проведения поведенческого тестирования не обязательно знать внутреннюю структуру исследуемой программы, структурные тесты требуют полного доступа к структуре программы.

КРИТЕРИИ: За отсутствие описания какого-либо из двух видов: **-5** (за каждый пропуск).
За пропуск объяснения различия подходов: **-5**.

9. Дать определения бесплодных и недостижимых символов алфавита контекстно-свободной грамматики. Преобразовать контекстно-свободную грамматику G к приведённому виду, выписывая результаты применения каждого шага используемого алгоритма.

G: $S \to Ab \mid Ba$
$A \to Ca$
$B \to Bb \mid Da$
$C \to Aa \mid Bb \mid a \mid b$
$D \to Db \mid Ba$

Ответ: Символ A из алфавита контекстно-свободной грамматики называется *бесплодным*, если множество цепочек терминальных символов им порождаемых пусто ($\{\alpha \mid A \Rightarrow \alpha, \alpha \in T^*\} = \varnothing$). Символ из алфавита контекстно-свободной грамматики называется *недостижимым*, если он не появляется ни в одной сентенциальной форме этой грамматики.

1. Удаление бесплодных символов. Строится множество N, из элементов которого за некоторое число шагов выводятся терминальные символы:

1. $N_0 = \varnothing$
2. $N_1 = \{C\}$, $N_1 \neq N_0$
3. $N_2 = \{A, C\}$, $N_2 \neq N_1$
4. $N_3 = \{A, C, S\}$, $N_3 \neq N_2$
5. $N_4 = \{A, C, S\}$, $N_4 = N_3$

Символы B и D являются бесплодными в грамматике G, их и правила их содержащие можно удалить из грамматики:

G_1: $S \to Ab$ $A \to Ca$ $C \to Aa \mid a \mid b$

2. Удаление недостижимых символов. Строится множество V, в которое помещаются символы, за некоторое число шагов выводимые из символа S:

1. $V_0 = \{A\}$
2. $V_1 = \{A, C\}$, $V_1 \neq V_0$
3. $V_2 = \{A, C\}$, $V_2 = V_1$

Недостижимых символов в грамматике G не обнаружено.

$G_{прив}$: $S \to Ab$ $A \to Ca$ $C \to Aa \mid a \mid b$

КРИТЕРИИ: За отсутствие каждого определения или ошибки в нем: **-3**.
За ошибку при преобразовании: **-7**.

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ПОЛИЗ	&a	x	@	11	*	y	x	@	/	—	+=	;	&x	#+	5	&y	+#

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
—	x	+	1	—	<=	28	!F	1	!	&y	&x	y	8	+	=	7	+	=	;

Ответ: *M:* *a += – x * 11 – y / (– x);*
if (! (x ++ <= 5 – ++ y + x – 1)) goto L;
goto M;
L: *y = (x = y + 8) + 7;*

*do a += – x * 11 – y / (– x); while (x ++ <= 5 – ++ y + x – 1); y = (x = y + 8) + 7;*

КРИТЕРИИ: за неверный ответ на первый вопрос: **-7**.
за неверный ответ на второй вопрос, если первый вариант верен: **-3**, иначе: **-10**

Вариант 3_2010 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to aA \mid A$
$A \to Bb \mid B \mid Ac$
$B \to c \mid Bc$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные	
контекстно-зависимые	
регулярные	
языки типа 0	

(в) Каким из перечисленных классов грамматик принадлежит G?

Класс Π	$G \in \Pi$? (да/нет)
контекстно-свободные	
контекстно-зависимые	
регулярные	
грамматики типа 0	
неукорачивающие	

2. Описать иерархию классов языков по Хомскому. Привести примеры трех языков, относящихся к различным уровням иерархии.

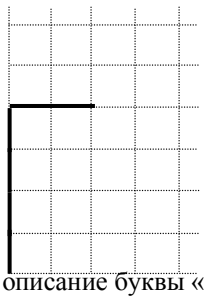
3. Построить по заданной грамматике G конечный автомат (в виде ДС). Если автомат оказался детерминированным, построить анализатор на Си, иначе – преобразовать автомат в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА и по полученному ДКА построить праволинейную грамматику.

G: $S \to A\perp \mid B\perp$
$A \to Aa \mid Ab \mid Ba \mid Bb \mid b$
$B \to Aa \mid Ab \mid Ba \mid Bb \mid a$

4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита $\{c, a, e\}$: c — переместить перо на одну клетку вверх;

a — переместить перо на одну клетку вправо;

e — переместить перо на одну клетку вниз.



Например, по последовательности $ccccaa$ интерпретатор построит букву «Г» размера 4×2, изображенную на рисунке.

Построить грамматику с терминальным алфавитом $\{c, a\}$, описывающую буквы «Г» любого размера $n \times m$, $n,m \geq 1$. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

В построенную грамматику добавить действия вида ⟨***cout*** **<< 'символ';**⟩ так, чтобы описание буквы «Г» размера $n \times m$ переводилось в описание буквы «П» размера $(n+1) \times m$.

5. Восстановить грамматику по заданному анализатору на C++. Сформулировать критерий применимости метода рекурсивного спуска для контекстно-свободных грамматик. Соответствует ли полученная грамматика этому критерию? Обосновать свой ответ. Корректен ли данный анализатор? Обосновать свой ответ (доказать корректность или найти примеры некорректного поведения анализатора).

```
void S () { if (c != 'b')  Error ();
            c = GetL (); A (); B (); C (); D ();
            if (c != 'b')  Error ();
            c = GetL (); A (); D ();
          }
void A () { C(); E(); }
void B () { if (c == 'b') { c = GetL (); C (); }
            else D ();
}
void C () { if (c == 'a') { c = GetL (); D (); }
}
void D () { if (c == 'b') { c = GetL (); E (); }
            else A ();
}
void E () { if (c == 'a') { c = GetL (); B (); }
}
```

6. Описать основные особенности программ, программных продуктов и интегрированных программных продуктов.

7. Перечислить основные виды оптимизации циклов. Привести пример оптимизационного преобразования цикла.

8. Описать деление библиотек систем программирования по функциональному признаку, дать краткие характеристики содержания различных видов библиотек. Признаки каких из перечисленных Вами видов библиотек можно наблюдать в стандартной библиотеке языка C++?

9. Дать определение эквивалентных грамматик. Для контекстно-свободной грамматики G построить эквивалентную ей неукорачивающую грамматику, выписывая результаты применения каждого шага используемого алгоритма.

G: $S \to Ab \mid BaCaC$
$A \to Ca$
$B \to Cb$
$C \to Aa \mid Bb \mid a \mid \varepsilon$

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&t	&p	#+	&q	+#	t	@	/	4	*	+	=	;	t	u	<	34

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	
!F	26	!	&t	+#	;	14	!	&r	p	7	+	–=	;	21	!	&r	r	2	%	=	;	

Вариант 3_2010 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

За каждую задачу — максимум 10 баллов

1. Дана грамматика G:

$S \to aA \mid A$
$A \to Bb \mid B \mid Ac$
$B \to c \mid Bc$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

Ответ: $L(G) = \{ a^i c^n b^j c^m \mid i, j \in \{0,1\}, n \geq 1, m \geq 0\}$

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\Im$	$L(G) \in \Im$? (да/нет)
контекстно-свободные	**да (-1)**
контекстно-зависимые	**да (-1)**
регулярные	**да (-3)**
языки типа 0	**да (-1)**

(в) Каким из перечисленных классов грамматик принадлежит G?

Ответ:

Класс Π	$G \in \Pi$? (да/нет)
контекстно-свободные	**да (-1)**
контекстно-зависимые	**да (-1)**
регулярные	**нет (-3)**
грамматики типа 0	**да (-1)**
неукорачивающие	**да (-1)**

КРИТЕРИИ: (а) описание отсутствует или с ошибками: **–4**
(б), (в) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице**,** обоснование не обязательно

2. Описать иерархию классов языков по Хомскому. Привести примеры трех языков, относящихся к различным уровням иерархии.

Ответ: (**а**) *Тип 3 (регулярные)* $\subset$ *Тип 2 (КС)* $\subset$ *Тип 1 (КЗ)* $\subset$ *Тип 0*
(**б**) При обосновании класса языка можно опираться на утверждения
- языки вида $L=\{a^n \mid n \geq 1\}$ – *типа 3*, регулярные.
- языки вида $L=\{a^n b^n \mid n \geq 1\}$ – *типа 2*, контекстно-свободные.
- языки вида $L=\{a^n b^n c^n \mid n \geq 1\}$ – *типа 1*, контекстно-зависимые.

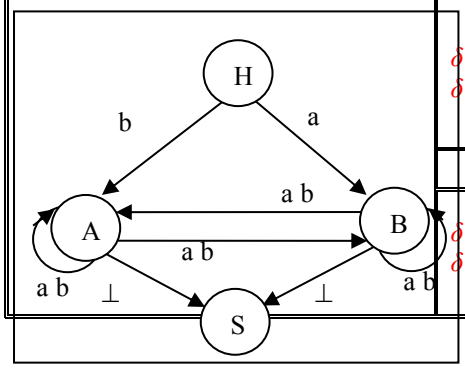
КРИТЕРИИ: (**а**) нет ответа или иерархия приведена с ошибками: **–5**
(**б**) нет ответа; неверный ответ: **–2** (в отношении каждого из трех примеров)

3. Построить по заданной грамматике G конечный автомат (в виде ДС). Если автомат оказался детерминированным, построить анализатор на Си, иначе – преобразовать автомат в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА и по полученному ДКА построить праволинейную грамматику.

G: $S \to A\perp \mid B\perp$
$A \to Aa \mid Ab \mid Ba \mid Bb \mid b$
$B \to Aa \mid Ab \mid Ba \mid Bb \mid a$

Ответ: Автомат недетерминирован
G': $H \to aB \mid bA$
$A \to aC \mid bC$
$B \to aC \mid bC$
$C \to aC \mid bC \mid \perp$

Функция переходов исходного автомата



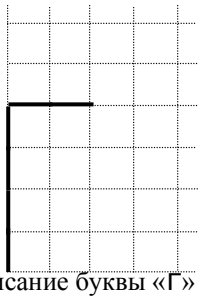
$\delta(H, a) = B$ $\delta(H, b) = A$	$\delta(A, a) = A$ $\delta(A, a) = B$ $\delta(A, b) = A$ $\delta(A, b) = B$ $\delta(A, \perp) = S$	$\delta(B, a) = A$ $\delta(B, a) = B$ $\delta(B, b) = A$ $\delta(B, b) = B$ $\delta(B, \perp) = S$
Функция переходов детерминированного автомата (C ≡ AB)		
$\delta(H, a) = B$ $\delta(H, b) = A$	$\delta(A, a) = C$ $\delta(A, b) = C$ $\delta(B, a) = C$ $\delta(B, b) = C$	$\delta(C, a) = C$ $\delta(C, b) = C$ $\delta(C, \perp) = S$

КРИТЕРИИ: (**а**) нет или неправильная диаграмма состояний исходного автомата: **–5**
(**б**) построен неэквивалентный или недетерминированный автомат: **–5**
(**в**) праволинейная грамматика построена, но неэквивалентная или не соответствующая новой диаграмме: **–5**
(**г**) нет ответа или приведена программа анализатора: **0**

4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*c*, *a*, *e*}: *c* — переместить перо на одну клетку вверх;

a — переместить перо на одну клетку вправо;

e — переместить перо на одну клетку вниз.



Например, по последовательности *ccccaa* интерпретатор построит букву «Г» размера 4×2, изображенную на рисунке.

Построить грамматику с терминальным алфавитом {*c*, *a*}, описывающую буквы «Г» любого размера *n*×*m*, *n*,*m* ≥1. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

В построенную грамматику добавить действия вида ⟨***cout*** **<< 'символ';**⟩ так, чтобы описание буквы «Г» размера *n*×*m* переводилось в описание буквы «П» размера (*n*+1) × *m*.

Ответ: Описание буквы «Г» размера *n*×*m* — это цепочка $c^n a^m$. Следовательно, требуется построить грамматику, порождающую язык $\{c^n a^m \mid n,m \geq 1\}$: $S \to cS \mid cA$

$A \to aA \mid a$

Соглашение: запись ⟨*a*⟩ является сокращением ⟨*cout* << "*a*";⟩
Грамматика с действиями: $S \to c\langle c\rangle S\langle e\rangle \mid c\langle c\rangle\langle c\rangle A\langle e\rangle\langle e\rangle$

$A \to a\langle a\rangle A \mid a\langle a\rangle$

КРИТЕРИИ: порождается пустая цепочка: **–5**
имеется лишняя (непустая) или недостающая цепочка: **–10**
за каждое лишнее правило: **–3**

5. Восстановить грамматику по заданному анализатору на C++. Сформулировать критерий применимости метода рекурсивного спуска для контекстно-свободных грамматик. Соответствует ли полученная грамматика этому критерию? Обосновать свой ответ. Корректен ли данный анализатор? Обосновать свой ответ (доказать корректность или найти примеры некорректного поведения анализатора).

```
void S () { if (c != 'b')  Error ();
            c = GetL (); A (); B (); C (); D ();
            if (c != 'b')  Error ();
            c = GetL (); A (); D ();
          }
void A () { C(); E(); }
void B () { if (c == 'b') { c = GetL (); C (); } else D (); }
void C () { if (c == 'a') { c = GetL (); D (); }           }
void D () { if (c == 'b') { c = GetL (); E (); } else A (); }
```

```
void E () { if (c == 'a') { c = GetL (); B (); }            }
```

Ответ: Восстановленная грамматика имеет вид:

$G = (\{a, b\}, \{A, B, C, D, S\}, P, S)$ P: $S \to bABCDbAD$

$A \to CE$

$B \to bC \mid D$

$C \to aD \mid \varepsilon$

$D \to bE \mid A$

$E \to aB \mid \varepsilon$

Анализатор некорректен: он выдаёт ошибку на правильной цепочке "bb".

Критерий применимости метода рекурсивного спуска для грамматик с эпсилон-правилами:

Метод рекурсивного спуска применим, если и только если для любых двух правил $X \to \alpha \mid \beta$ выполняются следующие условия:

(1) $first(\alpha) \cap first(\beta) = \varnothing$;

(2) справедливо не более чем одно из двух соотношений: $\alpha \Rightarrow \varepsilon$, $\beta \Rightarrow \varepsilon$;

(3) если $\beta \Rightarrow \varepsilon$, то $first(\alpha) \cap follow(X) = \varnothing$.

Грамматика G показанным условиям не соответствует: для двух правил $B \to bC$ и $B \to D$ имеем: $first\ (bC) \cap first\ (D) = \{b\} \cap \{ab\} \neq \varnothing$.

КРИТЕРИИ: Неправильно восстановлена грамматика: **-5**.
Неверная формулировка критерия применимости: **-5**.
Неверный ответ на вопрос о соответствии грамматики критерию применимости: **-3**.
Нет обоснования соответствия/несоответствия грамматики: **-2**.
Неверный ответ на вопрос о корректности анализатора: **-3**.
Нет обоснования корректности (не найден пример "плохой" цепочки): **-2**.

6. Описать основные особенности программ, программных продуктов и интегрированных программных продуктов.

Ответ: Программа создаётся для решения отдельной задачи автором программы и используется в некоторой конкретной операционной среде. Для программы точно определены и зафиксированы решаемая задача, её автор и операционная среда, для которой предназначена программа. Программа неотделима от автора.

Программным продуктом называется программа, работающая без авторского надзора в рамках некоторого набора операционных сред. Программный продукт может исполняться, тестироваться и модифицироваться без участия автора (он отчуждён от автора). Для программного продукта должна быть разработана достаточно полная пользовательская и техническая документация. Программный продукт должен быть настраиваемым.

Системный (интегрированный) программный продукт есть комплекс (взаимосогласованных) программных продуктов (пакет).

КРИТЕРИИ: За отсутствие комментариев к какому-либо виду программ: **-4**.

7. Перечислить основные виды оптимизации циклов. Привести пример оптимизационного преобразования цикла.

Ответ: (1) вынесение инвариантных вычислений из тела цикла, (2) замена операций с переменными цикла, (3) слияние, (4) расщепление и (5) развертывание циклов

Например, расщепление цикла:

```
for (i = 0; i < n; i ++)                          if (x < y)
    {   if (x < y)      { S1; }                       for (i = 0; i < n; i ++) { S1; }
        else            { S2; }     }         else    for (i = 0; i < n; i ++) { S2; }
```

КРИТЕРИИ: (**а**) За отсутствие какого-либо из видов оптимизации: **-2**.
(**б**) За отсутствие примера: **-4**.

8. Описать деление библиотек систем программирования по функциональному признаку, дать краткие характеристики содержания различных видов библиотек. Признаки каких из перечисленных Вами видов библиотек можно наблюдать в стандартной библиотеке языка C++?

Ответ: По функциональному наполнению современные библиотеки делятся на (1) библиотеки функций (процедур) и макроопределений, (2) библиотеки классов и (3) библиотеки компонентов. В свою очередь библиотеки классов могут представлять собой (2.1) совокупности независимых классов, (2.2) иерархии классов, (2.3) иерархии шаблонов классов. Библиотека C++ относится к библиотекам, построенным на основе иерархий шаблонов классов с добавлением отдельных функций и макроопределений.

1. Библиотеки функций (процедур) и макроопределений содержат простейшие (с точки зрения библиотек) составные части, некоторые из которых имеют только неформализованные, семантические связи
2. Библиотеки классов, в зависимости от степени своей организации, содержат более или менее формально связанные и структурированные описания и реализации классов объектов.
3. Библиотеки компонентов содержат тесно связанные описания и реализации сложных программных объектов, предназначенных для совместного использования.

КРИТЕРИИ: За пропуск какого-либо вида библиотек: **-2** (за каждый пропуск из 6-ти).
За пропуск объяснения: **-3** (за каждый из трёх).

9. Дать определение эквивалентных грамматик. Для контекстно-свободной грамматики G построить эквивалентную ей неукорачивающую грамматику, выписывая результаты применения каждого шага используемого алгоритма.

G: $S \to Ab \mid BaCaC\ A \to Ca\ B \to Cb\ C \to Aa \mid Bb \mid a \mid \varepsilon$

Ответ: Грамматики G_1 и G_2 называются *эквивалентными*, если их языки совпадают: $L(G_1) = L(G_2)$.

Строим множество правил X, в которое входят все правила, из которых за некоторое число шагов выводится пустая цепочка:

1. $X_0 = \{C\}$
2. $X_1 = \{C\}$, $N_1 = N_0$

Строим множество правил P', в которое входят все правила, кроме правил вида $R \to \varepsilon$:

P': $S \to Ab \mid BaCaC\ A \to Ca\ B \to Cb\ C \to Aa \mid Bb \mid a$

Дополняем множество правил P' правилами с всевозможными комбинациями вхождений и невхождений символа C, так как в исходной грамматике есть правило $C \to \varepsilon$:

P'': $S \to Ab \mid BaCaC \mid BaaC \mid BaCa \mid Baa$

$A \to Ca \mid a$ $\quad B \to Cb \mid b$ $\quad C \to Aa \mid Bb \mid a$

$P_{неукор} = P''$

КРИТЕРИИ: Нет определения или оно неверно: **0**.
За каждую существенную ошибку при преобразовании: **-3**.

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&t	&p	#+	&q	+#	t	@	/	4	*	+	=	;	t	u	<	34

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
!F	26	!	&t	+#	;	14	!	&r	p	7	+	–=	;	21	!	&r	r	2	%	=	;

Ответ:

```
t = (p ++) + (++ q) / (– t) * 4;
L:      if (! (t < u))      goto E;
                            goto N;
M:      ++ t;               goto L;
N:      r –= p + 7;         goto M;
E:      r = r % 2;
for (t = (p ++) + (++ q) / (– t) * 4; t < u; ++ t) r –= p + 7; r = r % 2;
```

КРИТЕРИИ: за неверный ответ на первый вопрос: **-7**.
за неверный ответ на второй вопрос, если первый вариант верен: **-3**, иначе: **-10**

Вариант 4_2010 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to A \mid Ab \mid Sc$
$A \to Bc \mid Ac$
$B \to a \mid \varepsilon$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

(б) Каким из перечисленных классов принадлежит грамматика G?

Класс П	$G \in$ П? (да/нет)
контекстно-свободные	
контекстно-зависимые	
неукорачивающие	
грамматики типа 0	
регулярные	

(в) Классификация: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком типа k+1.

2. Дать определение неоднозначных контекстно-свободных грамматик и языков. Покажите, что язык L= {a^n | n<=2} однозначен. Можно ли описать этот язык неоднозначной грамматикой? Обоснуйте ответ.

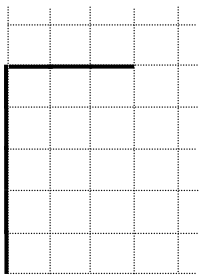
3. Дана функция переходов конечного автомата. Построить по этой функции соответствующую праволинейную грамматику. Если заданный автомат детерминирован, построить анализатор на Си, иначе – преобразовать его в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА, и по полученному ДКА построить леволинейную грамматику.

$\delta(H, a) = A$ $\delta(H, a) = B$ $\delta(H, b) = B$ $\delta(H, \perp) = S$	$\delta(A, a) = A$ $\delta(A, a) = B$ $\delta(A, \perp) = S$	$\delta(B, b) = A$ $\delta(B, \perp) = S$

4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {c, a, e}: c — переместить перо на одну клетку вверх;

a — переместить перо на одну клетку вправо;

e — переместить перо на одну клетку вниз.



Например, по последовательности $ccccaaa$ интерпретатор построит букву «Г» размера 5×3, изображенную на рисунке.

Построить грамматику с терминальным алфавитом {c, a}, описывающую буквы «Г» любого размера $n \times m$, $n,m \geq 1$. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

В построенную грамматику добавить действия вида ⟨***cout*** **<< 'символ';**⟩ так, чтобы описание буквы «Г» размера $n \times m$ переводилось в описание буквы «П» удвоенного размера.

5. Восстановить грамматику по заданному анализатору на C++. Соответствует ли полученная грамматика каноническому виду? Обосновать свой ответ. Корректен ли данный анализатор? Обосновать свой ответ (доказать корректность или найти примеры некорректного поведения анализатора).

```
void S () { if (c == 'q') { c = GetL ();  A ();
                            if (c != 'p') Error ();
                            c = GetL ();  C();
                            if (c != 'q') Error ();
                            c = GetL ();
                          }
            else Error ();
          }
void A () { if (c == 'p') { c = GetL ();  A(); }
            else B();
}
void B () { if (c == 'q') { c = GetL ();  B(); }
            else C();
}
void C () { if (c == 'q') { c = GetL ();  C(); }
}
```

6. Описать основные схемы процесса разработки программного продукта, указав их главные особенности.

7. Какие области памяти выделяются компиляторами для хранения локальных данных процедур? Какая дисциплина распределения памяти при этом реализуется? Какой способ использования выбирается для этих областей?
8. Описать основное назначение редакторов связей

9. Выполнить преобразование грамматики *G* к каноническому виду для метода рекурсивного спуска, выписывая результаты применения каждого последовательного шага преобразования.

G: $S \to aAb \mid aBc$
$A \to Aa$
$B \to bC$
$C \to aA \mid b$

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	n	0	<	38	!F	x	y	+	z	—	4	>	23	!F	&x	n	2

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	
+	=	;	38	!	z	0	>	34	!F	&x	2	=	;	38	!	&y	z	=	;		

Вариант 4_2010 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

За каждую задачу — максимум 10 баллов

1. Дана грамматика G:

$S \to A \mid Ab \mid Sc$
$A \to Bc \mid Ac$
$B \to a \mid \varepsilon$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).
Ответ: $L(G) = \{ a^i c^n b^j c^m \mid i, j \in \{0,1\}, n \geq 1, m \geq 0\}$

(б) Каким из перечисленных классов принадлежит грамматика G?

Ответ:

Класс П	$G \in$ П? (да/нет)
контекстно-свободные	**да (-1)**
контекстно-зависимые	**нет (-1)**
неукорачивающие	**нет (-1)**
грамматики типа 0	**да (-1)**
регулярные	**да (-3)**

(в) Классификация: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком типа $k+1$.
Ответ: $k = 3$ (поскольку для него существует леволинейная, то есть регулярная грамматика)

КРИТЕРИИ: (а) описание отсутствует или с ошибками: **–4**
(б) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице (обоснование не обязательно)
(в) нет ответа; неверный ответ: **–4** (обоснование не обязательно)

2. Дать определение неоднозначных контекстно-свободных грамматик и языков. Покажите, что язык L= {a^n | n<=2} однозначен. Можно ли описать этот язык неоднозначной грамматикой? Обоснуйте ответ.

Ответ:

(**а**) КС-грамматика G называется неоднозначной, если существует (хотя бы одна) цепочка $\alpha \in L(G)$, для которой может быть построено два или более различных деревьев вывода.
(**б**) Этот язык однозначен, так как для него можно построить однозначную грамматику:

$S \to \varepsilon$

$S \to a$

$S \to aa$

(**в**) Да, можно. Для этого языка существует неоднозначная грамматика:

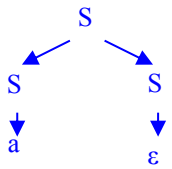
$S \to SS$

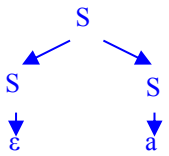
$S \to \varepsilon$

$S \to a$

$S \to aa$

(**г**) Порождаемая грамматикой цепочка "a" имеет два разных дерева вывода:



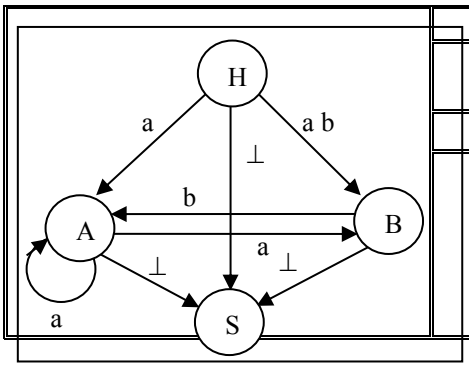


КРИТЕРИИ: (**а**) нет ответа; определение с ошибками: **–5**
(**б**) нет ответа; неверный ответ, нет грамматики-примера: –3
(**в**) нет ответа; неверный ответ: –6
(**г**) нет обоснования неоднозначности грамматики-примера: **–2**

3. Дана функция переходов конечного автомата. Построить по этой функции соответствующую праволинейную грамматику. Если заданный автомат детерминирован, построить анализатор на Си, иначе – преобразовать его в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА, и по полученному ДКА построить леволинейную грамматику.

$\delta(H, a) = A$ $\delta(H, a) = B$ $\delta(H, b) = B$ $\delta(H, \perp) = S$	$\delta(A, a) = A$ $\delta(A, a) = B$ $\delta(A, \perp) = S$	$\delta(B, b) = A$ $\delta(B, \perp) = S$

Ответ: Автомат недетерминирован:



Праволинейная грамматика исходного автомата	
$H \to aA \mid aB \mid bB \mid \perp$	$A \to aA \mid aB \mid \perp$ $B \to bA \mid \perp$

Функция переходов детерминированного автомата (C ≡ AB)		
$\delta(H, a) = C$ $\delta(H, b) = B$ $\delta(H, \perp) = S$	$\delta(B, b) = A$ $\delta(B, \perp) = S$ $\delta(A, a) = C$ $\delta(A, \perp) = S$	$\delta(C, a) = C$ $\delta(C, b) = A$ $\delta(C, \perp) = S$

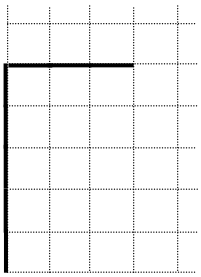
Глев: $S \to A\perp \mid B\perp \mid C\perp \mid \perp$
$A \to Bb \mid Cb$
$B \to b$
$C \to Aa \mid Ca \mid a$

КРИТЕРИИ: **(а)** нет или неправильная праволинейная грамматика исходного автомата: **–5**
(б) построен неэквивалентный или недетерминированный автомат: **–5**
(в) леволинейная грамматика построена, но неэквивалентная или не соответствующая новой диаграмме: **–5**
(г) нет ответа или приведена программа анализатора: **0**

4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*c*, *a*, *e*}: *c* — переместить перо на одну клетку вверх;

a — переместить перо на одну клетку вправо;

e — переместить перо на одну клетку вниз.



Например, по последовательности *ccccaaa* интерпретатор построит букву «Г» размера 5×3, изображенную на рисунке.

Построить грамматику с терминальным алфавитом {*c*, *a*}, описывающую буквы «Г» любого размера *n*×*m*, *n*,*m* ≥1. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

В построенную грамматику добавить действия вида ⟨***cout*** **<< 'символ';**⟩ так, чтобы описание буквы «Г» размера *n*×*m* переводилось в описание буквы «П» удвоенного размера.

Ответ: Описание буквы «Г» размера *n*×*m* — это цепочка $c^n a^m$. Следовательно, требуется построить грамматику, порождающую язык $\{c^n a^m \mid n,m \geq 1\}$: $S \to cS \mid cA$

$A \to aA \mid a$

Соглашение: запись ⟨*a*⟩ является сокращением ⟨*cout* << “*a*”;⟩
Грамматика с действиями: $S \to c\langle c\rangle\langle c\rangle S\langle e\rangle\langle e\rangle \mid c\langle c\rangle\langle c\rangle A\langle e\rangle\langle e\rangle$
$A \to a\langle a\rangle\langle a\rangle A \mid a\langle a\rangle\langle a\rangle$

КРИТЕРИИ: порождается пустая цепочка: **–5**
имеется лишняя (непустая) или недостающая цепочка: **–10**
за каждое лишнее правило: **–3**

5. Восстановить грамматику по заданному анализатору на C++. Соответствует ли полученная грамматика каноническому виду? Обосновать свой ответ. Корректен ли данный анализатор? Обосновать свой ответ (доказать корректность или найти примеры некорректного поведения анализатора).

```
void S () { if (c == 'q') { c = GetL ();  A ();
```

```
                        if (c != 'p') Error ();
                        c = GetL ();  C();
                        if (c != 'q') Error ();
                        c = GetL ();
                      }
          else Error ();
        }
void A () { if (c == 'p') { c = GetL ();  A(); } else B(); }
void B () { if (c == 'q') { c = GetL ();  B(); } else C(); }
void C () { if (c == 'q') { c = GetL ();  C(); } }
```

Ответ: Восстановленная грамматика имеет вид:

G = ({p, q}, {A, B, C, D, S}, P, S) P: $S \to qApCq$

$A \to pA \mid B$

$B \to qB \mid C$

$C \to qC \mid \varepsilon$

Анализатор некорректен: он выдаёт ошибку, например, на правильной цепочке "qppq". Вообще, любая правильная цепочка должна заканчиваться символом 'q', а этот символ может быть зря прочитан процедурой "C()", что приведёт к неправильной трактовке всех правильных цепочек.

Контекстно-свободная грамматика имеет канонический вид для метода рекурсивного спуска, если выполняется критерий применимости этого метода:

Метод рекурсивного спуска применим, если и только если для любых двух правил $X \to \alpha \mid \beta$ выполняются следующие условия:

(1) $first(\alpha) \cap first(\beta) = \varnothing$;

(2) справедливо не более чем одно из двух соотношений: $\alpha \Rightarrow \varepsilon$, $\beta \Rightarrow \varepsilon$;

(3) если $\beta \Rightarrow \varepsilon$, то $first(\alpha) \cap follow(X) = \varnothing$.

Грамматика G каноническому виду не соответствует: для двух правил $A \to pA$ и $A \to B$ имеем:

(1) $first(pA) \cap first(B) = \{p\} \cap \{q\} = \varnothing$;

(2) $B \Rightarrow \varepsilon$; из цепочки aA пустую цепочку вывести нельзя, но

(3) $first(pA) \cap follow(A) = \{p\} \cap \{pq\} = \{p\} \neq \varnothing$

КРИТЕРИИ: Неправильно восстановлена грамматика: **-5**.
Неверный ответ на вопрос о соответствии грамматики каноническому виду: **-3**.
Нет обоснования соответствия/несоответствия грамматики: **-2**.
Неверный ответ на вопрос о корректности анализатора: **-3**.
Нет обоснования корректности (не найден пример "плохой" цепочки): **-2**.

6. Описать основные схемы процесса разработки программного продукта, указав их главные особенности.

Ответ: Основные схемы: нисходящая (идеальный случай, последовательность слабо связанных шагов обработки программ), каскадная (циклическое повторение основных шагов разработки), каскадно-возвратная (оценка ситуации после каждого шага с возможным возвратом на ранее пройденные стадии разработки), спиральная (разработка прототипов и выпуск постепенно развивающихся версий)

КРИТЕРИИ: За пропуск какой-либо из четырех схем: **-3**.
За отсутствие комментариев к какой-либо схеме: **-2**.

7. Какие области памяти выделяются компиляторами для хранения локальных данных процедур? Какая дисциплина распределения памяти при этом реализуется? Какой способ использования выбирается для этих областей?

Ответ: Для хранения локальных данных процедур используется (1) динамическая память (2) со стековой дисциплиной распределения, (3) относящаяся к локальным областям памяти

КРИТЕРИИ: Неправильный ответ, нет ответа на любой вопрос: **-4**.

8. Описать основное назначение редакторов связей

Ответ: (1) Связывание между собой объектных модулей, составляющих единую программу, (2) подготовка таблиц относительных адресов объектов, входящих в общие для них зоны памяти, (3) статическое подключение библиотек.

КРИТЕРИИ: За пропуск какой-либо задачи компонента: **-4** (за каждый пропуск).

9. Выполнить преобразование грамматики G к каноническому виду для метода рекурсивного спуска, выписывая результаты применения каждого последовательного шага преобразования.

G: $S \to aAb \mid aBc$

$A \to Aa$

$B \to bC$
$C \to aA \mid b$

Ответ: 1. Избавляемся от левой рекурсии в правиле для символа A:
G: $S \to aAb \mid aBc \quad A \to aA \mid \varepsilon \quad B \to Bc \quad C \to aA \mid b$
2. В правилах для символа S проводим объединение альтернатив, начинающихся с одинаковых терминальных символов:
G: $S \to aS^1 \quad S^1 \to Ab \mid Bc \quad A \to aA \mid \varepsilon \quad B \to bC \; C \to aA \mid b$
3. Подставляем правила для символов A и B в те правила для символов S^1, которые не начинаются с терминальных символов:
G: $S \to aS^1 \quad S^1 \to aAb \mid b \mid bCc \quad A \to aA \mid \varepsilon \quad B \to bC \; C \to aA \mid b$
4. В правилах для символа S^1 проводим объединение альтернатив, начинающихся с одинаковых терминальных символов:
G: $S \to aS^1 \quad S^1 \to aAb \mid bS^2 \quad S^2 \to Cc \mid \varepsilon \quad A \to aA \mid \varepsilon \quad B \to bC \; C \to aA \mid b$
5. Подставляем правила для символа C в правило для символа S^2, которое не начинается с терминальных символов:
G: $S \to aS^1 \quad S^1 \to aAb \mid bS^2 \quad S^2 \to aAc \mid bc \mid \varepsilon$
$A \to aA \mid \varepsilon \quad B \to bC \quad C \to aA \mid b$
6. Считаем пересечения множеств first и follow для символов A и S^2:
first (A) ∩ follow (A) = {a} ∩ {b, c} = ∅
first (S^2) ∩ follow (S^2) = {a, b} ∩ {ε} = ∅

КРИТЕРИИ: За каждую существенную ошибку при преобразовании: **-3**.

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	n	0	<	38	!F	x	y	+	z	—	4	>	23	!F	&x	n	2

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
+	=	;	38	!	z	0	>	34	!F	&x	2	=	;	38	!	&y	z	=	;	

Ответ:

```
        if (! (n > 0))          goto E;
        if (! (x + y – z > 4))  goto L;
        x = n + 2;              goto E;
L:      if (! (z > 0))          goto M;
        x = 2;                  goto E;
M:      y = z;
E:
```

if (n < 0) if (x + y – z > 4) x = n + 2; else if (z > 0) x = 2; else y = z;
КРИТЕРИИ: за неверный ответ на первый вопрос: **-7**.
за неверный ответ на второй вопрос, если первый вариант верен: **-3**, иначе: **-10**

2011 год
Коллоквиум

Вариант 1_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Если есть ошибки в операторах функции `main()`, исправьте их с помощью операции указания области видимости `"::"`. Вычеркните операторы, которые не удалось исправить с помощью `"::"`. Для оставшихся операторов с помощью операции `"::"` обозначьте ту область видимости, к которой относится вызываемый метод.

```
int x = 4;
class A { int x; public: A (int n = 1);
          int f (int a = 0, int b = 0);   };
class B:  public  A { int x; public: B (int n = 2);
           int f (int a);                  };
class C:  public  B { int x; public: C (int n = 3);
           int f (int a, int b);
           int g (A * p);     };
A * p; B b; C c;
int main () { p = & b;
              x = c.g (& c);
              x = c.f ();
              x = c.f (x);
              x = c.f (x, 1);
              x = p -> f ();
              x = p -> f (x);
              x = p -> f (x, 1);  return 1; }
```

2. Если есть ошибки в приведенной программе, то **объясните,** в чем они заключаются. Ошибочные операторы или ключевые слова вычеркните (допускается не более двух вычеркиваний). Что будет выдано в стандартный поток вывода при работе получившейся программы?

```
  class A {public:  static void f (int x) { h (x); cout << "A::f," << x <<
endl; }
                          void g ()      {        cout << "A::g"       <<
endl; }
                          void h (int x) { g ();  cout << "A::h," << x <<
endl; }
  };
  class B: virtual public A {
           public:  static void f (int x) { h (x); cout << "B::f," << x <<
endl; }
                          void g ()      {        cout << "B::g"       <<
endl; }
                          void h (int x) { g ();  cout << "B::h," << x <<
endl; }
  };
  int main( ){ B::f (0); B b; A * p = & b;
      p -> f (1);
      p -> g ();
      p -> h (2);
      A::f (3);  return 0; }
```

3. Для каждого вызова перегруженной функции с одним параметром укажите шаг алгоритма, на котором будет выбрана наиболее подходящая функция или выявлена неоднозначность. Для выбранной функции укажите ее прототип.

```
int f (int a = 0) { return a; }
int f (double a) { return a; }
int main () { short int s; int i; bool b; enum e {A, B, C}; float f = 1.0f;
      f ();
      f (s);
      f (f);
      f (b);
      f (A);
}
```

4. Для объектов из задания **1** определить, какие конструкторы и деструкторы и в каком порядке будут выполняться при работе следующего фрагмента программы:

```
int main () { C c; A a = c; struct D { B b; D (): b (5) {} } } d; }
```

5. В открытой части класса *cl*, предназначенного для работы со сложными объектами, имеются конструктор и функция преобразования. Для облегчения работы с объектами этого класса создаётся дополнительный класс *ObjPtr*, в закрытую часть которого включается поле указателя на класс *cl*.

```
template <class T> class cl { /* ... */
   public: cl (const T s = 0);
           operator T ();
};
template <class T> class ObjPtr { /* ... */ cl<T> * c; public: /* ...
*/
};
```

Для класса *"ObjPtr"* написать реализацию операций сложения *"+"* таким образом, чтобы правильными оказались выражения (обязательно использовать и методы, и внешние функции):

```
ObjPtr <int> t1, t2;
/* ... */ (t1 + 3) + (5 + t1) + (t1 + t2) /* ... */
```

6. Что такое виртуальный базовый класс и для чего он используется в Си++? Как синтаксически описываются классы, объявляющие и использующие виртуальные базовые классы?

7. Что будет выдано в стандартный поток вывода при работе следующей программы?

```
void f(X & x, int n);
struct X { X () { try  { f(*this, -1); cout << "a"; }
                  catch (X){ cout << "b"; }
                  catch (int){ cout << "c"; }
                }
           X (X &) { cout << "d"; }
  virtual ~X () { cout << "e"; }  };
struct Y: X { Y () { try { f (*this, 0); cout << "f"; }
                     catch (Y) { cout << "g"; }
                     catch (int){ cout << "h"; }
                     cout << "i"; }
              Y (Y &){ cout << "j"; }
             ~Y (){ cout << "k"; }       };
void f(X & x, int n) { try  { if (n < 0) throw -n;
                              else if (n == 0) throw x;
                              else throw n; }
                       catch (int){ cout << "l"; } }
int main() { try { Y a; }
             catch (...){ cout << "m"; return 1; }
             cout << "n"; return 0;
           }
```

8. Если есть ошибки в следующем фрагменте, то в чем они заключаются? Исправьте ошибки, ничего не удаляя, добавив в общей сложности не более 14 символов.

```
class A                     { public: int * n; int m; };
class B: public A           { public: int * p; };
class C: public A           { public: int * c; };
class D: public B: public C { public: int * e; };
int main () {
      D fA,
      * f = new D;
      fA.m =0;
      return *((* f).e = & fA.m);  }
```

9. Какие виды пользовательских преобразований допустимы в программах на языке Си++? Какие ограничения накладываются на использование таких преобразований при разрешении статического полиморфизма функций? Привести пример фрагмента программы с использованием этих операций.

10. Описать функцию *g()* с двумя параметрами: контейнер-список целых элементов и контейнер-вектор указателей на целые. Функция должна, последовательно проходя по элементам контейнеров от начала к концу вектора и от конца к началу списка, менять местами элементы (целые значения) контейнеров, после чего распечатать в прямом порядке целые значения сначала для списка, затем для вектора. Функция возвращает количество переставленных элементов контейнеров.

Ответы_Вариант 1_2011

1. Ответ:

```
x = c.C::g (& c);
  x = c.A::f ();
  x = c.A::f (x);                    //   x = c.B::f (x);
  x = c.C::f (x, 1);                      x = p -> A::f ();
  x = p -> A::f (x);                      x = p -> A::f (x, 1);
```

Ошибочны 2 оператора, которые можно исправить (подчеркнуты).

Критерии: За пропуск ошибки (из 2-х): -4. За каждое не отмеченное или неверное уточнение области видимости (из 7): -1. За каждую лишнюю обнаруженную "ошибку" -4.

2. Ответ: Ошибочными являются указания на то, что статические методы *f()* вызывают нестатические методы. Надо исключить в методах *f()* вызовы методов *h()*:

```
B::f,0      A::f,1      A::g        A::g        A::h,2      A::f,3
```

Критерии: За каждую не найденную ошибку, лишнюю ошибку или неправильно вызванную функцию – 2.

3. Ответ:

```
f ();   // f (int),     (а) на этапе выбора функций,
                           подходящих по кол-ву параметров;
                           за ответ «точное отождествление»
                           не наказывать
f (s);      // f (int),     (б) целочисленное расширение
f (f);      // f (double),  (б) вещественное расширение
f (b);      // f (int),     (б) целочисленное расширение
f (A);      // f (int),     (б) целочисленное расширение
```

Критерии: За каждую ошибку **–2**.

4. Ответ: `A(),B(),C(),A(const A&),A(),B(),D(),~D(),~B(),~A(),~A(),~C(),~B(),~A().`

Критерии: За каждую ошибку **–2**.

5. Ответ:

```
template<class T> class ObjPtr { /* ... */ cl<T> * c; public: /* ... */

    const ObjPtr<T> operator+ (const ObjPtr<T> & s) const
       { ObjPtr<T> v; v.c = new cl<T>; * v.c = * c + * s.c; return v; }
    const ObjPtr<T> operator+ (const T & s)
       { ObjPtr<T> v; v.c = new cl<T>; * v.c = * c + s;     return v; }
    friend const ObjPtr<T> operator+ (const T & s, const ObjPtr<T> & t);
};
template<class T> const ObjPtr<T> operator+ (const T &s, const ObjPtr<T>
&t)
    { ObjPtr<T> v; v.c = new cl<T>;  * v.c = * t.c + s;  return v; }
```

Критерий: За отсутствие каждого примера (из 3-х) или ошибки в нем: **-4.** За отсутствие реализаций методами класса (из 2-х) при правильной реализации этих же операций функциями-друзьями: **-2.**

6. Ответ: 1. Виртуальный базовый класс – это такой базовый класс, который представляется в своих производных классах одним и тем же совместно используемым объектом. В варианте наследования (множественного) с использованием виртуальных базовых классов одинаковые части нескольких базовых классов включаются в их общий производный класс в единственном экземпляре. Виртуальное наследование позволяет избавиться от неоднозначности, которое при множественном наследовании возникает из-за наследования одного базового класса несколькими классами промежуточных наследственных уровней (как при ромбовидном наследовании).

2. Пример виртуального наследования (без виртуальности обращения к полю "*n*" из функции "*f()*" создают неоднозначность), где виртуальным базовым классом является класс "*L*":

```
class L { public int n; ... };
class A: virtual public L { ... };
class B: virtual public L { ... };
class C: public A, protected B { ... void f (); ... };
```

Критерий: **За неверный ответ: -5. За отсутствие пояснения: -4.. За отсутствие примера: -4.**

7. Ответ: `l a d e m e`

Критерии: За каждую неверную (лишнюю или пропущенную) печать -2.

8. Ответ: **1.** Требуется уточнить обращения к полю "*m*" с помощью операций разрешения области видимости "*::*" для класса "*B*" либо для "*C*", например, так:

```
fA.C::m = 0; return *((* f).e = & fA.C::m);
```

Либо во всех описания производных классов вставить указание виртуального наследования (всего 4 поправки, из которых исправления в классах "*B*" и "*C*" при таком подходе – обязательны):

```
class B: virtual public A { public: int * p; };
class C: virtual public A { public: int * c; };
```

3. Других ошибок нет.

Критерий: **За пропуск любой из трёх ошибок или неправильное исправление: -4. За указание ошибки там, где ее нет: -5. Отсутствие указаний виртуальности наследования в классе "*D*" при наличии таких исправлений в классах "*B*" и "*C*" не наказывать.**

9. Ответ: К пользовательским преобразованиям относятся преобразования, выполняемые:

- с помощью конструкторов преобразования (то есть конструкторов с одним параметром);
- с помощью функций преобразования – методов класса с профилем "***operator*** <*mun*> *()*", которые преобразуют объект класса к типу <*mun*>:

Ограничения:

- Пользовательские преобразования применяются неявно только в том случае, если они однозначны.
- Допускается не более одного пользовательского преобразования для обработки одного вызова для одного параметра.
- Конструктор должен иметь разрешение быть вызванным неявно (у него должен отсутствовать спецификатор explicit).

```
struct S { S (long); operator int (); };
void f (long);          void f (char*);
void g (S);             void g (char*);
void h (const S&);      void h (char*);
void ex (S &a) { f (a); // f ((long) (a.operator int()));
                 g (1); // g (S ((long) (1)));
                 h (1); // h (S ((long) (1)));
}
```

Критерий: **За пропуск какого-либо преобразования или ограничения (из 5): -2. За пропуск примера какой-либо операции: -4.**

10. Ответ:

```
typedef   list<int>   L;
typedef vector<L::value_type *> V;

L::size_type g (V & vect, L & lst)
{ L:: size_type count = 0;
  L:: reverse_iterator lp = lst.rbegin ();
  for (V:: size_type i = 0; i < vect.size () && lp != lst.rend (); ++ i, ++ lp)
     { L::value_type t = * lp;
      * lp = * vect [i];
      * vect [i] = t;
      ++ count;
     }
  L:: iterator lq = lst.begin ();
  while (lq != lst.end ()) {
```

```
      cout << * lq;
      ++ lq;
 }
  for (V::size_type i = 0; i < vect.size (); ++ i)
      cout << * vect [i];
  return count;
}
```

КРИТЕРИИ: За каждую существенную ошибку: -3. Отсутствие ссылки в списке формальных параметров - -3. Если вместо индексации использованы указатели, но программа правильная – не снижать. Использование вместо типа *size_type* типа *int* - не наказывать.

Вариант 2_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Если есть ошибки в операторах функции *main()*, исправьте их с помощью операции указания области видимости *"::"*. Вычеркните операторы, которые не удалось исправить с помощью *"::"*. Для оставшихся операторов с помощью операции *"::"* обозначьте ту область видимости, к которой относится вызываемый метод.

```
int x = 4;
class A { int x; public: A (int n = 3);
          int f (int a = 0, int b = 0);  };
class B:  public  A { int x; public: B (int n = 1);
          int f (int a = 0);             };
class C:  public  B { int x; public: C (int n = 2);
          int f (int a, int b);
          int g (A * p)                  };
A * p; B b; C c;
int main () { p = & b;
              x = c.g (& b);
              x = c.f ();
              x = c.f (x);
              x = c.f (x, 1);
              x = p -> f ();
              x = p -> f (x);
              x = p -> f (x, 1);
              return 2;  }
```

2. Есть ли ошибки в интерфейсах классов *C* и *D* программы на C++? Если есть, то **объясните,** в чем они заключаются и внесите нужные исправления, оставив без изменения реализацию классов и функции *"main"*. Что будет выдано в стандартный поток вывода при работе получившейся программы?

```
class C {public:
                  C (int x = 0)      {}
    virtual int   f (int x) { cout << "C::f," << x << endl; return h (x); }
    virtual int   g      () { cout << "C::g"       << endl; return 1; }
    virtual int   h (int x) { cout << "C::h," << x << endl; return x; }
    virtual operator int () { return  99; }         };
class D: public C { public:
            int   f (int x) { cout << "D::f," << x << endl; return h (x); }
            int   g (int x) { cout << "D::g"       << endl; return 1; }
            int   h (int x) { cout << "D::h," << x << endl; return x; }
                  D (int x = 0)      {}
            operator int () { return 100; }         };
int main( ){ const D d; C const * const t = & d;
     t -> f (3);
     t -> f (d);
     t -> g ();
     t -> h (5);  return 0; }
```

3. Для каждого вызова перегруженной функции с одним параметром укажите шаг алгоритма, на котором будет выбрана наиболее подходящая функция или выявлена неоднозначность. Для выбранной функции укажите ее прототип.

```
int f(double a = 1.0){return a;}
int f(long double a = 5.0){return a;}
int main () { float f = 1.0f;   double d = 2.0;   long double ld = 3.0l;
      f ();
      f (4);
      f (f);
      f (d);
      f (ld);
}
```

4. Для объектов из задания **1** определить, какие конструкторы и деструкторы и в каком порядке будут выполняться при работе следующего фрагмента программы:

```
int main () { class D { B b; A a; public: D (): a (b) { } } d; C c; }
```

5. В открытой части класса *cl*, предназначенного для работы со сложными объектами, имеются конструктор и функция преобразования. Для облегчения работы с объектами этого класса создаётся дополнительный класс *ObjPtr*, в закрытую часть которого включается поле указателя на класс *cl*.

```
template <class T> class cl { /* ... */
   public: cl (const T s = 0);
           operator T ();
};
template <class T> class ObjPtr { /* ... */ cl <T> * c; public: /* ...
*/
};
```

Для класса *"ObjPtr"* написать реализацию операций сравнения на равенство *"=="* таким образом, чтобы правильными оказались выражения (обязательно использовать и методы, и внешние функции):

```
ObjPtr<int> t1, t2;
/* ... */ (t1 == 3) || (5 == t1) && (t1 == t2) /* ... */
```

6. Что такое наследование и для чего оно используется в Си++? Какой длины и какого вида цепочки наследования можно строить в Си++? Как синтаксически описывается наследственная иерархия разных видов в Си++?

7. Что будет выдано в стандартный поток вывода при работе следующей программы?

```
void f(X & x, int n);
struct X { X () { try  { f(*this, 0); cout << "a"; }
                  catch (X){ cout << "b"; }
                  catch (int){ cout << "c"; }  }
           X (X &){ cout << "d"; }
  virtual ~X () { cout << "e"; }  };
struct Y: X { Y () { try { f (*this, 0); cout << "f"; }
                     catch (Y) { cout << "g"; }
                     catch (int) { cout << "h"; }
                     cout << "i"; }
              Y (Y &) { cout << "j"; }
             ~Y () { cout << "k"; }
};
void f(X & x, int n) { try  { if (n < 0) throw -n;
                              else if (n == 0) throw x;
                              else throw n; }
                       catch (int) { cout << "l"; }
}
int main() { try { Y a; }
             catch (...) { cout << "m"; return 1; }
             cout << "n";
             return 0;
           }
```

8. Если есть ошибки в следующем фрагменте, то в чем они заключаются? Исправьте ошибки, ничего не удаляя, добавив в общей сложности не более 12 символов.

```
class S    { public int s; void sp (int si) { s = si; }};
class T: S { public int t; void tp (int ti) { t = ti; s = ti; }};
class U: T { public int u; void up (int ui) { u = ui; t = ti; s = ti; }};
void g ()  {      U * pu = new U;
                  T * pt = pu;
                  S * ps = pu; }
```

9. Что такое “дружественная функция”? Описать основное отличие функций-друзей от методов классов. Привести пример использования дружественной функции.

10. Написать функцию *g()* с параметром, представляющим собой контейнер-список указателей на элементы длинного целого типа. Функция, просматривает контейнер от конца к началу и меняет знак значения, на которое указывает текущий указатель, если следующее за ним значение отрицательно. Например, список (-1,-2,3,4) превратится в (-1,2,-3,4). Затем функция печатает значения, на которые указывают элементы контейнера в прямом порядке. Функция возвращает число измененных элементов.

Ответы_Вариант 2_2011

1. Ответ:

```
x = c.C::g (& b);
    x = c.A::f ();          x = c.B::f ();
    x = c.A::f (x);         x = c.B::f (x);
    x = c.C::f (x, 1);      x = p -> A::f ();
    x = p -> A::f (x);      x = p -> A::f (x, 1);
```

Ошибочны 2 оператора, которые можно исправить (подчеркнуты).

Критерии: За пропуск ошибки (из 2-х): -3. За каждое не отмеченное или неверное уточнение области видимости (из 7): -1. За каждую лишнюю обнаруженную "ошибку" -3.

2. Ответ: 1. Чтобы функцию *f()* можно было вызывать для констант (*t->f(3)*), надо поставить модификатор *const* после списка формальных параметров этой функции в классе *C*. Чтобы из функции *C::f()* можно было вызвать функцию *C::h()*, надо поставить модификатор *const* после списка формальных параметров функции *C::h()*. Чтобы функцию преобразования *operator int()* можно было вызывать для констант (*t->f(d)*), надо поставить модификатор *const* после списка формальных параметров этой функции в классе *D*. Чтобы функцию *g()* можно было вызывать для констант (*t->g()*), надо поставить модификатор *const* после списка формальных параметров этой функции в классе *C*. При таких исправлениях все функции станут невиртуальными, и будет выдано в поток:

```
C::f,3      C::h,3      C::f,100      C::h,100      C::g      C::h,5
```

Если будут сделаны синхронные исправления описания функций *f()*, *g()* и *h()*, в классе *D*, чтобы функции остались виртуальными, будет выдано в поток:

```
D::f,3      D::h,3      D::f,100      D::h,100      C::g      D::h,5
```

Критерии: За каждое неправильное исправление и неправильно вызванную функцию -2.

3. Ответ:

```
f ();       //    (а) неоднозначность определения числа параметров
f (4);      //    (в) неоднозначность при стандартном преобразовании
f (f);      // f (double), (б) вещественное расширение
f (d);      // f (double), (а) точное отождествление
f (ld);     // f (long double), (а) точное отождествление
```

Критерии: За каждую ошибку –2.

4. Ответ: `A(),B(),A(const A&),D(),A(),B(),C(),~C(),~B(),~A(),~D(),~A(),~B(),~A().`

Критерии: За каждую ошибку –2.

5. Ответ:

```
template<class T> class ObjPtr { /* ... */ cl<T> * c; public: /* ... */

    const bool operator==(const ObjPtr<T> & s) const
        { return * c == * s.c; }
    const bool operator==(const T & s)
        { return * c == s; }
    friend const bool operator==(const T & s, const ObjPtr<T> & t);
};
```

```
template<class T> const bool operator==(const T & s, const ObjPtr<T> &
t)
   { return * t.c == s; }
```

Критерий: **За отсутствие каждого примера (из 3-х) или ошибки в нем: -4. За отсутствие реализаций методами класса (из 2-х) при правильной реализации этих же операций функциями-друзьями: -2.**

6. Ответ: **1.** Наследование есть способ упорядочения абстракций, путём создания иерархии классов. Наследование позволяет производному объекту наследовать от базового объекта общие атрибуты, определяя собственные дополнительные характеристики.

2. Наследование может одиночным (единичным) и множественным, когда наследуются атрибуты сразу нескольких классов. Ограничений на длину цепочек в языке нет. Одиночное наследование:
```
class A: { ... };  class B: A { ... };  class C: B { ... };
```

3. Множественное наследование:
```
class A: { ... };  class B: { ... };    class C: A, B { ... };
```

Критерий: **За неверный ответ: -5. За отсутствие ответа на вопрос о видах наследования: -5. За отсутствие любого из двух примеров: -2.**

7. Ответ: `d b e d e m e`

Критерии: За каждую неверную (лишнюю или пропущенную) печать -2.

8. Ответ: **1.** Необходимо заменить вид наследования класса *"S"* на открытый, чтобы поле *"s"* стало доступным в классе *"U"*:

```
class T: public S { public int t; void tp (int ti) { t = ti; s = ti; }};
```

2. Необходимо дополнительно заменить вид наследования класса *"T"* на открытый, чтобы стали возможными преобразования указателей в процедуре *"g()"*:

```
class U: public T { public int u; void up (int ui) { u = ui; t = ti; s = ti;
}};
```

3. Открытости наследования можно добиться, заменяя в наследующем классе слово "class" словом "struct".

4. Других ошибок нет.

Критерий: **За пропуск любой из двух ошибок или неправильное исправление: -5. За указание ошибки там, где ее нет: -5.**

9. Ответ:

Функция-друг, не являясь методом класса, имеет доступ к защищённым и закрытым членам класса. Функция может быть объявлена другом нескольких классов, а метод всегда относится к одному классу.

```
class Matrix;
class Vector { friend Vector operator*(const Matrix&, const Vector&); };
class Matrix { friend Vector operator*(const Matrix&, const Vector&); };
Vector operator*(const Matrix&, const Vector&) { ... }
```

Критерий: За пропуск объяснения понятия “дружбы”: **-5**. За пропуск ответа на вопрос об отличиях методов и “друзей”: **-3.** За пропуск примера: **-2.**

10. Ответ:

```
typedef list <long int *> L;
L::size_type g (L & lst)
{ L:: reverse_iterator lp, lq;
  L:: size_type count = 0;
  lp = lst.rbegin ();
  while (lp != lst.rend ())
    { lq = lp;
      ++ lp;
```

```
      if (lp == lst.rend ())
            break;
      if (** lp < 0) {
            ** lq = - ** lq;
            ++ count;
      }
      ++ lp;
    }
  L:: iterator lr = lst.begin ();
  while (lr != lst.end ()) {
      cout << ** lr;
      ++ lr;
  }
  return count;
}
```

КРИТЕРИИ: За каждую существенную ошибку: -3. Отсутствие ссылки в списке формальных параметров - -3. Использование вместо типа *size_type* типа *int* - не наказывать.

Вариант 3_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Если есть ошибки в операторах функции *main()*, исправьте их с помощью операции указания области видимости *"::"*. Вычеркните операторы, которые не удалось исправить с помощью *"::"*. Для оставшихся операторов с помощью операции *"::"* обозначьте ту область видимости, к которой относится вызываемый метод.

```
int x = 4;
class A { int x; public: A (int n = 1);
          virtual int f (int a = 0, int b = 0);  };
class B { int x; public: B (int n = 2);
                   int f (int a = 0); };
class C:  public A, public B { int x; public: C (int n = 3);
                   int f (int a, int b = 0);
                   int g (A * p);
};
A * p; C c;
int main () { p = & c;
              x = c.g (p);
              x = c.f ();
              x = c.f (x);
              x = c.f (x, 1);
              x = p -> f ();
              x = p -> f (x);
              x = p -> f (x, 1);  return 3;  }
```

2. Укажите лишние и ошибочные операции динамического приведения типа, если таковые имеются в функции ***main()***. Дайте необходимые пояснения своим исправлениям.

```
class K           { public:         void g () { cout << "K::g"; } };
class L: public K { public:         void f () { cout << "L::f"; } };
class M: public K { public: virtual void h () { cout << "M::h"; } };
class P: public L { public:         void f () { cout << "P::f"; } };
class Q: public M { public: virtual void h () { cout << "Q::h"; } };
class R: public P { public: virtual void f () { cout << "R::f"; }
                            virtual void h () { cout << "R::h"; } };
class S: public Q { public: virtual void f () { cout << "S::f"; }
                            virtual void h () { cout << "S::h"; } };
int main() {
        S os, * s = & os; K * k; L * l; M * m; P * p; Q * q; R * r;
    k = dynamic_cast <K *>(s);    s = dynamic_cast <S *>(k);
```

```
 l = dynamic_cast <L *>(k);     m = dynamic_cast <M *>(s);
 p = dynamic_cast <P *>(l);     q = dynamic_cast <Q *>(m);
 r = dynamic_cast <R *>(q);     s = dynamic_cast <S *>(p);     return 0; }
```

3. Для каждого вызова перегруженной функции с одним параметром укажите шаг алгоритма, на котором будет выбрана наиболее подходящая функция или выявлена неоднозначность. Для выбранной функции укажите ее прототип.

```
int f(int a = 1){return a;}
int f(long double a = 5.0){return a;}
int main () {
 short int s; int i; bool b; float f = 1.0f; double d = 2.0;
      f (s);
      f (i);
      f (b);
      f (f);
      f (d);  }
```

4. Для объектов из задания **1** определить, какие конструкторы и деструкторы и в каком порядке будут выполняться при работе следующего фрагмента программы:

```
int main () { C c; class D { C c; B b; public: D (): b (c) { } } d; }
```

5. В открытой части класса *cl*, предназначенного для работы со сложными объектами, имеются конструктор и функция преобразования. Для облегчения работы с объектами этого класса создаётся дополнительный класс *ObjPtr*, в закрытую часть которого включается поле указателя на класс *cl*.

```
template <class T> class cl { /* ... */
   public: cl ( const T s = 0);        operator T ();  };
template <class T> class ObjPtr { /* ... */ cl <T> * c; public: /* ...
*/
};
```

Для класса *"ObjPtr"* написать реализацию операции изменения знака *"–"* и операций присваивания *"="* таким образом, чтобы правильными оказались выражения (обязательно использовать и методы, и внешние функции):

```
ObjPtr <int> t1, t2;
/* ... */ t1 = 5; t2 = -t1; /* ... */
```

6. Что такое инкапсуляция и для чего она используется в Си++? Привести синтаксическую запись фрагмента программы, на котором можно проиллюстрировать понятие инкапсуляции.

7. Что будет выдано в стандартный поток вывода при работе следующей программы?

```
void f(X & x, int n);
struct X { X ()    { try  { f(*this, 1);  cout << "a"; }
                     catch (X){ cout << "b"; }
                     catch (int){ cout << "c"; }
                   }
           X (X &){ cout << "d"; }
  virtual ~X () { cout << "e"; }
};
struct Y: X { Y () { try { f (*this, 1);    cout << "f"; }
                     catch (Y){ cout << "g"; }
                     catch (int){ cout << "h"; }
                     cout << "i"; }
              Y (Y &){ cout << "j"; }
             ~Y (){ cout << "k"; }
};
void f(X & x, int n) { try  { if (n < 0) throw -n;
                              else if (n == 0) throw x;
                              else throw n; }
                       catch (int){ cout << "l"; }
}
```

```
int main() { try { Y a; }
             catch (...){ cout << "m"; return 1; }
             cout << "n"; return 0; }
```

8. Если есть ошибки в следующем фрагменте, то в чем они заключаются? Исправьте ошибки, ничего не удаляя, добавив в общей сложности не более 29 символов.

```
class W         { public: int * w; void wf (int * wp) { w = wp; }};
class X: W      { public: int * x; void xf (int * xp) { x = xp; w = xp; }};
class Y: W      { public: int * y; void yf (int * yp) { y = yp; w = yp; }};
class Z: X, Y   { public: int * z; void zf (int * zp) { z = zp; x = zp; y = zp;
}};
void h () { int hi; W * pw; X * px; Y * py; Z * pz;
            pz = new W;
            (*pz).w = & hi;
            pz -> xf ((*pz).X::w); }
```

9. Описать условия включения механизма виртуальности для метода класса. Привести пример записи виртуальной функции и обращения к ней.

10. Написать функцию *g()* с параметром, представляющим собой контейнер-вектор указателей на элементы вещественного типа. Функция просматривает контейнер в прямом порядке и обнуляет текущее указываемое значение, если следующее значение отрицательно. Например, [1, -2, 4, -6, -4, 5] ➔ [0, -2, 0, 0 ,-4, 5]. Далее функция в обратном порядке печатает значения, на которые указывают элементы контейнера. Функция возвращает число измененных значений.

Ответы_Вариант 3_2011

1. Ответ:

```
x = c.g (p);
x = c.A::f ();          //   x = c.B::f ();
x = c.C::f (x);              x = c.C::f (x, 1);
x = p -> C::f ();
x = p -> C::f (x);           x = p -> C::f (x, 1);
```

Ошибочен 1 оператор, который можно исправить (выделен шрифтом).

Критерии: За пропуск ошибки: -4. За каждое не отмеченное или неверное уточнение области видимости (из 7): -1. За каждую лишнюю обнаруженную "ошибку" -4.

2. Ответ:

```
k = s;                      // Приведение производного указателя к
базовому
//  s = dynamic_cast<S *>(k);  // Тип K – не полиморфный
//  l = dynamic_cast<L *>(k);  // Тип K – не полиморфный
    m = s;                     // Приведение производного указателя к
базовому
//  p = dynamic_cast<S *>(l);  // Тип L – не полиморфный
    q = dynamic_cast<Q *>(m);
    r = dynamic_cast<R *>(q);
//  s = dynamic_cast<S *>(p);  // Тип P – не полиморфный
```

Критерии: За каждое не найденное лишнее преобразование (из двух): –2. За каждое не удаленное ошибочное преобразование (из 4-х): –2.

3. Ответ:

```
f (s);     // f (int),     (б) целочисленное расширение
f (i);     // f (int),     (а) точное отождествление
f (b);     // f (int),     (б) целочисленное расширение
f (f);     //               (в) неоднозначность при стандартном
преобразовании
f (d);     //               (в) неоднозначность при стандартном
преобразовании
```

Критерии: За каждую ошибку –2.

4. Ответ: A B C A B C B(B&) D ~D ~B ~C ~B ~A ~C ~B ~A.

Критерии: За каждую ошибку –2.

5. Ответ:

```
template<class T> class ObjPtr { /* ... */ cl<T> * c; public: /* ... */

    ObjectPtr<T> & operator= (const ObjectPtr<T> & s)
       { if (this != & s)
            { delete c; c = new cl<T>; * c = * (s.c); }
         return * this; }
    ObjectPtr<T> & operator= (const int & s)
       { delete c; c = new cl<T>; * c = s;
         return * this; }

    friend const ObjectPtr<T>   operator-  (const ObjectPtr<T> & s);
};
template<class T> const ObjectPtr<T> operator- (const ObjectPtr<T> & s)
   { ObjectPtr<T> v; v.c = new cl<T>; * v.c = (T) 0 - * s.c; return v;
}
```

Критерий: **За отсутствие каждого примера (из 3-х) или ошибки в нем: -4. За отсутствие реализации унарного минуса функцией-другом при правильной реализации этой операции методом класса: -2. За попытку реализации присваивания функцией-другом: -5.**

6. Ответ: **1.** Инкапсуляция обеспечивает сокрытие состояния объекта от остальных объектов, процедур. Инкапсуляция защищает данные от несанкционированного доступа со стороны посторонних алгоритмов. Инкапсуляция изолирует внешнее представление об объекте от его внутренней реализации. Доступ к связанным между собой алгоритмам и данным контролируется интерфейсом (формальным описанием способов доступа к алгоритмам и данным).

2. Пример инкапсуляции (доступ к полю "*n*" возможен только из методов класса и функций-друзей):

```
class A: { int n; public: f () ... };
```

Критерий: За демонстрацию непонимания сущности инкапсуляции: **-5**. За отсутствие примера: **-5.**

7. Ответ:

```
l a l f i k e n
```

Критерий : За каждую неверную (лишнюю или пропущенную) печать **-2.**

8. Ответ: **1.** Чтобы преобразование из класса "*W*" в функции "*h()*" в класс "*Z*" стало возможным, это преобразование надо исправить: `pz = (Z *)(X *)(new W);`

2. Дополнительно необходимо сделать открытым вид наследования класса "*W*" классом "*X*" и класса "*X*" классом "*Z*":

```
class X: public W   { ... };  class Z: public X,Y {  ... };
```

3. Необходимо уточнить область видимости поля "*w*" в теле функции "*h()*":

```
(*pz).X::w=& hi;
```

4. Открытости наследования можно добиться, заменяя в наследующем классе слово "`class`" словом "`struct`".

5. Других ошибок нет.

Критерий: **За пропуск любой из пяти ошибок или неправильное исправление: -2. За указание ошибки там, где ее нет: -5.**

9. Ответ:

1. Имеется иерархия классов, хотя бы из двух классов – базового и производного
2. В базовом классе функция объявлена с ключевым словом ***virtual***
3. В производном классе есть функция с таким же именем, с таким же списком параметров (количество, типы и порядок параметров совпадают) и с таким же типом возвращаемого значения
4. Вызов функции производного класса осуществляется через указатель на объект базового класса (или с помощью ссылки на объект базового класса) без указания самого объекта и уточнения с помощью операции разрешения области видимости

```
class B           { public: virtual void print (); };
class P: public B { public:         void print (); };
 void B::print () { ... }
 void P::print () { ... }

B * ps = new P; ...  ps -> print ();
```

Критерий: **За пропуск любого условия включения механизма виртуальности: -3. За пропуск примера: -2.**

10. **Ответ:**

```
typedef vector <float *> V;
V::size_type g (V & vect)
{ V::size_type count = 0;
  V::size_type vs = vect.size () - 1;
  for (int i = 0; i < vs; i += 2)
    { if (* vect [i + 1] < 0) {
            * vect [i] = 0;
            ++ count; }
    }
  for (int i = vs; i > 0; -- i)
            cout << * vect [i];
  return count;
}
```

КРИТЕРИИ:
За каждую существенную ошибку: -3.
Отсутствие ссылки в списке формальных параметров - -3.
Если вместо индексации использованы указатели, но программа правильная – не снижать. Использование вместо типа *size_type* типа *int* - не наказывать.

Вариант 4_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Если есть ошибки в операторах функции *main()*, исправьте их с помощью операции указания области видимости *"::"*. Вычеркните операторы, которые не удалось исправить с помощью *"::"*. Для оставшихся операторов с помощью операции *"::"* обозначьте ту область видимости, к которой относится вызываемый метод.

```
int x = 4;
class A { int x; public: A (int n = 2);      int f (int a, int b); };
class B { int x; public: B (int n = 3);      int f (int a = 0);    };
class C:  public  B, public  A {      int x;
                                public: C(int n = 1);
                                        int f (int a);
                                        int g (A * p);  };

A * p; B b; C c;
int main () { p = & c;
              x = c.g (& b);              x = c.f ();
              x = c.f (x);                x = c.f (x, 1);
              x = p -> f ();              x = p -> f (x);
              x = p -> f (x, 1);  return 4;         }
```

2. Если есть ошибки в приведенной программе, то **объясните,** в чем они заключаются. Ошибочные операторы или ключевые слова вычеркните (допускается не более двух вычеркиваний). Что будет выдано в стандартный поток вывода при работе получившейся программы?

```
void h (int x);
void f (int x) { h (x); cout << " ::f," << x << endl; }
```

```
                        void g ()       { h (0); cout << " ::g"       << endl; }
                        void h ()       {        cout << " ::h"       << endl; }
                        void h (int x)  {        cout << " ::h," << x << endl; }
class T {
     public: virtual void f (int x) { h ();  cout << "T::f," << x << endl; }
                     void g ()      { h ();  cout << "T::g"       << endl; }
             virtual void h ()      {        cout << "T::h"       << endl; }
             virtual void h (int x) { g ();  cout << "T::h," << x << endl; }
};
class U: virtual public T {
     public:         void f (int x) { h (x); cout << "U::f," << x << endl; }
             virtual void g ()      {        cout << "U::g"       << endl; }
             virtual void h ()      {        cout << "U::h"        << endl;
}
                     void h (int x) { g ();  cout << "U::h," << x << endl; }
};
int main( ){ U u; T * v = &u;          v -> f (1);    f (2);
                 v -> g ();  g ();  v -> h ();     h ();
                 v -> h (3);  h (4);   return 0; }
```

3. Для каждого вызова перегруженной функции с одним параметром укажите шаг алгоритма, на котором будет выбрана наиболее подходящая функция или выявлена неоднозначность. Для выбранной функции укажите ее прототип.

```
class X   { int x; public: X (const int n = 0) { x = n;    }
                           operator int ()     { return x; } };
class Y   { X y;   public: Y (const X x)       { y = x;    }
                           operator X ()       { return y; } };
int f (X a = 1) { return a; }  X   f (Y a)     { return a; }
int main () { short int s = 1; bool b = true; float f = 1; X x = s; Y y = x;
      f (s);
      f (x);
      f (b);
      f (f);
      f (y);  }
```

4. Для объектов из задания **1** определить, какие конструкторы и деструкторы и в каком порядке будут выполняться при работе следующего фрагмента программы:

```
int main () { A a; class D { B b; A a; C c; public: D (): a (c) { } }; D
d; }
```

5. В открытой части класса *cl*, предназначенного для работы со сложными объектами, имеются конструктор и функция преобразования. Для облегчения работы с объектами этого класса создаётся дополнительный класс *ObjPtr*, в закрытую часть которого включается поле указателя на класс *cl*.

```
template <class T> class cl { /* ... */
   public: cl (const T s = 0);
           operator T ();
};
template <class T> class ObjPtr { /* ... */ cl <T> * c; public: /* ...
*/ };
```

Для класса *"ObjPtr"* написать реализацию префиксных и постфиксных операций уменьшения *"--"* таким образом, чтобы правильными оказались выражения:

```
ObjPtr <int> t1;
/* ... */ t1 --, -- t1 /* ... */
```

Если для какой-либо из префиксных (постфиксных) функций выбирается реализация методом класса *"ObjPtr"*, другая префиксная (постфиксная) функция должна реализовываться функцией-другом.

6. Какая функция называется в Си++ виртуальной? Для чего используются виртуальные функции? Как называются классы, содержащие виртуальные функции? Привести пример синтаксической записи такого класса. Можно ли создавать объекты и описывать конструкторы таких классов?

7. Что будет выдано в стандартный поток вывода при работе следующей программы?

```
void f(X & x, int n);
struct X { X (){ try  { f(*this, 0);  cout << "a"; }
                 catch (X) { cout << "b"; }
                 catch (int){ cout << "c"; }  }
           X (X &){ cout << "d"; }
  virtual ~X (){ cout << "e"; }  };
struct Y: X { Y () { try { f (*this, -1); cout << "f"; }
                     catch (Y){ cout << "g"; }
                     catch (int){ cout << "h"; }
                     cout << "i"; }
              Y (Y &){ cout << "j"; }
             ~Y (){ cout << "k"; }  };
void f(X & x, int n) { try  { if (n < 0) throw -n;
                              else if (n == 0) throw x;      else throw n; }
                       catch (int){ cout << "l"; }  }
int main() { try { Y a; }
             catch (...){ cout << "m"; return 1; }
             cout << "n";
             return 0;
}
```

8. Если есть ошибки в следующем фрагменте, то в чем они заключаются? Исправьте ошибки, ничего не удаляя, добавив в общей сложности не более 17 символов.

```
class M    { public: int t; int mp (int i) { return t + i; }};
class N    { public: int t; int np (int i) { return t + i; }};
class O: M { public: int t; int op (int i) { return t + i; }};
class P: N, O { public: int p; int pp (int i) { return t + i; }};
void g ()  {      O * po = new N;
                  N * pn = po;
                  M * pm = po;  }
```

9. Перечислить виды полиморфизма, которые можно встретить в программах на языке Си++, дать пояснения. Привести соответствующие примеры фрагментов программы.

10. Написать функцию *g()* с параметром, представляющим собой контейнер-вектор элементов целого типа. Функция просматривает контейнер от конца к началу и меняет текущий элемент с предшествующим в векторе, если текущий меньше предшествующего. Например [1,6,5,3] => [1,3,6,5] : значение 3 поменяется сначала с 5, затем с 6. Далее функция печатает значения элементов контейнера в прямом порядке. Функция возвращает число совершенных обменов значений.

Ответы_Вариант 4_2011

1. Ответ:

```
//          x = c.C::g (& b);
            x = c.B::f ();
            x = c.C::f (x);
            x = c.A::f (x, 1);
//          x = p -> f ();
//          x = p -> f (x);
            x = p -> A::f (x, 1);
```

Ошибочны 5 операторов, из которых только 2 можно исправить (подчеркнуты).

Критерии: За пропуск ошибки (из 5): -3. За каждое не отмеченное или неверное уточнение области видимости (из 4): -2. За каждую лишнюю обнаруженную "ошибку" -3.

2. Ответ: Ошибок нет.

Будет выдано в поток:

```
U::g      U::h,1    U::f,1      ::h,2     ::f,2     U::h
T::g       ::h,0     ::g       U::h       ::h      U::g
U::h,3     ::h,4
```

Критерии: **За каждую лишнюю ошибку или неправильно вызванную функцию – 2.**

3. Ответ:
```
f (s);      // f (X), (г) пользовательское преобразование
f (x);      // f (X), (а) точное отождествление
f (b);      // f (X), (г) пользовательское преобразование
f (f);      // f (X), (г) пользовательское преобразование
f (y);      // f (Y), (а) точное отождествление
```
Критерии: **За каждую ошибку –2.**

4. Ответ: `A(),B(),A(const A&),A(),B(),C(),D(),~D(),~C(),~B(),~A(),~A(),~B(),~A().`
Критерии: **За каждую ошибку –2.**

5. Ответ:
```
template<class T> class ObjPtr { /* ... */ cl<T> * c; public: /* ... */
   const ObjPtr<T> & operator-- ()    { * c = * c - 1;   return * this;
}
   const ObjPtr<T>   operator-- (int pusto)
      { ObjPtr<T> v; v.c = new cl<T>;   * c = * c - 1;   return v; }

   friend const ObjectPtr<T> & operator--(ObjectPtr<T> & s);
   friend const ObjectPtr<T>   operator--(ObjectPtr<T> & s, int pusto);

template<class T> const ObjectPtr<T>&operator--(ObjectPtr<T> &s)
   {* s.c = * s.c - 1;     return s;    }
template<class T> const ObjectPtr<T> operator--(ObjectPtr<T> &s, int
pusto)
   { ObjectPtr<T> v; v.c = new cl<T>; * v.c = * s.c - 1; return v; }
```

Должны быть представлены функции из каждого набора.

Критерий: **За отсутствие каждого примера (из 2-х) или ошибки в нем: -5. За отсутствие реализаций какого-либо вида при правильной реализации этих операций другим способом: -4.**

6. Ответ: **1.** Виртуальной функцией является функция-член класса, которая описана в этом классе с использованием слова *"virtual"*. Такие функции используются для организации выбора методов классов на этапе выполнения программы по текущему значению указателя (ссылки), используемому для обращения к ним. Классы, содержащие виртуальные функции, называются полиморфными. Создавать объекты и описывать конструкторы полиморфных классов можно. Пример полиморфных классов (пример должен содержать, как минимум, 1 базовый класс и 1 производный):
```
class B           { public: virtual void print (); };
class P: public B { public:         void print (); };
```

Критерий: За неверный ответ на вопрос о виртуальной функции: **-5**. За неверный ответ на вопрос о полиморфном классе: **-4**. За отсутствие примера: **-5**. За неверный ответ на вопрос об объектах и конструкторах: **-4**.

7. Ответ: `d b e l f i k e n`
Критерии: За каждую неверную (лишнюю или пропущенную) печать -2.

8. Ответ: ~~**1.** Необходимо удалить указание наследования классом *"P"* членов класса *"M"*, иначе эти члены наследуются дважды.~~

2. Необходимо сделать открытым наследование классом *"O"* членов класса *"M"*, иначе ошибочным оказывается последний оператор функции *"g()"*:
```
class O: public M    { public: int t; int op (int i) { return t + i; }};
```
3. Необходимо уточнить область видимости поля *"t"* в теле метода *"pp"* класса *"P"*, иначе возникает неоднозначность трактовки (*"N::t"* или *"O::t"*):
```
class P: /*M,*/ N, O { public: int p; int pp (int i) { return N::t + i; }};
```
4. Необходимо явно указывать способ преобразования указателей с помощью операций преобразования типа, например, так: `O * po = (O *)(new N); N * pn = (N *)(po);`
5. Открытости наследования можно добиться, заменяя в классе *"O"* слово *"class"* словом *"struct"*.
6. Других ошибок нет.

Критерий: **За пропуск любой из пяти ошибок или неправильное исправление: -2. За указание ошибки там, где ее нет: -5.**

9. Ответ: 1. Статический полиморфизм (действует только на этапе компиляции и характеризуется использованием одноимённых функций с разными профилями)

Пример:
```
int abs (int j);   long int abs (long int j);  double abs (double x);
int i = -4;        long int l = -200000;       double x = - 18;
int k = abs (i);   long int m = abs (l);       double y = abs (x);
```

2. Динамический полиморфизм (реализуется с помощью виртуальных функций и доступа к функциям по указателям, когда на этапе компиляции не удаётся определить точную вызываемую функцию, так как значение указателя в это время не известно)

Пример:
```
struct TData    { int T; virtual void Print () { /* ... */ } } t;
struct EData: T { int E;         void Print () { /* ... */ } } e;
void    Data (TData & d){ /* ... */ d.Print ()   /* ... */ }
Data (t); Data (e);
```

3. Параметрический или типовый полиморфизм (связан с введением шаблонов как средства параметрического описания классов)

Пример:
```
template<class T> T max (T x, T y) { return x > y ? x : y; }
max (1, 2);        max ('a', 'a');             max<int> ('a', 100);
max (2.5, 1.0);    max<double> (2.5, 1.0);
```

Критерий: За пропуск любого вида полиморфизма: **-4.** Отсутствие любого примера: **-2.**

10. **Ответ:**
```
typedef vector<int> V;
V::size_type g (V & vect)
{ V::size_type count = 0;
  for (int i = vect.size () - 1; i > 0; -- i)
    { V::value_type e_even, e_odd;
      e_even = vect [i];
      -- i;
      e_odd  = vect [i];
      if  (e_even >= e_odd)  continue;
      vect [i + 1] = e_odd;
      vect [i] = e_even;
      ++ count;
    }
  for (int i = 0; i < vect.size (); ++ i) cout << vect [i];
  return count;
}
```

КРИТЕРИИ: За каждую существенную ошибку: -3. Отсутствие ссылки в списке формальных параметров - -3. Если вместо индексации использованы указатели, но программа правильная – не снижать. Использование вместо типа *size_type* типа *int* - не наказывать.

2011 год
Экзамен

Вариант 1_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to AXCAB \mid ACAB \mid \varepsilon$
$X \to AXCA \mid ACA$
$AC \to CA$
$C \to a$
$aA \to aa$
$aB \to ab$

(а) Описать язык $L(G)$ в виде теоретико- множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы). **Ответ**:

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные языки	
контекстно-зависимые языки	
языки типа 0	
регулярные языки	

(в) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.
Ответ:

2. Является ли грамматика $G = \langle \{a,d\}, \{A, B, C, S\}, P, S \rangle$, где

$P = \{ S \to AC \mid AB; \quad A \to AC \mid \varepsilon; \quad B \to Bd \mid dBB; \quad C \to dCB \mid \varepsilon; \}$

(а) приведенной ? (б) однозначной ? Ответ обосновать.

3. По заданной регулярной грамматике G построить конечный автомат в виде ДС. Детерминирован ли автомат? Ответ обосновать. Если автомат недетерминированный, то с помощью соответствующего алгоритма преобразовать его в эквивалентный ДКА.

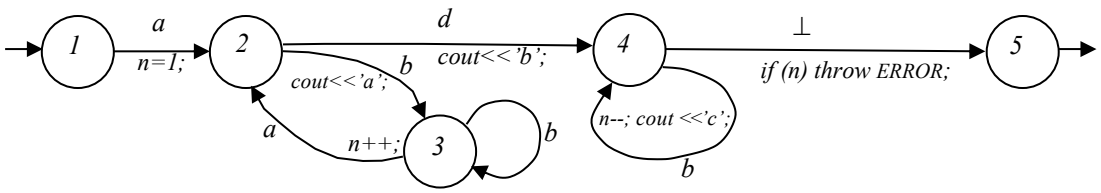
G:
$S \to B\perp \mid A\perp$
$B \to Bb \mid Ab \mid a$
$A \to Ab \mid Ba \mid a$

4. Сформулировать критерий применимости метода рекурсивного спуска. Применим ли этот метод к данной КС-грамматике? Ответ обосновать.

$S \to AC \mid bB$
$A \to aA \mid \varepsilon$
$B \to b$
$C \to Bb \mid \varepsilon$

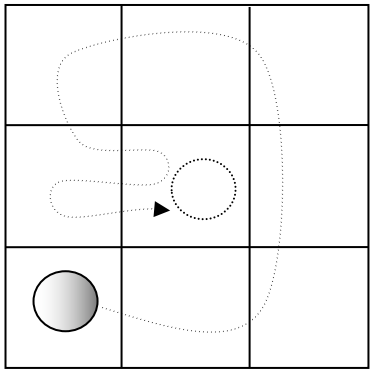
5. Дан автомат $\mathcal{A}$ в виде ДС с действиями. С помощью действий он допускает цепочки языка L_1 и переводит их в цепочки языка L_2. Определить языки L_1 и L_2. Построить КС-грамматику, анализируемую методом рекурсивного спуска, с действиями только вида *cout*<<'символ', задающую тот же перевод цепочек L_1 в цепочки L_2.

$\mathcal{A}$:



6. Привести три примера контекстных условий, выполнение которых контролируется в современных языках программирования. **Ответ:**

7. На клетчатой поверхности размера 3x3 в левой нижней клетке расположен робот-шар. Он умеет выполнять команды перехода в соседнюю клетку: `'a'` – переместиться на одну клетку вправо, `'b'` – влево, `'c'` – вверх, `'d'` – вниз. С помощью последовательности команд можно задавать траекторию движения шара. Например, последовательность `aaccbbdaba` перемещает шар в центральную клетку. Последовательность команд является цепочкой в алфавите {a,b,c,d}. Если цепочка пуста, шар остается на месте. Опишите ***регулярную*** грамматику с терминальным алфавитом {a,b,c,d}, задающую все возможные траектории шара, удовлетворяющие условиям: шар начинает движение из левой нижней клетки, может двигаться только вокруг центральной клетки против часовой стрелки, не покидая поверхности; остановиться может только в левой нижней клетке.
В грамматике должно быть не более 9 правил, считая альтернативы.



Ответ:

8. Является ли верификация частью отладки ? Ответ обосновать.

Ответ:

9. Перечислить этапы создания программного продукта, указав выходные данные каждого из них.

Ответ:

10. Дана польская инверсная запись фрагмента программы, в котором нет операторов перехода `goto` и `break`, но есть один оператор `continue`. Вставить пропущенные в польской записи команды перехода (’!’,’!F’) и недостающие метки; восстановить фрагмент на языке Си.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
x	0	<			y	0	!=			x	#+	;	y	–#	;	x	1	+	0	==		

24	25	26	27	28	29	30	31	32	33	34
1		32		x	y	=	;			

Вариант 1_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

За каждую задачу — максимум 10 баллов

1. Дана грамматика G:

$S \to AXCAB \mid ACAB \mid \varepsilon$
$X \to AXCA \mid ACA$
$AC \to CA$
$C \to a$
$aA \to aa$
$aB \to ab$

(а) Описать язык $L(G)$ в виде теоретико- множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы). **Ответ**:
$L(G) = \{ a^{3n} b \mid n \geq 1\} \cup \{\varepsilon\}$

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные языки	**да (-1)**
контекстно-зависимые языки	**да (-1)**
языки типа 0	**да (-1)**
регулярные языки	**да (-3)**

(в) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.

Ответ: $k = 1$ (есть правило $AC \to CA$, которое не является КС-правилом; правило $S \to \varepsilon$ единственное укорачивающее, и S не встречается в правых частях правил $\Rightarrow$ выполняются условия определения неукорачивающей грамматики)

КРИТЕРИИ: (а) описание отсутствует или с ошибками: -4
(б) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице (обоснование не обязательно)
(в) нет ответа; неверный ответ: -4 (обоснование не обязательно)

2. Является ли грамматика $G = \langle \{a,d\}, \{A, B, C, S\}, P, S \rangle$, где
$P = \{S \to AC \mid AB;\ A \to AC \mid \varepsilon;\ B \to Bd \mid dBB;\ C \to dCB \mid \varepsilon;\ \}$
(а) приведенной ? (б) однозначной ? Ответ обосновать.

Решение

(а) **Нет**. Символы $\{a, d, B\}$ бесполезны, т.е. не участвуют в выводах цепочек языка. Для обоснования достаточно указать хотя бы один бесполезный символ.

(б) **Нет**. Для единственной выводимой в грамматике G цепочки ε существуют бесконечно много левых выводов (и соответственно деревьев выводов). Для обоснования достаточно привести два различных левых (или правых) вывода (или два дерева вывода) цепочки ε :
$S \to AC \to C \to \varepsilon$;
$S \to AC \to ACC \to CC \to C \to \varepsilon$.

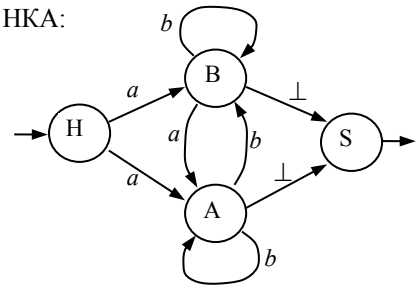
КРИТЕРИИ: ошибочные (отсутствующие) ответ или обоснование на первый вопрос: -5
ошибочные (отсутствующие) ответ или обоснование на второй вопрос: -5

3. По заданной регулярной грамматике G построить конечный автомат в виде ДС. Детерминирован ли автомат? Ответ обосновать. Если автомат недетерминированный, то с помощью соответствующего алгоритма преобразовать его в эквивалентный ДКА.

G:
$S \to B\perp \mid A\perp$
$B \to Bb \mid Ab \mid a$
$A \to Ab \mid Ba \mid a$

Решение

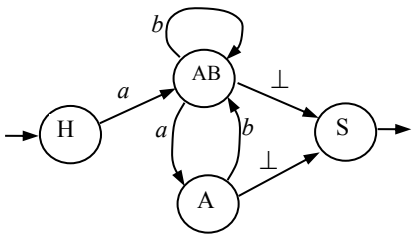
НКА:



Строим таблицу переходов ДКА:

	a	*b*	⊥
H	*AB*	∅	∅
AB	*A*	*AB*	*S*
A	∅	*AB*	*S*
S	∅	∅	∅

По таблице переходов строим диаграмму ДКА:



(Допускается представить ДКА только в виде таблицы переходов (или набора команд δ) или только в виде диаграммы.)

КРИТЕРИИ:

Есть ошибки в НКА : -5

Нет обоснования или ошибочное : -3

Есть ошибки в преобразовании в ДКА: -5

4. Сформулировать критерий применимости метода рекурсивного спуска. Применим ли этот метод к данной КС-грамматике? Ответ обосновать.

$S \to AC \mid bB$

$A \to aA \mid \varepsilon$

$B \to b$

$C \to Bb \mid \varepsilon$

Решение

Критерий применимости метода рекурсивного спуска.

Пусть G — КС-грамматика. Метод рекурсивного спуска применим к G, если и только если для любой пары альтернатив $X \to \alpha \mid \beta$ выполняются следующие условия:

(4) $first(\alpha) \cap first(\beta) = \varnothing$;

(5) справедливо не более чем одно из двух соотношений: $\alpha \Rightarrow \varepsilon$, $\beta \Rightarrow \varepsilon$;

(6) если $\beta \Rightarrow \varepsilon$, то $first(\alpha) \cap follow(X) = \varnothing$.

Для данной грамматики $first(AC) \cap first(bB) = \{b\} \neq \varnothing$. То есть, для пары правил $S \to AC$ и $S \to bB$ не выполняется пункт (1) критерия. Других пар, нарушающих критерий, нет.

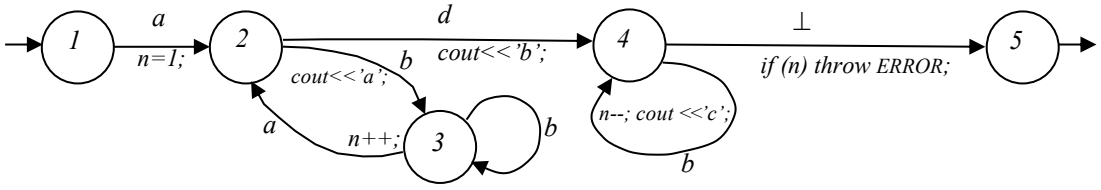
КРИТЕРИИ:

Есть ошибки в формулировке критерия: -5

Есть ошибки в обосновании: -5

5. Дан автомат $\mathcal{A}$ в виде ДС с действиями. С помощью действий он допускает цепочки языка L_1 и переводит их в цепочки языка L_2. Определить языки L_1 и L_2. Построить КС-грамматику, анализируемую методом рекурсивного спуска, с действиями только вида *cout*<<'символ', задающую тот же перевод цепочек L_1 в цепочки L_2.

$\mathcal{A}$:



Решение

Автомат $\mathcal{A}$ с действиями допускает цепочки вида $ay_1y_2\ldots y_{n-1}db^n\perp$, где $n \geq 1$, $y_i = bx_ia$, $x_i \in \{b\}^*$ для $i = 1,\ldots, n-1$, и каждую такую цепочку переводит соответственно в цепочку $a^{n-1}bc^n$. Грамматика с действиями такова (запись вида $\langle a \rangle$ означает $\langle$*cout*<<'*a*'$\rangle$):

$S \to aAb\langle c\rangle\perp$

$A \to bBa\langle a\rangle Ab\langle c\rangle \mid d\,\langle b\rangle$ (или $A \to b\langle a\rangle BaAb\langle c\rangle \mid d\,\langle b\rangle$)

$B \to bB \mid \varepsilon$

Возможен и такой вариант ответа:

$S \to aAb\langle c\rangle\perp$

$A \to bBaAb\langle c\rangle \mid d\,\langle b\rangle$

$B \to bB \mid \varepsilon\ \langle a\rangle$

Еще вариант, основанный на рассмотрении цепочек L_1 как $ay_1y_2\ldots y_ndbb^n\perp$, где $n \geq 0$, $y_i = bx_ia$, $x_i \in \{b\}^*$ для $i = 1,\ldots, n$,

$S \to aA\perp$

$A \to bBaAb\langle c\rangle \mid db\,\langle b\rangle\,\langle c\rangle$

$B \to bB \mid \varepsilon\ \langle a\rangle$

КРИТЕРИИ: -- есть ошибки в описаниях языков L_1 , L_2 : -3
-- грамматика для L_1 ошибочна: -4
-- метод рекурсивного спуска неприменим: -2
-- есть действие не вида *cout*<<'символ' : -10
-- конечное число цепочек переводятся неправильно: -3
-- бесконечно много цепочек переводятся неправильно: -6

6. Привести три примера контекстных условий, выполнение которых контролируется в современных языках программирования.

Ответ: (какие-нибудь три подходящих примера:)

1) Имя, используемое в программе, должно быть описано.
2) Не допускаются два описания одного и того же имени в одной области видимости.
3) В операторе присваивания типы переменной и выражения должны быть совместимыми (совпадать).
4) В условном операторе и в операторе цикла в качестве условия возможно только логическое выражение (Паскаль, Оберон, Ада).
5) Операнды операции отношения должны быть определенного типа (например, одного из перечислимых типов в Паскале).
6) Количество фактических параметров в вызове подпрограммы должно соответствовать количеству формальных параметров в описании подпрограммы, а типы фактических параметров должны быть совместимы по присваиванию с типами соответствующих формальных параметров.

Возможны другие условия.

КРИТЕРИИ: за каждый пропущенный (или ошибочный) пункт: -4.

Вариант 2_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to AUCAD$
$AUCA \to AAUCACA \mid \varepsilon$
$AC \to CA$
$C \to c$
$cD \to cb$
$cA \to cc$

(а) Описать язык $L(G)$ в виде теоретико- множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы). **Ответ**:

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\Im$	$L(G) \in \Im$? (да/нет)
контекстно-свободные языки	
контекстно-зависимые языки	
языки типа 0	
регулярные языки	

(в) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.
Ответ:

2. Является ли грамматика $G = \langle \{a,b\}, \{A, B, D, S\}, P, S \rangle$, где
$P = \{ S \to AB \mid Aa \mid aD; \quad A \to aAB \mid a; \quad B \to Bb \mid BbB; \quad D \to BD \mid a; \}$
(а) приведенной ? (б) однозначной ? Ответ обосновать.

3. По заданной регулярной грамматике G построить конечный автомат в виде ДС. Детерминирован ли автомат? Ответ обосновать. Если автомат недетерминированный, то с помощью соответствующего алгоритма преобразовать его в эквивалентный ДКА.

G:
$S \to C\perp \mid A\perp$
$C \to Cd \mid Ad \mid a$
$A \to Ad \mid Ca \mid a$

4. Сформулировать критерий применимости метода рекурсивного спуска. Применим ли этот метод к данной КС-грамматике? Ответ обосновать.

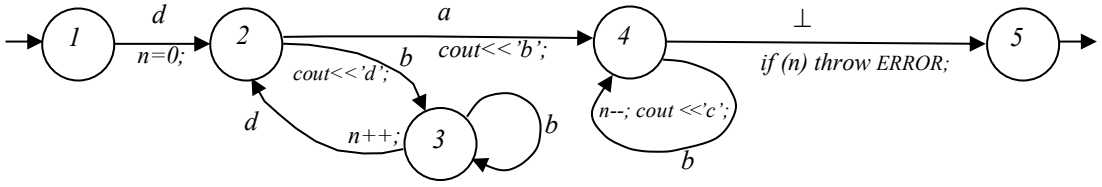
$S \to AdS \mid B \mid \varepsilon$

$A \to aB \mid cAb$

$B \to bB \mid \varepsilon$

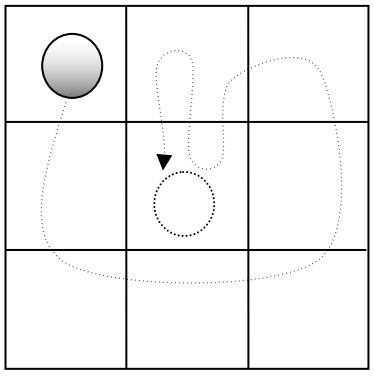
5. Дан автомат $\mathcal{A}$ в виде ДС с действиями. С помощью действий он допускает цепочки языка L_1 и переводит их в цепочки языка L_2. Определить языки L_1 и L_2. Построить КС-грамматику, анализируемую методом рекурсивного спуска, с действиями только вида *cout*<<'символ', задающую тот же перевод цепочек L_1 в цепочки L_2.

$\mathcal{A}$:



6. Привести пример фрагмента программы (на языке Паскаль, Си или Си++), содержащей по одной ошибке каждого вида: лексическую, синтаксическую, семантическую. Пояснить каждую ошибку. **Ответ:**

7. На клетчатой поверхности размера 3x3 в левой верхней клетке расположен робот-шар. Он умеет выполнять команды перехода в соседнюю клетку: `'a'` – переместиться на одну клетку вправо, `'b'` – влево, `'c'` – вверх, `'d'` – вниз. С помощью последовательности команд можно задавать траекторию движения шара. Например, последовательность `ddaaccbdcd` перемещает шар в центральную клетку. Последовательность команд является цепочкой в алфавите {a,b,c,d}. Если цепочка пуста, шар остается на месте. Опишите ***регулярную*** грамматику с терминальным алфавитом {a,b,c,d}, задающую все возможные траектории шара, удовлетворяющие условиям: шар начинает движение из левой верхней клетки, может двигаться только вокруг центральной клетки против часовой стрелки, не покидая поверхности; остановиться может только в левой верхней клетке.



В грамматике должно быть не более 9 правил, считая альтернативы. **Ответ:**

8. Является ли верификация видом тестирования? Ответ обосновать.

Ответ:

9. Перечислить методы оптимизации циклов. Привести один пример оптимизации.

Ответ:

10. Дана польская инверсная запись фрагмента программы, в котором нет операторов перехода `goto` и `break`, но есть один оператор `continue`. Вставить пропущенные в польской записи команды перехода ('!','!F') и недостающие метки; восстановить фрагмент на языке Си.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
z	0	!=			i	1	=	;	i	N	<	40				i	#+	;			x	i

24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
*	y	>			x	2	–=	;	17		z	–#	;			1		

Вариант 2_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

За каждую задачу — максимум 10 баллов

1. Дана грамматика G:

$S \to AUCAD$
$AUCA \to AAUCACA \mid \varepsilon$
$AC \to CA$
$C \to c$
$cD \to cb$
$cA \to cc$

(а) Описать язык $L(G)$ в виде теоретико- множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы). **Ответ**: $L(G) = \{ c^{3n} b \mid n \geq 1 \}$

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные языки	**да (-1)**
контекстно-зависимые языки	**да (-1)**
языки типа 0	**да (-1)**
регулярные языки	**да (-3)**

(в) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.
Ответ: $k = 0$ (есть укорачивающее правило $AUCA \to \varepsilon$, которое не удовлетворяет условиям для грамматики типа 1 (неукорачивающей).

КРИТЕРИИ: (а) описание отсутствует или с ошибками: -4
(б) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице (обоснование не обязательно)
(в) нет ответа; неверный ответ: -4 (обоснование не обязательно)

2. Является ли грамматика $G = \langle \{a,b\}, \{A, B, D, S\}, P, S \rangle$, где
$P = \{S \to AB \mid Aa \mid aD;\ A \to aAB \mid a;\ B \to Bb \mid BbB;\ D \to BD \mid a;\ \}$

(а) приведенной ? (б) однозначной ? Ответ обосновать.

Решение

(а) **Нет**. Символы $\{b,\ B\}$ бесполезны, т.е. не участвуют в выводах цепочек языка. Для обоснования достаточно указать хотя бы один бесполезный символ.)

(б) **Нет**. Для единственной выводимой в грамматике G цепочки aa существуют два различных левых (правых) вывода (и соответственно, два различных дерева вывода) :
$S \to aD \to aa$;
$S \to Aa \to aa$.

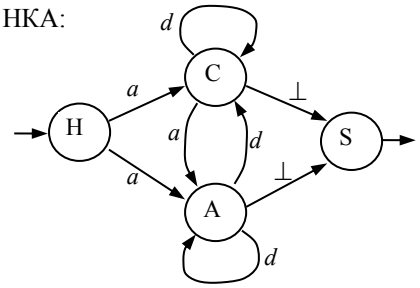
КРИТЕРИИ: ошибочные ответ или обоснование на первый вопрос: -5
ошибочные ответ или обоснование на второй вопрос: -5

3. По заданной регулярной грамматике G построить конечный автомат в виде ДС. Детерминирован ли автомат? Ответ обосновать. Если автомат недетерминированный, то с помощью соответствующего алгоритма преобразовать его в эквивалентный ДКА.

G:
$S \to C\perp \mid A\perp$
$C \to Cd \mid Ad \mid a$
$A \to Ad \mid Ca \mid a$

Решение

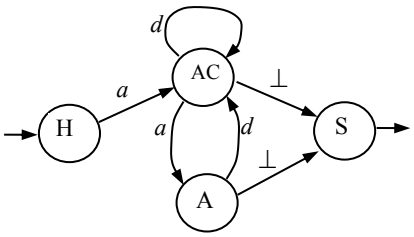
НКА:



Строим таблицу переходов ДКА:

	a	*d*	⊥
H	*AC*	∅	∅
AC	*A*	*AC*	*S*
A	∅	*AC*	*S*
S	∅	∅	∅

По таблице переходов строим диаграмму ДКА:



(Допускается представить ДКА только в виде таблицы переходов (или набора команд δ) или только в виде диаграммы.)

КРИТЕРИИ:

Есть ошибки в НКА : -5

Нет обоснования или ошибочное : -3

Есть ошибки в преобразовании в ДКА: -5

4. Сформулировать критерий применимости метода рекурсивного спуска. Применим ли этот метод к данной КС-грамматике? Ответ обосновать.

$S \to AdS \mid B \mid \varepsilon$

$A \to aB \mid cAb$

$B \to bB \mid \varepsilon$

Решение

Критерий применимости метода рекурсивного спуска.

Пусть G — КС-грамматика. Метод рекурсивного спуска применим к G, если и только если для любой пары альтернатив $X \to \alpha \mid \beta$ выполняются следующие условия:

(7) $first(\alpha) \cap first(\beta) = \varnothing$;

(8) справедливо не более чем одно из двух соотношений: $\alpha \Rightarrow \varepsilon$, $\beta \Rightarrow \varepsilon$;

(9) если $\beta \Rightarrow \varepsilon$, то $first(\alpha) \cap follow(X) = \varnothing$.

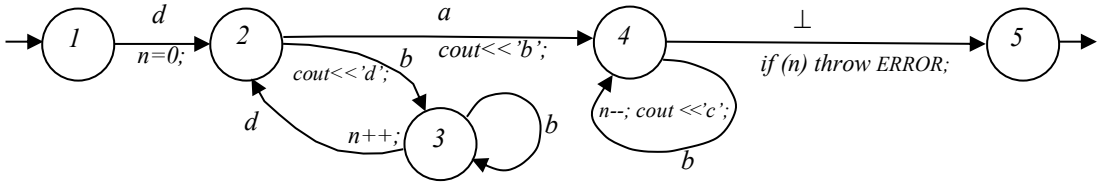
В данной грамматике справедливы соотношения $\varepsilon \Rightarrow \varepsilon$ и $B \Rightarrow \varepsilon$, то есть для пары правил $S \to \varepsilon$ и $S \to B$ не выполняется пункт (2) критерия. Других пар, нарушающих критерий, нет.

КРИТЕРИИ:

Есть ошибки в формулировке критерия: -5

Есть ошибки в обосновании: -5

5. Дан автомат $\mathcal{A}$ в виде ДС с действиями. С помощью действий он допускает цепочки языка L_1 и переводит их в цепочки языка L_2. Определить языки L_1 и L_2. Построить КС-грамматику, анализируемую методом рекурсивного спуска, с действиями только вида *cout*<<'символ', задающую тот же перевод цепочек L_1 в цепочки L_2.
$\mathcal{A}$:



Решение

Автомат $\mathcal{A}$ с действиями допускает цепочки вида $dbx_1dbx_2\ldots dx_ndab^n\perp$, где $n\geq 0$, $x_i\in\{b\}^*$ для $i=1,\ldots,n$, $dy_1y_2\ldots y_nab^n\perp$, где $n\geq 0$, $y_i = bx_id$, $x_i\in\{b\}^*$ для $i=1,\ldots,n$, и каждую такую цепочку переводит соответственно в цепочку d^nbc^n.
Грамматика с действиями такова (запись вида $\langle a\rangle$ означает $\langle$*cout*<<'*a*'$\rangle$):
$S \to dA\perp$
$A \to bBd\langle d\rangle Ab\langle c\rangle \,|a\,\langle b\rangle$ (или $A \to b\langle d\rangle BdAb\langle c\rangle \,|a\,\langle b\rangle$)
$B \to bB \mid \varepsilon$

Возможен и такой вариант ответа:
$S \to dA\perp$
$A \to bBdAb\langle c\rangle \,|a\,\langle b\rangle$
$B \to bB \mid \varepsilon\ \langle d\rangle$

КРИТЕРИИ: -- есть ошибки в описаниях языков L_1 , L_2 : -3
-- грамматика для L_1 ошибочна: -4
-- метод рекурсивного спуска неприменим: -2
-- есть действие не вида *cout*<<'символ' : -10
-- конечное число цепочек переводятся неправильно: -3
-- бесконечно много цепочек переводятся неправильно: -6

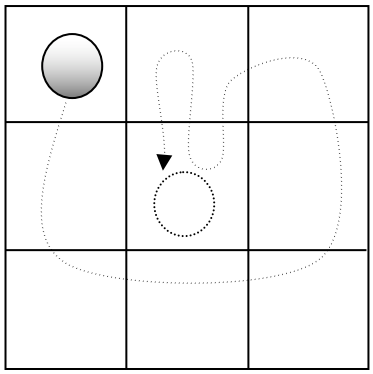
6. Привести пример фрагмента программы (на языке Паскаль, Си или Си++), содержащей по одной ошибке каждого вида: лексическую, синтаксическую, семантическую. Пояснить каждую ошибку.

Ответ:

```
program Errors;
var i: integer;
    t: bool;
begin  read(i); t:= (i< >1) and (i#2);  {лексическая: не существует лексемы #}
   begin
     t:=1 {семантическая: тип выражения в операторе присваивания
           не соответствует типу переменной }
end. {синтаксическая: не хватает парного end для  begin}
```

КРИТЕРИИ: в качестве лексической предложена синтаксическая или семантическая ошибка, например, в описании a: intger, intger является законным именем (лексемой), но не описанным : -2
за остальные ошибки в примерах и пояснениях: -4

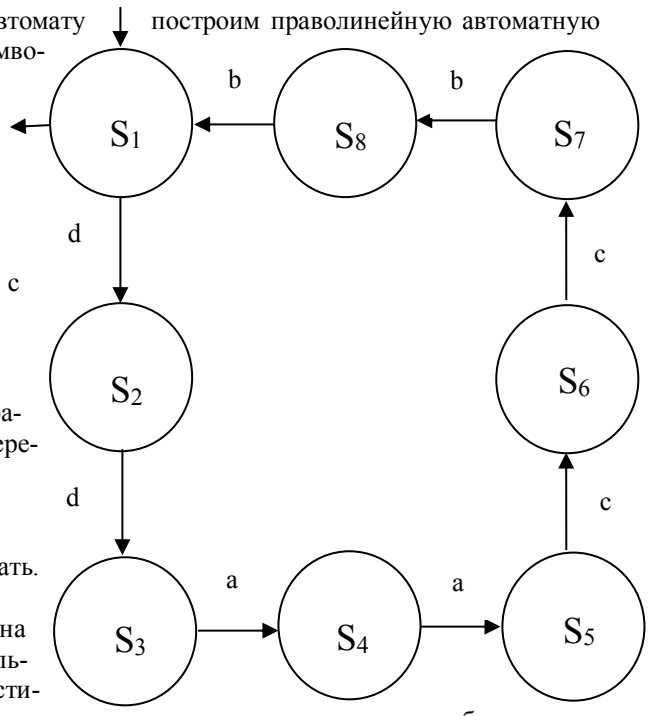
7. На клетчатой поверхности размера 3x3 в левой верхней клетке расположен робот-шар. Он умеет выполнять команды перехода в соседнюю клетку: `'a'` – переместиться на одну клетку вправо, `'b'` – влево, `'c'` – вверх, `'d'` – вниз. С помощью последовательности команд можно задавать траекторию движения шара. Например, последовательность `ddaaccbdcd` перемещает шар в центральную клетку. Последовательность команд является цепочкой в алфавите {a,b,c,d}. Если цепочка пуста, шар остается на месте. Опишите ***регулярную*** грамматику с терминальным алфавитом {a,b,c,d}, задающую все возможные траектории шара, удовлетворяющие условиям: шар начинает движение из левой верхней клетки, может двигаться только вокруг центральной клетки против часовой стрелки, не покидая поверхности; остановиться может только в левой верхней клетке.
В грамматике должно быть не более 9 правил, считая альтернативы.



Решение

1	8	7
2		6
3	4	5

Согласно условиям, шар, начав с левой верхней клетки, дальше может двигаться вокруг центральной клетки против часовой стрелки и может остановиться в левой верхней клетке. Перенумеруем против часовой стрелки все клетки (кроме центральной), начиная с левой верхней. Построим конечный автомат с состояниями S_1,S_2, …, S_8. Состояние S_i соответствует положению шара в *i-й* клетке. Дуга из S_i в S_{i+1} помечается подходящей буквой из алфавита {a,b,c,d}. Например, переход из S_1 в S_2 осуществляется по команде `'d'`, и соответствующая дуга помечается символом `'d'`. Начальным является состояние S_1. Оно же является и заключительным состоянием. По конечному автомату построим праволинейную автоматную грамматику с начальным символом S_1:



$S_1 \rightarrow$ dS_2 | ε

$S_2 \rightarrow$ dS_3

$S_3 \rightarrow$ aS_4

...

$S_8 \rightarrow$ bS_1

Можно построить эквивалентную регулярную грамматику с начальным символом S, состоящую всего из двух правил:

$S \rightarrow$ ddaaccbbS | ε

КРИТЕРИИ: не допускается ε: -2, за недопуск любой другой правильной цепочки или за допуск каждой лишней цепочки: -5. Нерегулярная грамматика: -10

8. Является ли верификация видом тестирования ? Ответ обосновать.

Ответ: **Нет**. Верификация – это процесс проверки программы на правильность. Кроме логических методов доказательства правильности программы существуют эвристические, основанные на тестировании. Тестирование — это процесс обнаружения дефектов (несоответствия программы исходным требованиям и спецификациям) путем проверки программы на данных с заранее известными ответами (тестах). То есть верификация более общее понятие, и тестирование является видом верификации.

КРИТЕРИИ:

Нет обоснования, но ответ верно угадан: 1 балл

За отсутствие или ошибочность определения верификации: -5

За отсутствие или ошибочность определения тестирования: -5

9. Перечислить методы оптимизации циклов. Привести один пример оптимизации.

Ответ: (примеры даются на каждый метод, для ответа достаточно одного примера)

1) вынесение инвариантных вычислений из тела цикла:

```
for (i=0; i < 10; i++) A[i] = B*C*A[i];
```

⇒ `D=B*C;  for (i=0; i < 10; i++)  A[i] = D*A[i];`

2) замена операций с переменными цикла:
(а) `S = 10; for (i=0; i < N; i++) A[i] = i * S;`
i – индуктивная переменная

⇒ `S = 10; T = 0;`
`for (i=0; i < 10; i++) T = T + S, A[i] = T;`

(б) элиминация индуктивной переменной:
```
S = 10;
for (i=0; i < N; i ++) Q = Q + F (S), S = S + 10;
```

⇒ `S = 10; M = S + N * 10; while (S <= M) {Q = Q + F(S) ; S = S + 10}`

3) слияние, расщепление и развертывание циклов.
(а)
```
for (i=0; i < N; i++) { S1 }
 for (i=0; i < N; i++) { S2 }
```

⇒ `for (i=0; i < N; i++) { S1; S2 }`

(б) `for (i=0; i < n; i++) if (x < y) { S1; } else { S2; }`

⇒ `if (x < y) for (i=0; i < n; i++) { S1; } else for (i=0; i < n; i++) { S2; }`

(в) `for (i = 0; i < n; i ++) { A[i]=B[i]*C[i]; }`

⇒ `for (i = 0; i < n; i +=2) { A[i]=B[i]*C[i]; A[i+1]=B[i+1]*C[i+1]; }`

вложенные циклы по обработке массивов могут заменяться
на одиночный цикл с соответствующим пересчетом
индексов.

КРИТЕРИИ:
За ошибочность/отсутствие каждого из трех пунктов: -3
За отсутствие примера или за ошибку в нем (достаточно одного примера): -2

10. Дана польская инверсная запись фрагмента программы, в котором нет операторов перехода `goto` и `break`, но есть один оператор `continue`. Вставить пропущенные в польской записи команды перехода (’!’,’!F’) и недостающие метки; восстановить фрагмент на языке Си. (Красным выделены пропущенные клетки).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
z	0	!=	42	!F	i	1	=	;	i	N	<	40	!F	22	!	i	#+	;	10	!	x	i

24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
*	y	>	35	!F	x	2	–=	;	17	!	z	–#	;	17	!	1	!	}

Ответ:
```
while (z!=0)
        for (i=1; i<N; i++){
            if (x*i>y)  {x-=2;  continue; }
            --z;
        }
```

План решения:
1) Расставляем знаки операций ’!’ и ’!F’, исходя из следующих соображений:
если перед пропущенным местом стоит логическое выражение и нет операции ’;’, то переход условный, иначе — безусловный.
2) Расставляем метки, учитывая, что при трансляции управляющих конструкций языка Си:
(а) переходы по лжи возможны только вперед;

(б) безусловные переходы назад (кроме явных goto) возможны только в циклах.

КРИТЕРИИ: Есть ошибки в ПОЛИЗЕ: -5
Есть ошибки в восстановленном фрагменте: -5
Если без ошибок только операции перехода расставлены, даем 2 балла за задачу.

Вариант ответа

```
while (z!=0){
    for (i=1; i<N; i++){
        if (x*i>y)  x-=2;
              else  --z;
        } continue;
}
```

формально не является правильным, но семантически эквивалентен. За такое решение снимаем 1 балл.

Вариант 3_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to Ybba \mid \varepsilon$
$Y \to bZ \mid b$
$Z \to Ybb$
$Z \to Z$
$Y \to Y$

(а) Описать язык $L(G)$ в виде теоретико- множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).
Ответ:

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные языки	
контекстно-зависимые языки	
языки типа 0	
регулярные языки	

(в) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.
Ответ:

2. Является ли грамматика $G = \langle \{a,b\}, \{A, B, C, S\}, P, S \rangle$, где
$P = \{ S \to AbB \mid Aa; \quad A \to aAB \mid a; \quad B \to Bb \mid BB; \quad C \to bC \mid b; \}$
(а) приведенной ? (б) однозначной ? Ответ обосновать.

3. По заданной регулярной грамматике G построить конечный автомат в виде ДС. Детерминирован ли автомат? Ответ обосновать. Если автомат недетерминированный, то с помощью соответствующего алгоритма преобразовать его в эквивалентный ДКА.

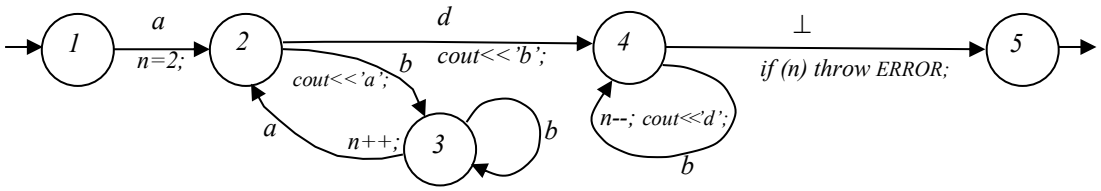
G:
$S \to C\perp \mid B\perp$
$C \to Cd \mid Bd \mid c$
$B \to Bd \mid Cc \mid c$

4. Сформулировать критерий применимости метода рекурсивного спуска. Применим ли этот метод к данной КС-грамматике? Ответ обосновать.

$S \to A \mid cB$
$A \to aB \mid bBa \mid \varepsilon$
$B \to aA \mid b$

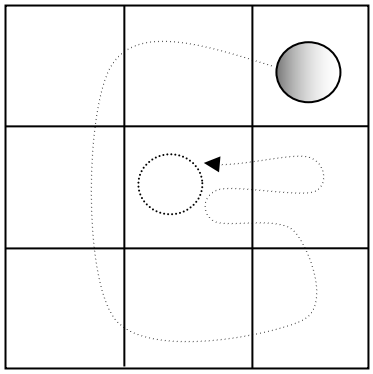
5. Дан автомат $\mathcal{A}$ в виде ДС с действиями. С помощью действий он допускает цепочки языка L_1 и переводит их в цепочки языка L_2. Определить языки L_1 и L_2. Построить КС-грамматику, анализируемую методом рекурсивного спуска, с действиями только вида *cout*<<'символ', задающую тот же перевод цепочек L_1 в цепочки L_2.

$\mathcal{A}$:



6. Какие задачи решаются на этапе лексического анализа? **Ответ:**

7. На клетчатой поверхности размера 3x3 в правой верхней клетке расположен робот-шар. Он умеет выполнять команды перехода в соседнюю клетку: `'a'` – переместиться на одну клетку вправо, `'b'` – влево, `'c'` – вверх, `'d'` – вниз. С помощью последовательности команд можно задавать траекторию движения шара. Например, последовательность `bbddaacbab` перемещает шар в центральную клетку. Последовательность команд является цепочкой в алфавите {a,b,c,d}. Если цепочка пуста, шар остается на месте. Опишите ***регулярную*** грамматику с терминальным алфавитом {a,b,c,d}, задающую все возможные траектории шара, удовлетворяющие условиям: шар начинает движение из правой верхней клетки, может двигаться только вокруг центральной клетки против часовой стрелки, не покидая поверхности; остановиться может только в правой верхней клетке. В грамматике должно быть не более 9 правил, считая альтернативы.



Ответ:

8. Является ли отладка частью тестирования? Ответ обосновать.

Ответ:

9. Перечислить аспекты машинно-зависимой оптимизации.

Ответ:

10. Дана польская инверсная запись фрагмента программы, в котором нет операторов перехода `goto` и `continue`, но есть один оператор `break`. Вставить пропущенные в польской записи команды перехода ('!','!F') и недостающие метки; восстановить фрагмент на языке Си.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
i	1	=	;	i	N	<			17		i	#+	;			z	0	!=			i	2

24	25	26	27	28	29	30	31	32	33	34	35	36	37
%	1	!=			36	!	z	–#	;			12	

Вариант 3_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

За каждую задачу — максимум 10 баллов

1. Дана грамматика G:

$S \to Ybba \mid \varepsilon$
$Y \to bZ \mid b$
$Z \to Ybb$
$Z \to Z$
$Y \to Y$

(а) Описать язык $L(G)$ в виде теоретико- множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).
Ответ: $L(G) = \{ b^{3n} a \mid n \geq 1 \} \cup \{\varepsilon\}$

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\Im$	$L(G) \in \Im$? (да/нет)
контекстно-свободные языки	**да (-1)**
контекстно-зависимые языки	**да (-1)**
языки типа 0	**да (-1)**
регулярные языки	**да (-3)**

(в) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.

Ответ: $k = 2$ (есть и леволинейное, и праволинейное правила: $Z \to Ybb$ *и* $Y \to bZ$. Все правила удовлетворяют ограничениям грамматики типа 2.)

КРИТЕРИИ: (а) описание отсутствует или с ошибками: -4
(б) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице (обоснование не обязательно)
(в) нет ответа; неверный ответ: -4 (обоснование не обязательно)

2. Является ли грамматика $G = \langle \{a,b\}, \{A, B, C, S\}, P, S \rangle$, где
$P = \{S \to AbB \mid Aa;\ A \to aAB \mid a;\ B \to Bb \mid BB;\ C \to bC \mid b;\ \}$

(а) приведенной ? (б) однозначной ? Ответ обосновать.

Решение

(а) **Нет**. Символы $\{b, B, C\}$ бесполезны, т.е. не участвуют в выводах цепочек языка. Для обоснования достаточно указать хотя бы один бесполезный символ.

(б) **Да**. Для единственной выводимой в грамматике G цепочки aa существует только один вывод (и, следовательно, единственное дерево вывода):
$S \to Aa \to aa$.

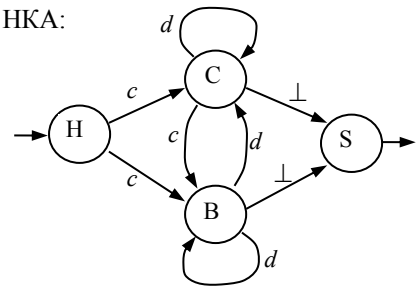
КРИТЕРИИ: ошибочные ответ или обоснование на первый вопрос: -5
ошибочные ответ или обоснование на второй вопрос: -5

3. По заданной регулярной грамматике G построить конечный автомат в виде ДС. Детерминирован ли автомат? Ответ обосновать. Если автомат недетерминированный, то с помощью соответствующего алгоритма преобразовать его в эквивалентный ДКА.

G:
$S \to C\perp \mid B\perp$
$C \to Cd \mid Bd \mid c$
$B \to Bd \mid Cc \mid c$

Решение

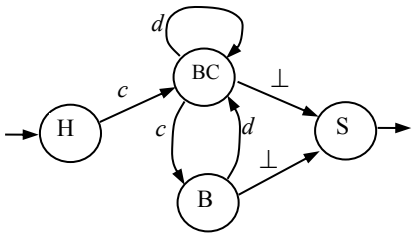
НКА:



Строим таблицу переходов ДКА:

	c	*d*	⊥
H	*BC*	∅	∅
BC	*B*	*BC*	*S*
B	∅	*BC*	*S*
S	∅	∅	∅

По таблице переходов строим диаграмму ДКА:



(Допускается представить ДКА только в виде таблицы переходов (или набора команд δ) или только в виде диаграммы.)

КРИТЕРИИ:
Есть ошибки в НКА : -5

Нет обоснования или ошибочное : -3
Есть ошибки в преобразовании в ДКА: -5

4. Сформулировать критерий применимости метода рекурсивного спуска. Применим ли этот метод к данной КС-грамматике? Ответ обосновать.

$S \to A \mid cB$

$A \to aB \mid bBa \mid \varepsilon$

$B \to aA \mid b$

Решение

Критерий применимости метода рекурсивного спуска.

Пусть G — КС-грамматика. Метод рекурсивного спуска применим к G, если и только если для любой пары альтернатив $X \to \alpha \mid \beta$ выполняются следующие условия:

(10) $first(\alpha) \cap first(\beta) = \varnothing$;

(11) справедливо не более чем одно из двух соотношений: $\alpha \Rightarrow \varepsilon$, $\beta \Rightarrow \varepsilon$;

(12) если $\beta \Rightarrow \varepsilon$, то $first(\alpha) \cap follow(X) = \varnothing$.

В данной грамматике справедливы соотношения $first(aB) \cap follow(A) = \{a\} \neq \varnothing$,

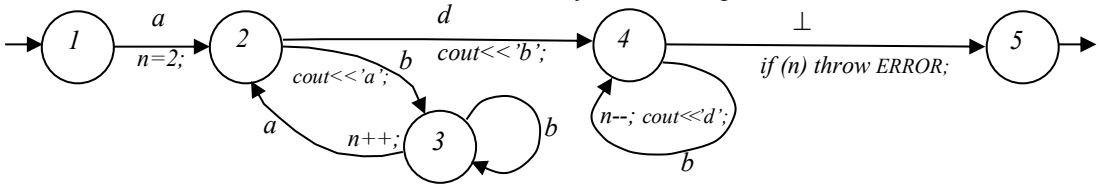
то есть для пары правил $A \to aB$ и $A \to \varepsilon$ не выполняется пункт (3) критерия. Других пар, нарушающих критерий, нет.

КРИТЕРИИ:

Есть ошибки в формулировке критерия: -5
Есть ошибки в обосновании: -5

5. Дан автомат $\mathcal{A}$ в виде ДС с действиями. С помощью действий он допускает цепочки языка L_1 и переводит их в цепочки языка L_2. Определить языки L_1 и L_2. Построить КС-грамматику, анализируемую методом рекурсивного спуска, с действиями только вида *cout*<<'символ', задающую тот же перевод цепочек L_1 в цепочки L_2.

$\mathcal{A}$:



Решение

Автомат $\mathcal{A}$ с действиями допускает цепочки вида $ay_1y_2\ldots y_{n-2}db^n\perp$, где $n \geq 2$, $y_i = bx_ia$, $x_i \in \{b\}^*$ для $i = 1,\ldots, n-2$, и каждую такую цепочку переводит соответственно в цепочку $a^{n-2}bd^n$.

Грамматика с действиями такова (запись вида $\langle a \rangle$ означает $\langle cout<<'a' \rangle$):

$S \to aAb\langle d\rangle b\langle d\rangle\perp$

$A \to bBa\langle a\rangle Ab\langle d\rangle \mid d\,\langle b\rangle$ (или $A \to b\langle a\rangle BaAb\langle d\rangle \mid d\,\langle b\rangle$)

$B \to bB \mid \varepsilon$

Возможен и такой вариант ответа:

$S \to aAb\langle d\rangle b\langle d\rangle\perp$

$A \to bBaAb\langle d\rangle \mid d\,\langle b\rangle$

$B \to bB \mid \varepsilon\ \langle a\rangle$

Еще вариант ответа, основанный на трактовке цепочек L_1 как $ay_1y_2\ldots y_ndbbb^n$, $n \geq 0$, $y_i = bx_ia$, $x_i \in \{b\}^*$ для $i = 1,\ldots, n$.

$S \to aA\perp$

$A \to bBaAb\langle d\rangle \mid d\,\langle b\rangle\langle d\rangle\langle d\rangle$

$B \to bB \mid \varepsilon\ \langle a\rangle$

КРИТЕРИИ: -- есть ошибки в описаниях языков L_1 , L_2 : -3
-- грамматика для L_1 ошибочна: -4
-- метод рекурсивного спуска неприменим: -2
-- есть действие не вида *cout*<<'символ' : -10

-- конечное число цепочек переводятся неправильно: -3
-- бесконечно много цепочек переводятся неправильно: -6

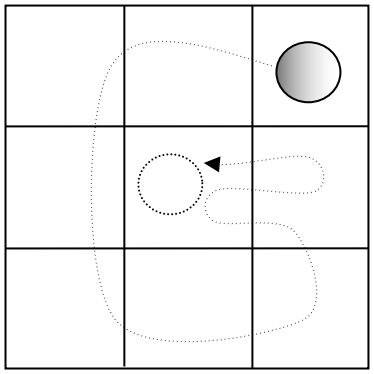
6. Какие задачи решаются на этапе лексического анализа?

Ответ:

1) Выделение в тексте цепочек символов, представляющих лексемы, и проверка правильности их записи.
2) Фиксация в специальных таблицах факта появления лексемы в тексте.
3) Преобразование цепочки, представляющей лексему, в пару ⟨тип_лексемы, указатель_на_информацию_о_ней⟩.
4) Удаление пробельных символов и комментариев.

КРИТЕРИИ: за каждый неверный/не указанный пункт: -3

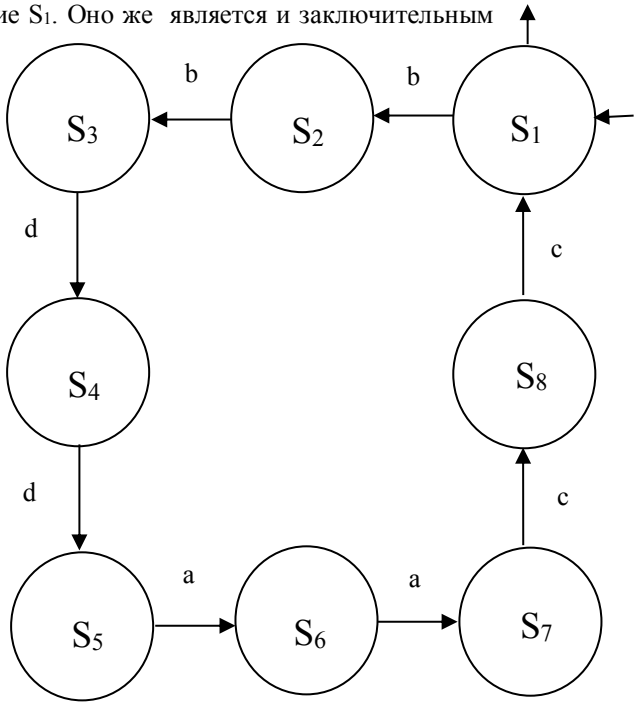
7. На клетчатой поверхности размера 3x3 в правой верхней клетке расположен робот-шар. Он умеет выполнять команды перехода в соседнюю клетку: `'a'` – переместиться на одну клетку вправо, `'b'` – влево, `'c'` – вверх, `'d'` – вниз. С помощью последовательности команд можно задавать траекторию движения шара. Например, последовательность `bbddaacbab` перемещает шар в центральную клетку. Последовательность команд является цепочкой в алфавите {a,b,c,d}. Если цепочка пуста, шар остается на месте. Опишите ***регулярную*** грамматику с терминальным алфавитом {a,b,c,d}, задающую все возможные траектории шара, удовлетворяющие условиям: шар начинает движение из правой верхней клетки, может двигаться только вокруг центральной клетки против часовой стрелки, не покидая поверхности; остановиться может только в правой верхней клетке. В грамматике должно быть не более 9 правил, считая альтернативы.



Решение

3	2	1
4		8
5	6	7

Согласно условиям, шар, начав с правой верхней клетки, дальше может двигаться вокруг центральной клетки против часовой стрелки и может остановиться в правой верхней клетке. Перенумеруем против часовой стрелки все клетки (кроме центральной), начиная с правой верхней. Построим конечный автомат с состояниями S_1,S_2, …, S_8. Состояние S_i соответствует положению шара в *i-й* клетке. Дуга из S_i в S_{i+1} помечается подходящей буквой из алфавита {a,b,c,d}. Например, переход из S_1 в S_2 осуществляется по команде `'b'`, и соответствующая дуга помечается символом `'b'`. Начальным является состояние S_1. Оно же является и заключительным состоянием. По конечному автомату построим праволинейную автоматную грамматику с начальным символом S_1:



$S_1 \rightarrow$ bS$_2$ | ε
$S_2 \rightarrow$ bS$_3$
$S_3 \rightarrow$ dS$_4$
…
$S_8 \rightarrow$ cS$_1$

Можно построить эквивалентную регулярную грамматику с начальным символом S, состоящую всего из двух правил:
S→ `bbddaacc`S | ε

КРИТЕРИИ: не допускается ε: -2, за недопуск любой другой правильной цепочки или за допуск каждой лишней цепочки: -5. Нерегулярная грамматика: -10

8. Является ли отладка частью тестирования? Ответ обосновать.

Ответ: **Нет**. Тестирование — это процесс обнаружения дефектов (несоответствия программы исходным требованиям и спецификациям) путем проверки программы на данных с заранее известными ответами (тестах), а отладка — это поиск причин и устранение уже обнаруженных дефектов.
КРИТЕРИИ:
Нет обоснования, но ответ верно угадан: 1 балл
За отсутствие или ошибочность определения тестирования: -5
За отсутствие или ошибочность определения отладки: -5

9. Перечислить аспекты машинно-зависимой оптимизации.

Ответ:
2) учет регистровой структуры вычислительной аппаратуры (оптимальное распределение регистров, передача параметров в процедуры и функции через регистры);
3) удаление лишних команд;

4) оптимизация потока управления и удаление недостижимых участков программ;
5) снижение "стоимости" программы (есть «дорогие» и «дешевые» команды с точки зрения времени их выполнения процессором);
6) использование «машинных идиом» (например: xor ax,ax для обнуления регистра);
7) слияние, дробление и развертывание циклов, иногда требующееся из-за технических особенностей аппаратуры;

8) учет векторных и конвейерных свойств архитектуры

КРИТЕРИИ:
За отсутствие или ошибку для каждого из пунктов 1-3 : -2 балла
За отсутствие или ошибку для каждого из остальных пунктов: -1

10. Дана польская инверсная запись фрагмента программы, в котором нет операторов перехода `goto` и `continue`, но есть один оператор `break`. Вставить пропущенные в польской записи команды перехода (’!’,’!F’) и недостающие метки; восстановить фрагмент на языке Си. (Красным выделены пропущенные клетки).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
i	1	=	;	i	N	<	38	!F	17	!	i	#+	;	5	!	z	0	!=	36	!F	i	2

24	25	26	27	28	29	30	31	32	33	34	35	36	37	
%	1	!=	31	!F	36	!	z	–#	;	17	!	12	!	

Ответ:

```
for (i=1; i<N; i++)
    while (z!=0)  {
       if (i%2!=1)  break;
       --z;
    }
```

Другой правильный ответ (обнаружен в процессе экзамена):

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
i	1	=	;	i	N	<	38	!F	17	!	i	#+	;	5	!	z	0	!=	31	!F	i	2

24	25	26	27	28	29	30	31	32	33	34	35	36	37	
%	1	!=	31	!F	36	!	z	–#	;	38	!	12	!	

```
for (i=1; i<N; i++)
    if (z!=0)&&(i%2!=1); else
                {--z; break;}
    }
```

План решения:
1) Расставляем знаки операций '!' и '!F', исходя из следующих соображений:
если перед пропущенным местом стоит логическое выражение и нет операции ';', то переход условный, иначе — безусловный.
2) Расставляем метки, учитывая, что при трансляции управляющих конструкций языка Си:
(а) переходы по лжи возможны только вперед;
(б) безусловные переходы назад (кроме явных goto) возможны только в циклах.

КРИТЕРИИ: Есть ошибки в ПОЛИЗЕ: -5;
Есть ошибки в восстановленном фрагменте: -5;
Если без ошибок *только* операции перехода расставлены, даем 2 балла за задачу.

Вариант ответа

```
for (i=1; i<N; i++)
    while (z!=0)  {
          if (i%2!=1)  else  --z;
    }
```

формально не является правильным, но семантически эквивалентен. За такое решение снимаем 1 балл.

Вариант 4_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to Taad \mid \varepsilon$
$T \to Ua \mid a$
$U \to Taa$
$U \to U$
$T \to T$

(а) Описать язык $L(G)$ в виде теоретико- множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).
Ответ:

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\mathfrak{I}$	$L(G) \in \mathfrak{I}$? (да/нет)
контекстно-свободные языки	
контекстно-зависимые языки	
языки типа 0	
регулярные языки	

(в) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.
Ответ:

2. Является ли грамматика $G = \langle \{0,1\}, \{A,S\}, P, S \rangle$, где $P = \{ S \to A; \quad A \to AA \}$
(а) приведенной ? (б) однозначной ? Ответ обосновать.

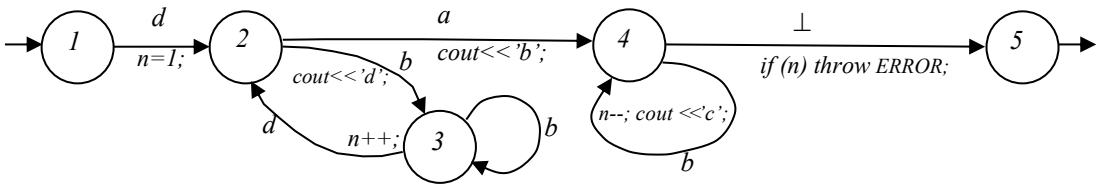
3. По заданной регулярной грамматике G построить конечный автомат в виде ДС. Детерминирован ли автомат? Ответ обосновать. Если автомат недетерминированный, то с помощью соответствующего алгоритма преобразовать его в эквивалентный ДКА.

G:
$S \to A\perp \mid B\perp$
$A \to Ad \mid Bd \mid a$
$B \to Bd \mid Aa \mid a$

4. Сформулировать критерий применимости метода рекурсивного спуска. Применим ли этот метод к данной КС-грамматике? Ответ обосновать.
$S \to A \mid cB$
$A \to aB \mid bBa \mid \varepsilon$
$B \to bB \mid \varepsilon$

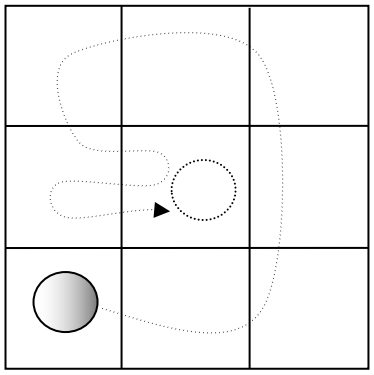
5. Дан автомат $\mathcal{A}$ в виде ДС с действиями. С помощью действий он допускает цепочки языка L_1 и переводит их в цепочки языка L_2. Определить языки L_1 и L_2. Построить КС-грамматику, анализируемую методом рекурсивного спуска, с действиями только вида *cout*<<'символ', задающую тот же перевод цепочек L_1 в цепочки L_2.

$\mathcal{A}$:



6. Привести три способа внутреннего представления программ и проиллюстрировать каждый способ на примере оператора a:=a+b. **Ответ:**

7. На клетчатой поверхности размера 3x3 в левой нижней клетке расположен робот-шар. Он умеет выполнять команды перехода в соседнюю клетку: `'a'` – переместиться на одну клетку вправо, `'b'` – влево, `'c'` – вверх, `'d'` – вниз. С помощью последовательности команд можно задавать траекторию движения шара. Например, последовательность `aaccbbdaba` перемещает шар в центральную клетку. Последовательность команд является цепочкой в алфавите {a,b,c,d}. Если цепочка пуста, шар остается на месте. Опишите ***регулярную*** грамматику с терминальным алфавитом {a,b,c,d}, задающую все возможные траектории шара, удовлетворяющие условиям: шар начинает движение из левой нижней клетки, может двигаться только вокруг центральной клетки против часовой стрелки, не покидая поверхности; остановиться может в любой клетке (кроме центральной). В грамматике должно быть не более 16 правил, считая альтернативы. **Ответ:**



8. Является ли тестирование видом верификации ? Ответ обосновать.

Ответ:

9. Перечислите основные функции отладчика в рамках интегрированной среды разработки программного обеспечения.

Ответ:

10. Дана польская инверсная запись фрагмента программы, в котором нет операторов перехода `break` и `continue`, но есть один оператор `goto`. Вставить пропущенные в польской записи команды перехода ('!','!F') и недостающие метки; восстановить фрагмент на языке Си.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
i	1	=	;	i	K	<			17		i	#+	;			j	1	=	;	j	L	<

24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
		33		j	#+	;			s	j	+=	;	s	N	>			48			!	12

47	48	49	50	51
	s	N	–=	;

Вариант 4_2011 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

За каждую задачу — максимум 10 баллов

1. Дана грамматика G:

$S \to Taad \mid \varepsilon$
$T \to Ua \mid a$
$U \to Taa$
$U \to U$
$T \to T$

(а) Описать язык $L(G)$ в виде теоретико- множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).
Ответ: $L(G) = \{ a^{3n}d \mid n \geq 1\} \cup \{\varepsilon\}$

(б) Каким из перечисленных классов принадлежит язык $L(G)$?

Ответ:

Класс $\Im$	$L(G) \in \Im$? (да/нет)
контекстно-свободные языки	**да (-1)**
контекстно-зависимые языки	**да (-1)**
языки типа 0	**да (-1)**
регулярные языки	**да (-3)**

(в) Классификация: найти такое целое k, что G является грамматикой типа k и не является грамматикой типа k+1.

Ответ: $k = 3$ (все правила леволинейны)

КРИТЕРИИ: (а) описание отсутствует или с ошибками: -4
(б) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице (обоснование не обязательно)
(в) нет ответа; неверный ответ: -4 (обоснование не обязательно)

2. Является ли грамматика $G = \langle \{0,1\}, \{A,S\}, P, S \rangle$, где $P = \{S \to A;\ A \to AA\}$
(а) приведенной ? (б) однозначной ? Ответ обосновать.

Решение

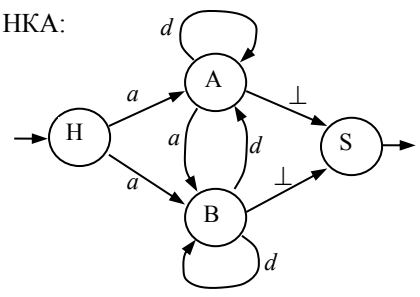
(а) **Нет**. Все символы грамматики бесполезны, т.е. недостижимы или бесплодны. Грамматика порождает пустой язык: $L(G) = \varnothing$. (Для обоснования достаточно указать хотя бы один бесполезный символ.)
(б) **Да**. Не существует цепочки, выводимой в грамматике G, имеющей два различных дерева вывода, так как язык пуст.

КРИТЕРИИ: ошибочные ответ или обоснование на первый вопрос: -5
ошибочные ответ или обоснование на второй вопрос: -5

3. По заданной регулярной грамматике G построить конечный автомат в виде ДС. Детерминирован ли автомат? Ответ обосновать. Если автомат недетерминированный, то с помощью соответствующего алгоритма преобразовать его в эквивалентный ДКА.

G:
$S \to A\perp \mid B\perp$
$A \to Ad \mid Bd \mid a$
$B \to Bd \mid Aa \mid a$

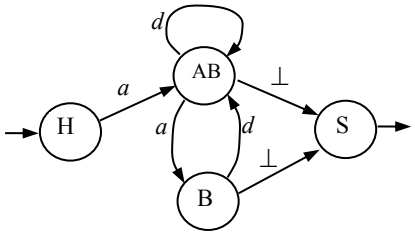
НКА:



Решение
Строим таблицу переходов ДКА:

	a	d	$\perp$
H	AB	$\varnothing$	$\varnothing$
AB	B	AB	S
B	$\varnothing$	AB	S
$\boldsymbol{S}$	$\varnothing$	$\varnothing$	$\varnothing$

По таблице переходов строим диаграмму ДКА:



(Допускается представить ДКА только в виде таблицы переходов (или набора команд δ) или только в виде диаграммы.)

КРИТЕРИИ:

Есть ошибки в НКА : -5

Нет обоснования или ошибочное : -3

Есть ошибки в преобразовании в ДКА: -5

4. Сформулировать критерий применимости метода рекурсивного спуска. Применим ли этот метод к данной КС-грамматике? Ответ обосновать.

$S \to A \mid cB$

$A \to aB \mid bBa \mid \varepsilon$

$B \to bB \mid \varepsilon$

Решение

Критерий применимости метода рекурсивного спуска.

Пусть G — КС-грамматика. Метод рекурсивного спуска применим к G, если и только если для любой пары альтернатив $X \to \alpha \mid \beta$ выполняются следующие условия:

(13) $first(\alpha) \cap first(\beta) = \varnothing$;

(14) справедливо не более чем одно из двух соотношений: $\alpha \Rightarrow \varepsilon$, $\beta \Rightarrow \varepsilon$;

(15) если $\beta \Rightarrow \varepsilon$, то $first(\alpha) \cap follow(X) = \varnothing$.

Вычислим необходимые для проверки критерия множества.

$first(A)=\{a,b\}$, $first(cB)=\{c\}$, $first(aB)=\{a\}$, $first(bBa)=\{b\}$, $first(bB)=\{b\}$,
$first(\varepsilon)=\varnothing$, $follow(A)=\varnothing$, $follow(B)=\{a\}$.

Видно, что пересечение множеств *first* и *follow* для соответствующих нетерминалов пусто, поэтому выполняются пункты (1) и (3) критерия применимости. Кроме того, у каждого нетерминала не более одной альтернативы, из которой выводится ε, поэтому выполняется пункт (2).

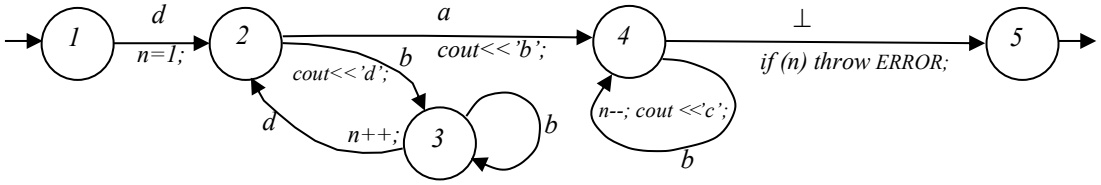
КРИТЕРИИ:

Есть ошибки в формулировке критерия: -5

Есть ошибки в обосновании: -5

5. Дан автомат $\mathcal{A}$ в виде ДС с действиями. С помощью действий он допускает цепочки языка L_1 и переводит их в цепочки языка L_2. Определить языки L_1 и L_2. Построить КС-грамматику, анализируемую методом рекурсивного спуска, с действиями только вида *cout*<<'символ', задающую тот же перевод цепочек L_1 в цепочки L_2.

$\mathcal{A}$:



Решение

Автомат $\mathcal{A}$ с действиями допускает цепочки вида $dy_1y_2\ldots y_nab^{n+1}\perp$, где $n\geq 0$, $y_i = bx_id$, $x_i \in \{b\}^*$ для $i = 1,\ldots,n$, и каждую такую цепочку переводит соответственно в цепочку d^nbc^{n+1}.

Грамматика с действиями такова (запись вида $\langle a\rangle$ означает $\langle cout<<'a'\rangle$):

$S \to dAb\langle c\rangle\perp$

$A \to bBd\langle d\rangle Ab\langle c\rangle \,|a\,\langle b\rangle$ (или $A \to b\langle d\rangle BdAb\langle c\rangle \,|a\,\langle b\rangle$)

$B \to bB \,|\, \varepsilon$

Возможен и такой вариант ответа:

$S \to aAb\langle c\rangle\perp$

$A \to bBaAb\langle c\rangle \,|\, a\,\langle b\rangle$

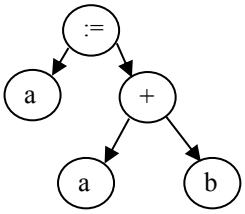
$B \to bB \,|\, \varepsilon\ \langle d\rangle$

КРИТЕРИИ: -- есть ошибки в описаниях языков L_1 , L_2 : -3
-- грамматика для L_1 ошибочна: -4
-- метод рекурсивного спуска неприменим: -2
-- есть действие не вида *cout*<<'символ' : -10
-- конечное число цепочек переводятся неправильно: -3
-- бесконечно много цепочек переводятся неправильно: -6

6. Привести три способа внутреннего представления программ и проиллюстрировать каждый способ на примере оператора a:=a+b.

Ответ:

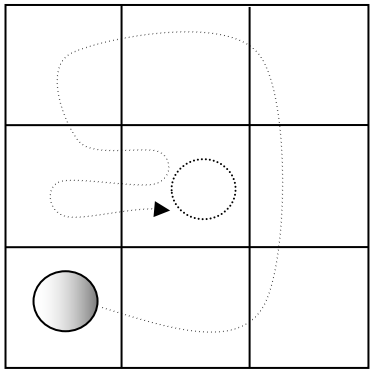
1) ПОЛИЗ: a, a,b +, :=
2) Префиксная запись: :=, a, +, a, b
3) Многоадресный код с явно именуемым результатом (тетрады):
   1 + a b T1
   2 := T1 ? a
4) Многоадресный код с неявно именуемым результатом (триады):
   1 + a b
   2 := a ^1
   (вторая команда в этом примере может быть удалена при оптимизации)
5) Динамические структуры, представляющие синтаксическое дерево
   Пример – дерево операций:



Допустимы и другие внутренние представления, например язык ассемблера, байт-код и т.п.

КРИТЕРИИ: за каждый ошибочный/отсутствующий пункт (из трех): -4
если отсутствует только пример: -1

7. На клетчатой поверхности размера 3x3 в левой нижней клетке расположен робот-шар. Он умеет выполнять команды перехода в соседнюю клетку: `'a'` – переместиться на одну клетку вправо, `'b'` – влево, `'c'` – вверх, `'d'` – вниз. С помощью последовательности команд можно задавать траекторию движения шара. Например, последовательность `aaccbbdaba` перемещает шар в центральную клетку. Последовательность команд является цепочкой в алфавите {a,b,c,d}. Если цепочка пуста, шар остается на месте. Опишите ***регулярную*** грамматику с терминальным алфавитом {a,b,c,d}, задающую все возможные траектории шара, удовлетворяющие условиям: шар начинает движение из в левой нижней клетки, может двигаться только вокруг центральной клетки против часовой стрелки, не покидая поверхности; остановиться может в любой клетке (кроме центральной). В грамматике должно быть не более 16 правил, считая альтернативы.



Решение

7	6	5
8		4
1	2	3

Согласно условиям, шар, начав с левой нижней клетки, дальше может двигаться вокруг центральной клетки против часовой стрелки и может остановиться в любой момент. Перенумеруем против часовой стрелки все клетки (кроме центральной), начиная с левой нижней. Построим конечный автомат с состояниями S_1,S_2, …, S_8. Состояние S_i соответствует положению шара в *i-й* клетке. Дуга из S_i в S_{i+1} помечается подходящей буквой из алфавита {a,b,c,d}. Например, переход из S_1 в S_2 осуществляется по команде `'a'`, и соответствующая дуга помечается символом `'a'`. Начальным является состояние S_1. Заключительными будут все состояния, так как шар может остановиться в любой из пронумерованных клеток. По конечному автомату построим праволинейную автоматную грамматику с начальным символом S_1:

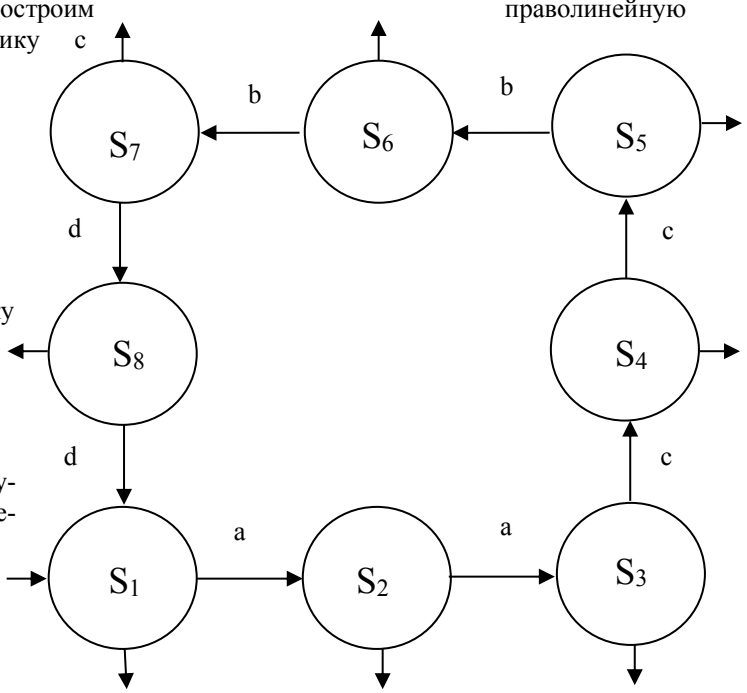
$S_1 \rightarrow aS_2 \mid \varepsilon$

$S_2 \rightarrow aS_3 \mid \varepsilon$

$S_3 \rightarrow cS_4 \mid \varepsilon$

...

$S_8 \rightarrow dS_1 \mid \varepsilon$



Можно построить эквивалентную регулярную грамматику с начальным символом S, состоящую из 10 правил:
S→ `aaccbbddS|ε|a| aa|aac| aacc| aaccb| aaccbb| aaccbbd| aaccbbdd`

КРИТЕРИИ: не допускается ε: -2, за недопуск любой другой правильной цепочки или за допуск каждой лишней цепочки: -5. Нерегулярная грамматика: -10

8. Является ли тестирование видом верификации ? Ответ обосновать.

Ответ: **Да**. Верификация – это процесс проверки программы на правильность. Кроме логических методов доказательства правильности программы существуют эвристические, основанные на тестировании. Тестирование — это процесс обнаружения дефектов (несоответствия программы исходным требованиям и спецификациям) путем проверки программы на данных с заранее известными ответами (тестах)

КРИТЕРИИ:

Нет обоснования, но ответ верно угадан: 1 балл

За отсутствие или ошибочность определения верификации: -5

За отсутствие или ошибочность определения тестирования: -5

9. Перечислите основные функции отладчика в рамках интегрированной системы разработки программного обеспечения.

Ответ:
1) пошаговое выполнение программы (шаг = строка; с трассировкой внутри вызываемой функции и без нее);
2) выполнение программы до строки, в которой в редакторе стоит курсор;
3) выделение выполняемой в данный момент строки;
4) приостановка выполнения программы:
   -- можно запросить значение переменной,
   -- можно заказать вычисление некоторого выражения,
   -- можно изменить значение переменной и продолжить выполнение
      программы (!но редко какой отладчик позволяет изменять программный
      код, т.е. поддерживает частичную перекомпиляцию);
5) расстановка/снятие точек останова, которые визуализируются в текстовом редакторе;
6) выдача всей информации в терминах исходной программы.

КРИТЕРИИ: достаточно привести пять из шести пунктов
За ошибку/отсутствие для каждого из пяти пунктов: -2

10. Дана польская инверсная запись фрагмента программы, в котором нет операторов перехода `break` и `continue`, но есть один оператор `goto`. Вставить пропущенные в польской записи команды перехода (’!’,’!F’) и недостающие метки; восстановить фрагмент на языке Си. (Красным выделены пропущенные клетки).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
i	1	=	;	i	K	<	48	!F	17	!	i	#+	;	5	!	j	1	=	;	j	L	<

24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
46	!F	31	!	j	#+	;	17	!	s	j	+=	;	s	N	>	44	!F	44	!	28	!	12

47	48	49	50	51
!	s	N	–=	;

Ответ:

```
for (i=1; i<K; i++)
    for (j=1; j<L; j++) {
         s+=j;
         if (s>N) goto cont;
    }
cont: s-=N;
```

План решения:
1) Расставляем знаки операций ’!’ и ’!F’, исходя из следующих соображений:
если перед пропущенным местом стоит логическое выражение и нет операции ’;’, то переход условный, иначе — безусловный.
2) Расставляем метки, учитывая, что при трансляции управляющих конструкций языка Си:
   (а) переходы по лжи возможны только вперед;
   (б) безусловные переходы назад (кроме явных goto) возможны только в циклах.

КРИТЕРИИ: Есть ошибки в ПОЛИЗЕ: -5
Есть ошибки в восстановленном фрагменте: -5
Если без ошибок *только* операции перехода расставлены, даем 2 балла за задачу.

2012 год
Коллоквиум

Вариант 1 _2012 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

Во всех заданиях считать, что там, где это необходимо, произведены нужные включения и использован оператор using namespace std;

1. Что такое АТД? Каким образом АТД реализуется в С++?

2. Описать класс А так, чтобы: - все конструкции функции main ( ) были верными,
- явно в классе А можно описать не более одного конструктора,
- на экран выдалось **15 60 7**.

Нельзя использовать исключения и любые функции досрочного завершения программы.

```
int main () {
        A a1 (5),   a2 = 4,  a3;
        a2  *=   a1  *=  3;
        cout << a1.get ( ) <<  ' ' << a2.get  () << ' ' <<  a3.get ( ) << endl;
        return 0;
}
```

3. Что будет выдано на печать при работе следующей программы?

```
struct S {
    int x;
    S (int n) {   x = n;   printf (" Cons    "); }
    S (const S & a) {  x = a.x;   printf (" Copy    "); }
    ~S ( ) {  printf ("Des   "); }
};
S  f ( S  y ) {  y = S (3);  return y; }
int main ( ) {
    S s (1);
    f (s);
    printf ("%d   ",  s.x);
    return 0;
}
```

4. Внести добавления в описания заданных методов (**не меняя вывод на экран!**) структур B и D так, чтобы все конструкции main () были правильными, а на печать выдалось **5535324242**.

```
struct B {
    float x;
    B (float a) {   x = a;   cout << 5; }
    ~B ( ) {  cout << 2; }
};
struct D : B {
    D ( ) { cout << 3; }
    ~D ( ) {  cout << 4; }
};
int main ( ) {
    B  * p1 = new B (1),  * p2 = new D[2];
    delete p1;
    delete [ ] p2;
    return 0;
}
```

5. Добавить (если нужно) **<u>в класс А</u>** сл.слова «**const**» , так, чтобы заданный фрагмент программы был верным.

```
class A {
    int i;
public:
```

```
        A (int x) { i = x; }
        A (A & y) { i = y.i; }
        const A f (const A & z) { cout << endl;  return *this;}
};
const A  t1 ( ) {
        const A a = 5;
        return a.f ( a );
}
```

6. Написать:

1) функцию – шаблон (от одного параметра–типа: контейнера STL целых чисел), которая для любого последовательного контейнера STL распечатывает его предпоследний элемент, если таковой имеется,

2) а также функцию main (), которая формирует контейнер-список из 5 целых чисел и применяет к нему написанную функцию-шаблон.

7. Что напечатает следующая программа?

```
struct B {
        virtual void f (int n) { cout << "f (int) from B\n"; }
        static int i;
};
struct D:  B {
        virtual void f (char n) { cout << "f (char) from D\n"; }
};
int B::i = 1;
int main ( ) {    D  d;  B  b1,   b2,   *pb = &d;
                  pb  -> f ( 'a');
                  b1.i += 2;          b2.i += 3;          d.i += 4;
                  cout << b1.i << ' ' << b2.i << ' ' << d.i << ' ' <<  B::i << endl;       return 0;
}
```

8. Что напечатает следующая программа?

```
struct S {
        S ( int a) {   try  { if (a > 0)  throw  *this;
                              else
                                    if (a < 0)  throw 0;
                        }
                        catch ( S & ) { cout << "SCatch_S&\n"; }
                        catch (int) {  throw; }
                        cout << "SConstr\n";
        }
        S (const S & a) { cout << "Copy\n"; }
        ~S ( ) { cout << "Destr\n"; }
};
int  main ( ) {   try  {    S  s1(1),     s2(-2);
                            cout << "Main\n";
                  }
                  catch (S  &) {   cout << "MainCatch_S&\n";   }
                  catch  ( ... ) {   cout << "MainCatch_...\n";   }
                  return 0;
}
```

9. Добавить в функцию f9 использование **механизма RTTI** так, чтобы ее выполнение всегда завершалось нормально.

```
struct B {
        virtual void g ( ) { }
};
struct D:  B {
        char y [100] ;
};

void f9 ( B & b,   D & d,  int  n )   {
        D * pd = ( n > 0 )  ?  & d  :  (D*) & b;
        strcpy (pd -> y, "one_variant\n");

}
```

10. Назвать три разных ситуации, когда **автоматически** вызывается деструктор.

Ответы - вариант 1_2012

Максимальная оценка каждой задачи – 10 баллов (всего 100 баллов)

1.

АТД - тип данных с полностью скрытой (инкапсулированной) структурой, а работа с переменными такого типа происходит только через специальные, предназначенные для этого функции.

В С++ АТД реализуются с помощью классов (структур), **все члены-данные** которых расположены в **private** области.

Критерии: Неверное понятие АТД - **-5,**
Неверный ответ о реализации АТД в С++ - **-5.**
За отсутствие слов о работе с переменными АТД – не наказывать.

2 .

```
class A {
     int x;
public:
     A ( int y = 7) { x = y; }
     int get () { return x; }
     int  operator *=  ( int y ) { return  x *=  y; }
};
```

Критерии: За каждый недостающий/неверный метод, /лишний констр. - **- 4**
За каждую мелкую ошибку реализации - **- 2**

3.

Cons Copy Cons Des Copy Des Des 1 Des

Критерии: За каждую неверную/лишнюю/недостающую печать - **- 4**
Если пропущен конструктор, то за парный ему деструктор - не наказывать.

4.

1) перед деструктором базового класса добавить **virtual.**
2) добавить либо B (float a **= 0**), либо D ( ) : **B (0)** {...}

Критерии: За каждое не сделанное добавление - **- 5**

5.

```
class A {
        int i;
public:
        A ( int x) { i = x; }
        A (const  A & y) { i = y.i; }
```

```
        const A f (const A & z)  const { cout <<  i << endl; }
};

const A  t1 ( ) {
        const A a = 5;
        return a.f ( a );
}
```

Критерии: За каждое не найденную или лишнюю ошибку - **- 5**

6.

```
template < class T >
void f (const T & t) {
if (t.size ( ) < 2) return ;
typename T :: const_reverse_iterator p = t.rbegin ( ) ;
cout << * ( ++p ) << endl;  //  можно без скобок
}
int main () {
list <int >  lst;
for (int i = 0; i < 5; i++)
        lst.push_back (i);
f (lst); // или f <list <int> > (lst);
return 0;
}
```

Критерии: Пропуск typename - **-4**,
Если параметр функции const, а итератор – не const - **-4**,
За каждое другую ошибку - **- 3**
Отсутствие & в заголовке функции - **-1**.

7.

```
f (int) from B
10  10  10  10
```

Критерии: За неверный вызов функции - **- 5**
За любую ошибку со статической переменной - **- 5**

8.

```
Copy ¬  SCatchS&  ¬  Destr ¬  SConstr  ¬  Destr  ¬  MainCatch...
```

Критерии: За отсутствие Copy (и его Destr) - **- 5**
За каждую другую ошибку - **- 2**

9.

```
void f9 ( B & b,   D & d,  int  n )  {
        D * pd = ( n > 0 )  ?  & d  :  (D*) & b;
        if (typeid (*pd) == typeid (d))  // возможен вариант с dynamic_cast
                strcpy (pd -> y, "one_variant\n");

}
```

Критерии: Правильно добавлены средства RTTI - **- 10**
Иначе - **- 0**

10.

1. при свертке стека - при выходе из блока описания объекта, в частности при обработке исключений, завершении работы функции;
2. при уничтожении временных объектов - сразу, как только завершается конструкция, в которой они использовались;
3. при выполнении операции delete для указателя на объект (инициализация указателя - с помощью операции new),
4. при завершении работы программы при удалении глобальных/статических объектов.

Критерии: За отсутствие каждой из трех ситуаций - **- 4**

Вариант 2 _2012 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

Во всех заданиях считать, что там, где это необходимо, произведены нужные включения и использован оператор using namespace std;

1. Какие способы наследования базового класса производным классом приняты в C++? В чем отличие статуса private и public членов базового класса в производном классе при **private** наследовании?

2. Описать класс B т.о., чтобы: - все конструкции функции main ( ) были верными,
- класс B содержал только один явно описанный конструктор,
- на экран выдалось **17 11 6**.

Нельзя использовать исключения и любые функции досрочного завершения программы.

```
int main () {
        B b1 (1),  b2 (2, 3), b3 (b1);
        b1 += b2 += b3;
        cout << b1.get() << ' ' << b2.get() <<  ' ' <<  b3.get () << endl;
        return 0;
}
```

3. Что будет выдано на печать при работе следующей программы?

```
#include <iostream>
struct S {
        int x;
        S (int n) {   x = n;   printf (" Cons    "); }
        S (const S & a) {  x = a.x;   printf (" Copy    "); }
        ~S ( ) {  printf ("Des   "); }
};
S  f ( S & y ) {  y = S (3);  return y; }
int main ( ) {
        S s (1);
        f (s);
        printf ("%d   ",  s.x);
        return 0;
}
```

4. Внести добавления в описания заданных методов (**не меняя вывод на экран!**) структур B и D так, чтобы все конструкции main () были правильными, а на печать выдалось **11163343**.

```
struct B {
        int x;
        B (int a) {   x = a;   cout << 1; }
        ~B ( ) {  cout << 3; }
};
struct D : B {
        D (int d ) : B (d) { cout << 6; }
        ~D ( ) {  cout << 4; }
};
int main ( ) {
        B  * p1 = new B [2],  * p2 = new D (1);
        delete  [ ] p1;
        delete  p2;
        return 0;
}
```

5. Добавить (если нужно) **в класс Cl** сл.слова «**const**» , так, чтобы заданный фрагмент программы был верным.

```
class Cl {
```

```
        int i;
    public:
        Cl (int x) { i = x; }
        Cl (Cl & y) { i = y.i; }
        const Cl   f ( Cl & c) const { cout << c. i << endl;    return *this; }
    };
    const Cl  t1 (const Cl a) {
        Cl b  = Cl (5);
        return b.f ( a );
    }
```

6. Написать:

1) функцию – шаблон (от одного параметра–типа: контейнера STL целых чисел), которая для любого последовательного контейнера STL распечатывает сумму его трех последних элементов, если таковые имеются,

2) а также функцию main (), которая формирует контейнер-вектор из 5 целых чисел и применяет к нему написанную функцию-шаблон.

7. Что напечатает следующая программа?

```
struct K {
        virtual void add_st ( K * n ) { st ++;    cout << "add_st (K*) from K\n"; }
        static int st;
};
struct L:  K {
        virtual void add_st ( L * a ) { st++;    cout << "add_st (L*) from L\n"; }
};
int K::st = 2;
int main ( ) {
        L   ob,  ob2;      K   k ,  *pl = &ob;
        pl  -> add_st (& ob2);
        k.st ++;      ++ob.st ;
        cout << k.st << ' ' << ob.st << ' ' <<  K::st << endl;          return 0;
}
```

10. Что напечатает следующая программа?

```
struct S {
        S ( int a) {        try {     if (a > 0)   throw  *this;
                                      else
                                              if (a < 0)   throw 0;
                            }
                            catch ( S & ) { cout << "SCatch_S&\n"; }
                            catch (int) {  throw; }
                            cout << "SConstr\n";
        }
        S (const S & a) { cout << "Copy\n"; }
        ~S ( ) { cout << "Destr\n"; }
};
int  main ( ) {
        try  {    S   s1( 0 ),      s2 ( 5 );
                  cout << "Main\n";
        }
        catch (S  &) {   cout << "MainCatch_S&\n";   }
        catch ( ... ) {   cout << "MainCatch_...\n";   }
        return 0;
}
```

11. Добавить в функцию putnull использование механизма RTTI так, чтобы ее выполнение всегда завершалось нормально.

```
struct B {
        virtual void empty ( ) { }
};
struct D:  B {
        int  mas [30] ;
};

void putnull ( B * pb )   {      D d ,  *pd = (D*) pb;
                                 if  ( ! pb )  return;
                                 for (int i = 0;  i < 30;  i++) pd -> mas[i] = 0 ;

}
```

10. Назвать три разных ситуации, когда **автоматически** вызывается конструктор копирования.

Ответы - вариант 2_2012

Максимальная оценка каждой задачи – 10 баллов (всего 100 баллов)

1.

В C++ наследование бывает public, private и protected. Private-члены базового класса **видны**, но **недоступны** в производном классе, public-члены – и **видны**, и **доступны**, но только внутри производного класса.

Критерии: За каждый недостающий способ наследования - **-4**
Неверный ответ о статусе private и public членов - **-4**
Если не упомянута видимость - не наказывать**.**

2.

```
class B {
     int x, y;
public:
     B (int a, int b = 5) { x = a;  y = b; }
     int get ( ) { return x + y; }
     B& operator=+ (B & b){ x += b.x; y += b.y; return *this;}
};
```

Критерии: За каждый недостающий/неверный метод - **- 4**
За каждую мелкую ошибку реализации - **-2**

3.

Cons Cons Des Copy Des 3 Des

Критерии: За каждую неверную/лишнюю/недостающую печать - **- 4**
Если пропущен конструктор, то за парный ему деструктор - не наказывать.

4.

1) перед деструктором базового класса добавить **virtual.**
2) добавить B (int a **= 0**) {...}

Критерии: За каждое не сделанное добавление - **- 5**

5.

```
class Cl {
        int i;
```

```
public:
        Cl (int x) { i = x; }
        Cl ( const Cl & y) { i = y.i; }
        const Cl   f ( const Cl & c) const { cout << c. i << endl; }
};

const Cl  t1 (const cl a) {
        Cl b  = Cl (5);
        return b.f ( a );
}
```

Критерии: За каждое не найденную или лишнюю ошибку - **- 5**

7.

```
template < class T >
void f (const T & t) {
if (t.size ( ) < 3) return ;
typename T :: const_reverse_iterator p = t.begin ( ) ;
cout << * p + * ( ++p ) + * ( ++p ) << endl;  //  можно без скобок и лучше не
                                                // рассчитывать на порядок вычисления операндов.
}
int main () {
vector <int >  vec (5);
for (int i = 0; i < 5; i++)
        vec.push_back (i);
f (vec);  // или f <vector <int> > (vec);
return 0;
}
```

Критерии: Пропуск typename - **-4**,
Если параметр функции const, а итератор – не const - **-4**,
За каждое другую ошибку - **- 3**
Отсутствие & в заголовке функции - **-1**.

7.

```
add_st (K*) from K
5   5   5
```

Критерии: За неверный вызов функции - **- 5**
За любую ошибку со статической переменной - **- 5**

8.

SConstr ¬ Copy ¬ SCatchS& ¬ Destr ¬ SConstr ¬ Main ¬ Destr ¬ Destr

Критерии: За отсутствие Copy (и его Destr) - **- 5**
За каждую другую ошибку - **- 2**

9.

```
void putnull ( B * pb )  {
        try {     D  d, * pd =  (D*) pb;
                  if (typeid (*pb) == typeid (d))  // возможен вариант с dynamic_cast
                          for (int i = 0; i < 30; i++)   pd -> mas [ i ] = 0;
        }
        catch ( bad_typeid ) { cout << "NULL!\n"; }
}
```

Критерии: Правильно добавлены средства RTTI - **- 10**
Иначе - **- 0**
За отсутствие try-catch блока не наказывать,
если это не может привести к ошибке.

10.

1. в случае: Box a (1, 2, 3); Box b = a;
2. в случае: Box c = Box (3, 4, 5); // оптимизирующий компилятор вызывает только обычный конструктор с указанными параметрами;

3. при передаче параметров функции по значению (при создании локального объекта);
4. при возвращении результата работы функции в виде объекта,
5. при генерации исключения-объекта класса.

Критерии: За отсутствие каждой из трех ситуаций - **- 4**

Вариант 3_2012 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

Во всех заданиях считать, что там, где это необходимо, произведены нужные включения и использован оператор using namespace std;

1. Какие виды полиморфизма реализованы в C++, с помощью каких механизмов? Когда происходит выбор конкретной языковой конструкции для каждого вида полиморфизма?

2. Описать класс C т.о., чтобы: **-** в main ( ) ошибочным было только описание объекта c2,
- класс C содержал только один явно описанный конструктор,
- после удаления описания c2 на экран выдалось **14 56**.

Нельзя использовать исключения и любые функции досрочного завершения программы.

```
int main () {
        C c1 (7),  c2 = 5, c3 (c1 + c1);
        cout << c1.get ( ) <<  ' '  <<  c3.get ( ) << endl;
        return 0;
}
```

3. Что будет выдано на печать при работе следующей программы?

```
struct S {
        int x;
        S ( int n ) {   x = n;   printf ( " Cons    " ); }
        S ( const S &  a ) {  x = a.x;   printf ( " Copy    " ); }
        ~S ( ) {  printf ( "Des   " ); }
};
S &  f  ( S  y ,  S & z) {  y =  S (3);   return z; }
int main ( ) {
        S s (1);
        f ( s,  s );
        printf ( "%d   ",  s.x );
        return 0;
}
```

4. Внести добавления в описания заданных методов (**не меняя вывод на экран!**) структур B и D так, чтобы все конструкции main () были правильными, а на печать выдалось **776898**.

```
struct B {
        int x;
        B ( ) {   x = 7;   cout << 7; }
        ~B ( ) {  cout << 8; }
};
struct D : B {
        D ( int d) { x  = d ;  cout << 6; }
        ~D ( ) {  cout << 9; }
};
int main ( ) {
        B  * p1 = new B [1],  * p2 = new D[1];
        delete [ ] p1;
        delete [ ] p2;
        return 0;
}
```

5. Добавить (если нужно) **в класс Cls** сл.слова «**const**» , так, чтобы заданный фрагмент программы был верным.

```
class Cls {
        int i;
public:
        Cls (int x) { i = x; }
        Cls (Cls & y) { i = y.i; }
        const Cls   f ( const Cls  c) { cout << c. i << endl;   return *this; }
};
const Cls  t1 (const Cls * a) {
        Cls b  =  Cls (3);
        return  a -> f ( b );
}
```

6. Написать

1) функцию – шаблон (от одного параметра–типа: контейнера STL целых чисел), которая для любого последовательного контейнера STL распечатывает каждый второй элемент, начиная с конца,

2) а также функцию main (), которая формирует контейнер-список из 5 целых чисел и применяет к нему написанную функцию-шаблон.

7. Что напечатает следующая программа?

```
struct S {
        static double d;
        virtual  S &  g ( ) { cout << "g ( ) from S\n"; }
};
struct T:  S {
        virtual  T & g ( ) { cout << "g ( ) from T\n"; }
};
double S::d = 1.5;
int main ( ) {
        T   t;    S   s ,  *ps = &t;
        ps  -> g ( );
        s.d = 5;    t.d = 7;
        cout << s.d << ' ' << t.d << ' ' <<  S::d << endl;     return 0;
}
```

12. Что напечатает следующая программа?

```
struct S {
        S ( int a) {        try  {    if (a > 0)  throw  *this;
                                      else
                                              if (a < 0)  throw 0;
                            }
                            catch ( S & ) { cout << "SCatch_S&\n";  throw;}
                            catch (int) {  cout << "SCatch_int\n"; }
                            cout << "SConstr\n";
        }
        S (const S & a) { cout << "Copy\n"; }
        ~S ( ) { cout << "Destr\n"; }
};
int  main ( ) {
        try  {    S   s1(-3),     s2(25);
                  cout << "Main\n";
        }
        catch (S  &) {   cout << "MainCatch_S&\n";   }
        catch ( ... ) {   cout << "MainCatch_...\n";   }
        return 0;
```

```
}
```

13. Добавить в функцию puthi использование механизма RTTI, так, чтобы ее выполнение всегда завершалось нормально.

```
struct B {
        virtual void hi ( ) { cout << "Hi!\n" }
};
struct D:  B {
        char txt [ 10 ] [ 4 ] ;
};
void puthi ( B *  pb,   D *  pd )   {       D d;  int  i = 10;
                                            if ( ! pb )  return;
                                            pd = (D*) pb;
                                            while ( i )  strcpy ( (pd -> txt) [ -- i ] , "Hi!");

}
```

10. Назвать три разных ситуации, когда **автоматически** вызывается обычный конструктор (не копирования).

Ответы - вариант 3_2012

Максимальная оценка каждой задачи – 10 баллов (всего 100 баллов)

1.

Различаются следующие виды полиморфизма:

статический (на этапе компиляции, реализуется с помощью перегрузки функций/операций),

динамический (во время выполнения программы, реализуется с помощью виртуальных функций),

параметрический (на этапе компиляции, с использованием механизма шаблонов).

Критерии: За каждый недостающий вид полиморфизма - **-4,**
За каждый неназванный механизм реализации - **-2,**
За каждый неверный ответ о этапе разрешения полиморфизма - **-2.**

2.

```
class C {
     int x;
public:
explicit  C (int y) { x = 2 * y; }
          int get () { return x; }
          C operator + ( C  c ) {
               C t ( x + c.x );
               return  t;
          }
};
```

Критерии: За каждый недостающий, лишний метод, /лишний констр.- **- 4**
За отсутсивие explicit - -4
За каждую ошибку реализации метода (неверный вывод…) - **- 2**

3. **Cons Copy Cons Des Des 1 Des**

Критерии: За каждую неверную/лишнюю/недостающую печать - **- 4**
Если пропущен конструктор, то за парный ему деструктор - не наказывать.

4.

1) перед деструктором базового класса добавить **virtual.**
2) добавить D (int d **= 0**) {...}

Критерии: За каждое не сделанное добавление - **- 5**

5.

```
class Cls {
        int i;
public:
        Cls (int x) { i = x; }
        Cls (const Cls &  y) { i = y.i; }
        const Cls   f ( const Cls  c) const{ cout << c. i << endl; }
};
const Cls  t1 (const Cls * a) {
        Cls b  = Cls (3);
        return  a -> f ( b );
}
```

Критерии: За каждое не найденную или лишнюю ошибку - **- 5**

8.

```
template < class T >
void f (const T & t) {
if (t.size ( ) < 2) return ;
typename T :: const_reverse_iterator p = t.rbegin ( ) ;
for (int i = 0; i < t.size() / 2; i++, p++)
        cout << * ( ++p ) << endl;  //  можно без скобок
}
int main () {
list <int >  lst;
for (int i = 0; i < 5; i++)
        lst.push_back (i);
f (lst); // или f <list <int> > (lst);
return 0;
}
```

Критерии: Пропуск typename - **-4**,
Если параметр функции const, а итератор – не const - **-4**,
За каждое другую ошибку - **- 3**
Отсутствие & в заголовке функции - **-1**.

7.

```
g ( ) from T
7  7  7
```

Критерии: За неверный вызов функции - **- 5**
За любую ошибку со статической переменной - **- 5**

8.

SCatch_int ¬ SConstr ¬ Copy ¬ SCatchS& ¬ Destr ¬ MainCatchS& ¬ Destr

Критерии: За отсутствие Copy (и его Destr) - **- 5**
За каждую другую ошибку - **- 2**

9.

```
void puthi ( B * pb,   D * pd )   {
        try {    D  d ;    int i = 10;
                if (typeid (*pb) == typeid (d)) { // возможен вариант с dynamic_cast
                        pd = (D*) pb;
                        while (i)  srtcpy ((pd -> txt) [--i] , "Hi!");}
        }
```

```
        catch ( bad_typeid ) { cout << "NULL!\n"; }
    }
```

Критерии: Правильно добавлены средства RTTI - **- 10**
Иначе - **- 0**
За отсутствие try-catch блока не наказывать,
если это не может привести к ошибке.

10.

2. при создании объекта (при обработке описания объекта),
3. при создании объекта в динамической памяти (по new),
4. при композиции объектов наряду с собственным конструктором вызывается конструктор объекта – члена класса,
5. при создании объекта производного класса также вызывается конструктор и базового класса,
6. при автоматическом приведении типов с помощью конструктора преобразования.

Критерии: За отсутствие каждой из трех ситуаций - **- 4**

2012 год
Экзамен

Вариант 1_2012 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to aS \,|Sb|aAb|ab$
$aAb \to aaAbb|aabb$
$aaAbb \to aaAbb$
$aAb \to aAb$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

Класс П	$G \in$ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

(в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком k+1.

2. Среди правил грамматик G_1 и G_2 найти и вычеркнуть одно правило (альтернативу) так, чтобы G_1 и G_2 стали эквивалентными. Ответ обосновать.

G_1: $S \to aAa \,|aSaB$
$A \to Aa \,|BA \,|BS$
$B \to bC$
$C \to \varepsilon$

G_2: $S \to aAa \mid aB \,|aS$
$Y \to Ya \,|b \,|YS$
$B \to bA \mid bBC \mid \varepsilon$
$C \to bAY$

3. а) Построить по заданной грамматике G конечный автомат (в виде ДС).
б) Если автомат оказался недетерминированным, преобразовать его в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА.
в) По ДКА построить леволинейную грамматику.

G: $H \to 1A \mid 1B$
$A \to 0B \mid 1C$
$B \to 1A \mid \perp$

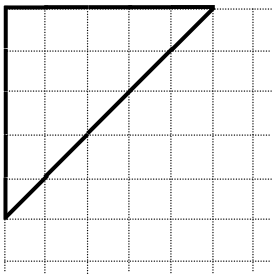
$C \to 1A \mid 1C$

4. Имеется клетчатый лист бумаги, бесконечный вправо и вниз. В левом верхнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*a*, *b, c*}:

a — переместить перо на одну клетку вправо;

b — переместить перо по диагонали на одну клетку влево-вниз;

c — переместить перо на одну клетку вверх.



Например, по последовательности *aaaaabbbbbccccc* интерпретатор построит прямоугольный равнобедренный треугольник с боковой стороной длины 5 (см. рисунок).

Построить грамматику с терминальным алфавитом {*a*, *b, c*}, описывающую подобные изображенному на рисунке треугольники с боковой стороной длины *n*, $n \geq 1$. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

5. Дана однозначная КС-грамматика $G_{balance}$, порождающая язык *L*, состоящий из цепочек, в которых символов *a* и *b* поровну:

$L=\{ x \in \{a,b\}^* \mid |x|_a = |x|_b \}$. Вставить в грамматику действия вида ⟨*cout* << "символы" ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(x, (ab)^{f(x)}) \mid x \in L, f(x) = |x|_a + |x|_b \}$.

$G_{balance}$:

$S \to aAbS \mid bBaS \mid \varepsilon$

$A \to aAbA \mid \varepsilon$

$B \to bBaB \mid \varepsilon$

6. Какие стратегии трансляции могут выбираться в различных системах программирования? Объяснить существенные отличия этих подходов к реализации процесса трансляции с языка программирования. Приведите по одному примеру формального языка, к которому применена каждая из названных вами стратегий трансляции.

7. В чём различие статических и динамических библиотек? Какие компоненты вычислительной системы занимаются подключением к программам каждого типа библиотек?

8. По заданной грамматике *G = {{+, –}, {R, S}, P, S}* получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

P: $S \to +R- \mid \varepsilon$
$R \to +R \mid -S \mid S$

9. Дана грамматика *G*. Применим ли к ней метод рекурсивного спуска? Ответ обосновать.

G: $S \to aSb \mid Yb \mid bb$
$Y \to cSY \mid dd \mid \varepsilon$

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&a	#+	4	+	b	<=	36	!F	&b	+#	a	b	—	<	34	!F	&a

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
b	8	—	7	*	b	a	@	23	/	/	—	=	;	9	!	39	!	&b	1	=	;

Вариант 1_2012 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to aS \mid Sb \mid aAb \mid ab$
$aAb \to aaAbb \mid aabb$
$aaAbb \to aaAbb$
$aAb \to aAb$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).
Ответ: $L(G) = \{a^n b^k \mid n, k \geq 1\}$

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

Класс П	$G \in$ П? (да/нет)
регулярные	**нет (-3)**
контекстно-зависимые	**да (-1)**
контекстно-свободные	**нет (-3)**
грамматики типа 0	**да (-3)**
неукорачивающие	**да (-1)**

(в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком k+1.
Ответ: $k = \mathbf{3}$ (для заданного языка существует порождающая его регулярная грамматика)
КРИТЕРИИ: (а) описание отсутствует или с ошибками: **–4**
(б) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице (обоснование не обязательно)
(в) нет ответа; неверный ответ: **–4** (обоснование не обязательно)

2. Среди правил грамматик G_1 и G_2 найти и вычеркнуть одно правило (альтернативу) так, чтобы G_1 и G_2 стали эквивалентными. Ответ обосновать.

G_1: $S \to aAa \mid aSaB$
$A \to Aa \mid BA \mid BS$
$B \to bC$
$C \to \varepsilon$

G_2: $S \to aAa \mid aB \mid aS$
$Y \to Ya \mid b \mid YS$
$B \to bA \mid bBC \mid \varepsilon$
$C \to bAY$

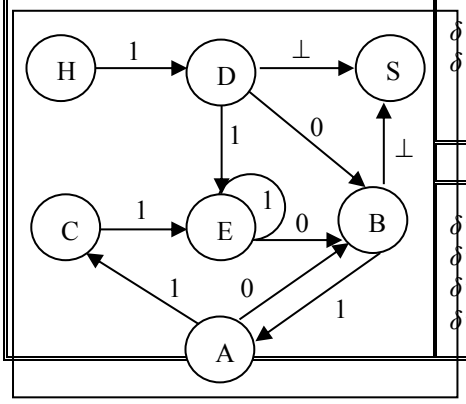
Ответ: в грамматике G_2 нужно вычеркнуть правило $S \to aB$ или правило $B \to \varepsilon$. Тогда $L(G_1)=L(G_2)=\varnothing$.

КРИТЕРИИ: не вычеркнуто ни одного правила, вычеркнуто больше одного, или вычеркнуто не то правило: **–10**. За отсутствие доказательства эквивалентности полученных грамматик: **–5**.

3. а) Построить по заданной грамматике G конечный автомат (в виде ДС).
б) Если автомат оказался недетерминированным, преобразовать его в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА,
в) По ДКА построить леволинейную грамматику.

G: $H \to 1A \mid 1B$
$A \to 0B \mid 1C$
$B \to 1A \mid \perp$
$C \to 1A \mid 1C$

Ответ: Автомат недетерминирован:



Функция переходов исходного автомата

$\delta(H, 1) = A$ $\delta(H, 1) = B$	$\delta(A, 0) = B$ $\delta(A, 1) = C$ $\delta(B, 1) = A$ $\delta(B, \perp) = S$	$\delta(C, 1) = A$ $\delta(C, 1) = C$

Функция переходов детерминированного автомата (D ≡ AB, E ≡ AC)

$\delta'(H, 1) = D$ $\delta'(D, 0) = B$ $\delta'(D, 1) = E$ $\delta'(D, \perp) = S$	$\delta'(E, 0) = B$ $\delta'(E, 1) = E$ $\delta'(B, 1) = A$ $\delta'(B, \perp) = S$	$\delta'(A, 0) = B$ $\delta'(A, 1) = C$ $\delta'(C, 1) = E$

G': $S \to B\perp \mid D\perp$
$A \to B1$
$B \to A0 \mid D0 \mid E0$
$C \to A1$
$D \to 1$
$E \to C1 \mid D1 \mid E1$

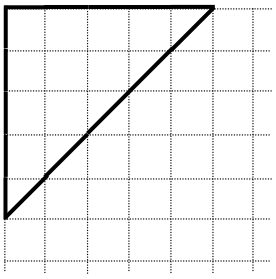
КРИТЕРИИ: (**а**) отсутствует или неэквивалентная ДС: **–4**
(**б**) отсутствует или построен неэквивалентный ДКА: **–8**
(**в**) отсутствует или неэквивалентная леволинейная грамматика: –4

4. Имеется клетчатый лист бумаги, бесконечный вправо и вниз. В левом верхнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*a*, *b, c*}:

a — переместить перо на одну клетку вправо;

b — переместить перо по диагонали на одну клетку влево-вниз;

c — переместить перо на одну клетку вверх.



Например, по последовательности *aaaaabbbbbccccc* интерпретатор построит прямоугольный равнобедренный треугольник с боковой стороной длины 5 (см. рисунок).

Построить грамматику с терминальным алфавитом {*a*, *b, c*}, описывающую подобные изображенному на рисунке треугольники с боковой стороной длины *n*, $n \geq 1$. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

Ответ: Треугольник с боковой стороной длины *n* описывается цепочкой $a^n b^n c^n$. Следовательно, требуется построить грамматику, порождающую язык $\{ a^n b^n c^n \mid n \geq 1\}$: $S \to \quad aSBc \mid abc$

$cB \to \quad Bc$
$bB \to \quad bb$

КРИТЕРИИ: порождается пустая цепочка: **–5**, имеется лишняя (непустая) или недостающая цепочка: **–10**, за каждое лишнее правило: **–4**. Если верно указан язык, но грамматика составлена неверно или имеет слишком много правил (превышение на три и более правила), за задачу ставить **2** балла.

5. Дана однозначная КС-грамматика $G_{balance}$, порождающая язык *L*, состоящий из цепочек, в которых символов *a* и *b* поровну:

$L=\{ x \in \{a,b\}^* \mid |x|_a = |x|_b \}$. Вставить в грамматику действия вида ⟨*cout* << "символы" ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(x, (ab)^{f(x)}) \mid x \in L$, $f(x)= |x|_a + |x|_b \}$.

$G_{balance}$:

$S \to aAbS \mid bBaS \mid \varepsilon$

$A \to aAbA \mid \varepsilon$

$B \to bBaB \mid \varepsilon$

Ответ: Соглашение: запись ⟨*a*⟩ является сокращением ⟨*cout* << *"a"*;⟩
При распознавании каждого символа *a* или *b* нужно печатать *ab*.

$S \to a\langle ab\rangle\, Ab\langle ab\rangle\, S \mid b\langle ab\rangle Ba\langle ab\rangle S \mid \varepsilon$
$A \to a\langle ab\rangle Ab\langle ab\rangle A \mid \varepsilon$
$B \to b\langle ab\rangle Ba\langle ab\rangle B \mid \varepsilon$

Другой вариант ответа – печатать $(ab)^2$ при распознавании «ведущего» *a* или *b*.

$S \to a\langle abab\rangle\, Ab\, S \mid b\langle abab\rangle BaS \mid \varepsilon$
$A \to a\langle abab\rangle AbA \mid \varepsilon$
$B \to b\langle abab\rangle BaB \mid \varepsilon$

Возможны другие варианты.

КРИТЕРИИ: Перевод реализован без ошибок: **10** баллов. Есть ошибки: **0** баллов

6. Какие стратегии трансляции могут выбираться в различных системах программирования? Объяснить существенные отличия этих подходов к реализации процесса трансляции с языка программирования. Приведите по одному примеру формального языка, к которому применена каждая из названных вами стратегий трансляции.

Ответ: (1) Трансляция может вестись на основе перевода одного языка программирования в другой – компиляция. (2) Трансляция может вестись с применением интерпретатора языка программирования. Такой подход существенно отличается от первого тем, что результирующая программа не создается. В результате обработки программы на исходном языке сразу вычисляется результат, алгоритм получения которого закодирован в исходной программе. (3) Возможен подход к трансляции, сочетающий в себе два предыдущих варианта (смешанная стратегия трансляции), когда сначала создается промежуточное представление исходной программы (подход «компиляция»), а затем это представление интерпретируется (подход «интерпретация»). (4) Паскаль, ассемблеры; ПЛЭНЕР, скриптовые языки; Java, ...

КРИТЕРИИ: За пропуск описания одного из двух основных подходов к трансляции снижать на **4** балла. За пропуск описания смешанной стратегии снижать на **2** балла. За пропуск или неверный ответ на последний вопрос **-1**.

7. В чём различие статических и динамических библиотек? Какие компоненты вычислительной системы занимаются подключением к программам каждого типа библиотек?

Ответ: Статические и динамические библиотеки различаются по выбору момента времени извлечения их составных частей при формировании полной программы. Статические компоненты извлекаются компоновщиками (редакторами связей) в момент редактирования связей до начала исполнения программы, динамические – при выполнении программы (загрузчиками).

КРИТЕРИИ: За отсутствие или неправильное объяснение разницы концепций **-6**. За пропуск любого из компонентов: **-3** (за каждый пропуск из двух).

8. По заданной грамматике $G = \{\{+, -\}, \{R, S\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

P: $S \to +R- \mid \varepsilon$
$R \to +R \mid -S \mid S$

Ответ: $S' \to S \mid \varepsilon$
$S \to +R- \mid +-$
$R \to +R \mid -S \mid S \mid + \mid -$

КРИТЕРИИ: За любую ошибку в ответе ставить **0** за всю задачу.

9. Дана грамматика G. Примени́м ли к ней метод рекурсивного спуска? Ответ обосновать.

G: $S \to aSb \mid Yb \mid bb$
$Y \to cSY \mid dd \mid \varepsilon$

Ответ: неприменим, так как $first\,(bb) \cap first\,(Yb) = \{b\} \neq \varnothing$.
КРИТЕРИИ: Нет обоснования или неверный ответ: **-10**. За каждую ошибку в обосновании: **-4**

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&a	#+	4	+	b	<=	36	!F	&b	+#	a	b	—	<	34	!F	&a

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
b	8	—	7	*	b	a	@	23	/	/	—	=	;	9	!	39	!	&b	1	=	;

Ответ:

```
        if (! (a ++ + 4 <= b))      goto N;
L:      if (! (++ b < a – b))       goto M;
        a = (b – 8) * 7 – b / (– a / 23);
                                    goto L;
M:                                  goto E;
N:      b = 1;
E:
```

```
if (a ++ + 4 <= b) while (++ b < a – b) a = (b – 8) * 7 – b / (– a / 23); else b = 1;
```

КРИТЕРИИ: за неверный ответ на первый вопрос: **-7**.
за неверный ответ на второй вопрос, если первый вариант верен: **-3**, иначе: **-10**

Вариант 2_2012 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to aS \mid aSb \mid aAb \mid ab$
$aAb \to aAbb \mid ab$
$aaAbb \to aaAbb$
$aS \to aaS$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

Класс П	$G \in$ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

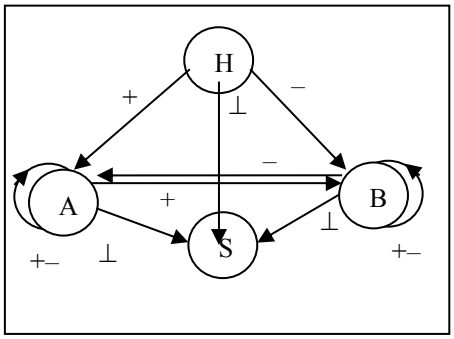
(в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком k+1.

2. Среди правил грамматик G_1 и G_2 найти и вычеркнуть одно правило (альтернативу) так, чтобы G_1 и G_2 стали эквивалентными. Ответ обосновать.

G_1: $S \to +S+Q \mid +R+$
$P \to \varepsilon$
$R \to QR \mid QS \mid R+$
$Q \to -P$

G_2: $S \to +V+ \mid +W \mid +S$
$Y \to Y+ \mid - \mid YS$
$W \to -V \mid -W-VY \mid \varepsilon$

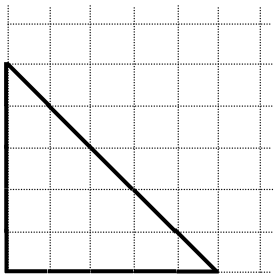
3. Дана ДС конечного автомата.
а) Построить по ней соответствующую леволинейную грамматику.
б) Если заданный автомат недетерминирован, преобразовать его в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА.
в) По ДКА построить праволинейную грамматику.



4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*a*, *d*, *q*}: *a* — переместить перо на одну клетку вправо;

d — переместить перо по диагонали на одну клетку влево-вверх;

q — переместить перо на одну клетку вниз.



Например, по последовательности *aaaaaddddqqqqq* интерпретатор построит прямоугольный равнобедренный треугольник с боковой стороной длины 5 (см. рисунок).
Построить грамматику с терминальным алфавитом {*a*, *d*, *q*}, описывающую подобные изображенному на рисунке треугольники с боковой стороной длины *n*, *n* ≥1.
В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

5. Дана однозначная КС-грамматика $G_{parenthesis}$, порождающая язык *L* сбалансированных скобочных систем. *Неделимой* называется скобочная система, которую нельзя представить в виде конкатенации двух других скобочных систем. Например, цепочка (( )( )) – неделимая скобочная система. *Протяжением* скобочной системы называется число неделимых систем, конкатенация которых дает данную систему. Например, цепочка (( )( )) имеет протяжение 1, а цепочка (( )( )( ))( ) имеет протяжение 2. Пустая цепочка имеет протяжение 0. Вставить в грамматику действия вида ⟨*cout* << 'символ' ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(x, 1^{f(x)}) \mid x \in L$, f (*x*) – протяжение цепочки *x* }.

$G_{parenthesis}$:

$S \to (A) S \mid \varepsilon$

$A \to (A) A \mid \varepsilon$

6. Дать определение и пояснить понятие «прохода компилятора». Как это понятие связано с понятием «фазы компиляции»?

7. Что является объектом оптимизационных преобразований в компиляторах? Укажите не менее трех видов машинно-независимых оптимизационных преобразований и приведите примеры каждого из них.

8. По заданной грамматике *G = {{0, 1}, {A, S}, P, S}* получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

P: $S \to 0A1 \mid A$
$A \to 0A \mid 1A \mid \varepsilon$

9. Дана грамматика *G*. Примени́м ли к ней метод рекурсивного спуска? Ответ обосновать.

G: $S \to dSbS \mid Y$
$Y \to cSY \mid ad \mid \varepsilon$

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	a	33	!F	&a	&a	b	8	—	7	*	b	a	@	23	/	/	—	=	+=

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
;	&b	#+	a	b	—	<	31	!F	4	!	40	!	&b	&a	&b	+#	/=	=	;	

Вариант 2_2012 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to aS \mid aSb \mid aAb \mid ab$
$aAb \to aAbb \mid ab$
$aaAbb \to aaAbb$
$aS \to aaS$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).
Ответ: $L(G) = \{a^n b^k \mid n, k \geq 1\}$

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

Класс П	$G \in$ П? (да/нет)
регулярные	**нет (-3)**
контекстно-зависимые	**нет (-1)**
контекстно-свободные	**нет (-3)**
грамматики типа 0	**да (-3)**
неукорачивающие	**нет (-1)**

(в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком k+1.

Ответ: k = **3** (для заданного языка существует порождающая его регулярная грамматика)

КРИТЕРИИ: (а) описание отсутствует или с ошибками: **–4**
(б) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице (обоснование не обязательно)
(в) нет ответа; неверный ответ: **–4** (обоснование не обязательно)

2. Среди правил грамматик G_1 и G_2 найти и вычеркнуть одно правило (альтернативу) так, чтобы G_1 и G_2 стали эквивалентными. Ответ обосновать.

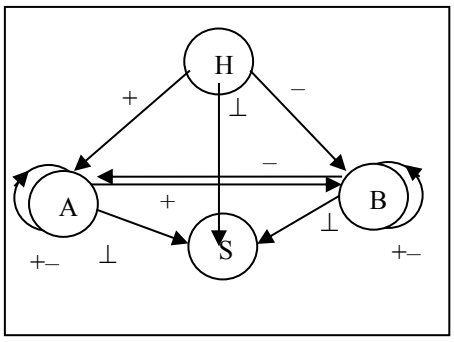
G_1: $S \to +S+Q \mid +R+$
$P \to \varepsilon$
$R \to QR \mid QS \mid R+$
$Q \to -P$

G_2: $S \to +V+ \mid +W \mid +S$
$Y \to Y+ \mid - \mid YS$
$W \to -V \mid -W-VY \mid \varepsilon$

Ответ: в грамматике G_2 нужно вычеркнуть правило $S \to +W$ или $W \to \varepsilon$. Тогда $L(G_1)=L(G_2)=\varnothing$.

КРИТЕРИИ: не вычеркнуто ни одного правила, вычеркнуто больше одного, или вычеркнуто не то правило: **–10**. За отсутствие доказательства эквивалентности полученных грамматик: **–5**.

3. Дана ДС конечного автомата.
а) Построить по ней соответствующую леволинейную грамматику.
б) Если заданный автомат недетерминирован, преобразовать его в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА.
в) По ДКА построить праволинейную грамматику.



Ответ: *Гл:* $S \to A\perp \mid B\perp \mid \perp$
$A \to A+ \mid B- \mid A- \mid +$
$B \to B+ \mid B- \mid A+ \mid -$ Автомат недетерминирован

Функция переходов исходного автомата

$\delta\ (H, +) = A$ $\delta\ (H, –) = B$ $\delta\ (H, \perp) = S$	$\delta\ (A, +) = A$ $\delta\ (A, +) = B$ $\delta\ (A, –) = A$ $\delta\ (A, \perp) = S$	$\delta\ (B, +) = B$ $\delta\ (B, –) = B$ $\delta\ (B, –) = A$ $\delta\ (B, \perp) = S$

Функция переходов детерминированного автомата (C ≡ AB)

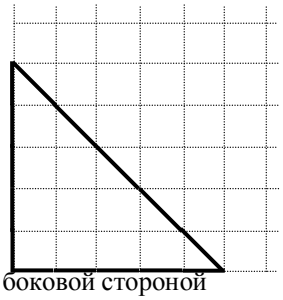
$\delta'\ (H, +) = A$ $\delta'\ (H, –) = B$ $\delta'\ (H, \perp) = S$	$\delta'\ (A, +) = C$ $\delta'\ (A, –) = A$ $\delta'\ (A, \perp) = S$	$\delta'\ (B, +) = B$ $\delta'\ (B, –) = C$ $\delta'\ (B, \perp) = S$	$\delta'(C, +) = C$ $\delta'(C, –) = C$ $\delta'\ (C, \perp) = S$

Glев: $S \to A\perp \mid B\perp \mid C\perp \mid \perp$
$A \to A– \mid +$
$B \to B+ \mid –$
$C \to A+ \mid B– \mid C+ \mid C–$

Gправ: $H \to +A \mid –B \mid \perp$
$A \to –A \mid +C \mid \perp$
$B \to +B \mid –C \mid \perp$
$C \to +C \mid –C \mid \perp$

КРИТЕРИИ: (**а**) отсутствует или неправильная леволинейная грамматика: **–4**
(**б**) отсутствует или построен неэквивалентный ДКА: **–8**
(**в**) отсутствует или неэквивалентная праволинейная грамматика: **–4**

4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*a*, *d, q*}: *a* — переместить перо на одну клетку вправо;



d — переместить перо по диагонали на одну клетку влево-вверх;

q — переместить перо на одну клетку вниз.

Например, по последовательности *aaaaaddddddqqqqq* интерпретатор построит прямоугольный равнобедренный треугольник с боковой стороной длины 5 (см. рисунок).

Построить грамматику с терминальным алфавитом {*a*, *d, q*}, описывающую подобные изображенному на рисунке треугольники с боковой стороной длины *n*, $n \geq 1$.

В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).**Ответ:** Треугольник с боковой стороной длины *n* описывается цепочкой $a^n d^n q^n$. Следовательно, требуется построить грамматику, порождающую язык $\{ a^n d^n q^n \mid n \geq 1\}$: $S \to aSDc \mid adq$

$qD \to Dq$
$dD \to dd$

КРИТЕРИИ: порождается пустая цепочка: **–5**, имеется лишняя (непустая) или недостающая цепочка: **–10**, за каждое лишнее правило: **–4**. Если верно указан язык, но грамматика составлена неверно или имеет слишком много правил (превышение на три и более правила), за задачу ставить **2** балла.

5. Дана однозначная КС-грамматика $G_{parenthesis}$, порождающая язык *L* сбалансированных скобочных систем. *Неделимой* называется скобочная система, которую нельзя представить в виде конкатенации двух других скобочных систем. Например, цепочка (( )( )) – неделимая скобочная система. *Протяжением* скобочной системы называется число неделимых систем, конкатенация которых дает данную систему. Например, цепочка (( )( )) имеет протяжение 1, а цепочка (( )( )( ))( ) имеет протяжение 2. Пустая цепочка имеет протяжение 0. Вставить в грамматику действия вида ⟨*cout* << '`символ`' ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(x, 1^{f(x)}) \mid x \in L$, f (*x*) – протяжение цепочки *x* }.

$G_{parenthesis}$:
$S \to (\ A\ )\ S \mid \varepsilon$
$A \to (\ A\ )\ A \mid \varepsilon$

Ответ: Соглашение : запись ⟨*1*⟩ является сокращением ⟨*cout* << *'1'*;⟩
$S \to (\ A\ )\ \langle 1\rangle\ S \mid \varepsilon$
$A \to (\ A\ )\ A \mid \varepsilon$

КРИТЕРИИ: Перевод реализован без ошибок: **10** баллов. Есть ошибки: **0** баллов

6. Дать определение и пояснить понятие «прохода компилятора». Как это понятие связано с понятием «фазы компиляции»?

Ответ: Проход – это процесс последовательного чтения компилятором данных из внешней памяти, их обработки и записи результата во внешнюю память. Фаза компиляции – смысловая часть процесса компиляции, на которой происходит тот или иной процесс обработки: анализа+преобразования или синтеза. На одном проходе может выполняться сразу две фазы компиляции, но каждая фаза компиляции может выполняться за несколько проходов.

КРИТЕРИИ: За отсутствие определения прохода: **-5**. Неправильная трактовка фазы компиляции: **-5**.

7. Что является объектом оптимизационных преобразований в компиляторах? Укажите не менее трех видов машинно-независимых оптимизационных преобразований и приведите примеры каждого из них.

Ответ: (1) Оптимизация в компиляторах проводится в отношении внутреннего представления исходной программы, (2) при оптимизации вновь формируется внутреннее представление программы, (3) к машинно-независимой оптимизации обычно относят (а) оптимизацию линейных участков, (б) оптимизацию циклов и (в) оптимизацию вызовов функций (процедур).

Примеры:

(а) Арифметические преобразования A=B*C+B*D => A=B*(C+D)
(б) Расщепление цикла:

```
for (i = 0; i < n; i ++)                          if (x < y)
    {   if (x < y)      { S1; }                         for (i = 0; i < n; i ++) { S1; }
        else            { S2; }     }         else      for (i = 0; i < n; i ++) { S2; }
```

(в) Оптимизация вызовов: прямая подстановка тела функции, передача параметров на регистрах

КРИТЕРИИ: (**а**) Неправильный ответ на первые два вопроса: **-3** (за каждый).
(**б**) За отсутствие какого-либо из видов оптимизации: **-2** (за каждый из трех пропусков).
(**в**) За отсутствие примеров: **-1** (за каждый).

8. По заданной грамматике $G = \{\{0, 1\}, \{A, S\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

P: $S \to 0A1 \mid A$
$A \to 0A \mid 1A \mid \varepsilon$

Ответ: $S' \to S \mid \varepsilon$
$S \to 0A1 \mid A \mid 01$
$A \to 0A \mid 1A \mid 0 \mid 1$

КРИТЕРИИ: За любую ошибку в ответе ставить **0** за всю задачу.

9. Дана грамматика G. Примени́м ли к ней метод рекурсивного спуска? Ответ обосновать.

G: $S \to dSbS \mid Y$
$Y \to cSY \mid ad \mid \varepsilon$

Ответ: примени́м, так как $first\ (dSbS) \cap first(Y) = \varnothing$, $first\ (cSY) \cap first\ (ad) = \varnothing$, $first\ (Y) \cap follow\ (Y) = \varnothing$.

КРИТЕРИИ: Нет обоснования или неверный ответ: **-10**. За каждую ошибку в обосновании: **-4**

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем

(если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	a	33	!F	&a	&a	b	8	—	7	*	b	a	@	23	/	/	—	=	+=

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
;	&b	#+	a	b	—	<	31	!F	4	!	40	!	&b	&a	&b	+#	/=	=	;	

Ответ:

```
        if (! a)                    goto N;
L:      a += a = (b – 8) * 7 – b / (– a / 23);
        if (! (b++ < a – b))        goto M;
                                    goto L;
M:                                  goto E;
N:      b = a /= ++b;
E:
```

*if (a) do a += a = (b – 8) * 7 – b / (– a / 23); while (++ b < a – b); else b = a /= ++ b;*

КРИТЕРИИ: за неверный ответ на первый вопрос: **-7**.
за неверный ответ на второй вопрос, если первый вариант верен: **-3**, иначе: **-10**

Вариант 3_2012 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to Sb|aSb|ab|aAb$
$aAb \to aDAb|aab$
$aD \to Da$
$D \to a$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

Класс П	$G \in$ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

(в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком k+1.

2. Среди правил грамматик G_1 и G_2 найти и вычеркнуть одно правило (альтернативу) так, чтобы G_1 и G_2 стали эквивалентными. Ответ обосновать.

G_1: $S \to 1T1 \mid 1S1U$
$B \to 0C \mid 0$
$C \to \varepsilon$
$T \to T1 \mid UT \mid US$

G_2: $S \to 1T1 \mid 10T \mid 10UC \mid U \mid 1U \mid 1S$
$C \to 0TZ$
$U \to 0T \mid 0UC \mid \varepsilon$
$Z \to Z1 \mid 0 \mid ZS$

3. а) Построить по заданной грамматике G конечный автомат (в виде ДС).

б) Если автомат оказался недетерминированным, преобразовать его в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА.

в) По ДКА построить праволинейную грамматику.

G: $S \to A\perp \mid B\perp \mid \perp$
$A \to Aa \mid Ba \mid Bb \mid a$
$B \to Bb \mid Aa \mid Ab \mid b$

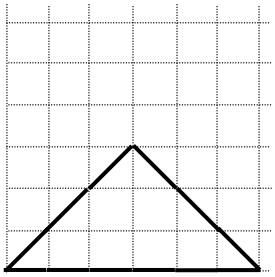
4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*a*, *b, d*}:

a — переместить перо на одну клетку вправо;

d — переместить перо по диагонали на одну клетку влево-вверх;

b — переместить перо по диагонали на одну клетку влево-вниз.

Например, по последовательности *aaaaaadddbbb* интерпретатор построит прямоугольный равнобедренный треугольник с основанием длины 6 (см. рисунок).



Построить грамматику с терминальным алфавитом {*a*, *b, d*}, описывающую подобные изображенному на рисунке треугольники с основанием длины $2k$, $k \geq 1$. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

5. Дана однозначная КС-грамматика $G_{parenthesis}$, порождающая язык L сбалансированных скобочных систем. *Неделимой* называется скобочная система, которую нельзя представить в виде конкатенации двух других скобочных систем. Например, цепочка (( )( )) – неделимая скобочная система. Вставить в грамматику действия вида ⟨*cout* << 'символ' ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(x, 1^{f(x)}) \mid x \in L$, f(x) – количество неделимых скобочных подсистем в x. Например, для x = (( )( )( ))( ) f(x) = 5 }.

$G_{parenthesis}$:
$S \to T\,S \mid \varepsilon$
$T \to (\,S\,)$

6. От каких компонентов интегрированных систем программирования получают и каким компонентам передают данные для обработки входящие в них редакторы связей? Что подвергается обработке в редакторах связей и с чем связана необходимость этой обработки?

7. Какие области памяти выделяются компиляторами для хранения локальных данных процедур? Какая дисциплина распределения памяти при этом реализуется? Какой способ использования выбирается для этих областей? Какая ещё информация должна размещаться в тех же областях памяти, с той же дисциплиной распределения и с тем же способом использования?

8. По заданной грамматике $G = \{\{a, b, c\}, \{S, T, U\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

P: $S \to aTb \mid UT \mid cS$
$T \to aT \mid bU \mid \varepsilon$
$U \to bT \mid aU \mid \varepsilon$

9. Дана грамматика G. Примени́м ли к ней метод рекурсивного спуска? Ответ обосновать.

G: $S \to aSb \mid Yb \mid bb$
$Y \to cSY \mid B \mid \varepsilon$
$B \to d \mid \varepsilon$

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	23	!F	&a	&a	b	8	—	7	*	b	a	@	23	/	/	—

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
=	+=	;	28	!	&a	&b	+#	=	;	&b	#+	a	b	—	<	38	!F	1	!	

Вариант 3_2012 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to Sb|aSb|ab|aAb$
$aAb \to aDAb|aab$
$aD \to Da$
$D \to a$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы или исчерпывающего словесного описания (не более 300 печатных знаков включая пробелы).
Ответ: $L(G) = \{a^n b^k \mid n, k \geq 1\}$

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

Класс П	$G \in$ П? (да/нет)
регулярные	**нет (-3)**
контекстно-зависимые	**нет (-1)**
контекстно-свободные	**нет (-3)**
грамматики типа 0	**да (-3)**
неукорачивающие	**да (-1)**

(в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком k+1.

Ответ: k = **3** (для заданного языка существует порождающая его регулярная грамматика)

КРИТЕРИИ: (а) описание отсутствует или с ошибками: **–4**
(б) снимаемые баллы за каждый неверный или отсутствующий ответ в таблице (обоснование не обязательно)
(в) нет ответа; неверный ответ: **–4** (обоснование не обязательно)

2. Среди правил грамматик G_1 и G_2 найти и вычеркнуть одно правило (альтернативу) так, чтобы G_1 и G_2 стали эквивалентными. Ответ обосновать.

G_1: $S \to 1T1 \mid 1S1U$
$B \to 0C \mid 0$
$C \to \varepsilon$
$T \to T1 \mid UT \mid US$

G_2: $S \to 1T1 \mid 10T \mid 10UC \mid U \mid 1U \mid 1S$
$C \to 0TZ$
$U \to 0T \mid 0UC \mid \varepsilon$
$Z \to Z1 \mid 0 \mid ZS$

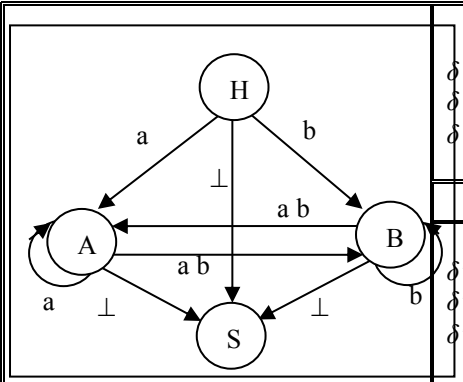
Ответ: в грамматике G_2 нужно вычеркнуть правило $U \to \varepsilon$. Тогда $L(G_1)=L(G_2)=\varnothing$.

КРИТЕРИИ: не вычеркнуто ни одного правила, вычеркнуто больше одного, или вычеркнуто не то правило: **–10**. За отсутствие доказательства эквивалентности полученных грамматик: **–5**.

3. а) Построить по заданной грамматике G конечный автомат (в виде ДС).
б) Если автомат оказался недетерминированным, преобразовать его в соответствии с алгоритмом преобразования НКА в эквивалентный ДКА,
в) По ДКА построить праволинейную грамматику.

G: $S \to A\perp \mid B\perp \mid \perp$
$A \to Aa \mid Ba \mid Bb \mid a$
$B \to Bb \mid Aa \mid Ab \mid b$

Ответ: Автомат недетерминирован: Функция переходов исходного автомата



$\delta\,(H, a) = A$ $\delta\,(H, b) = B$ $\delta\,(H, \perp) = S$	$\delta\,(A, a) = A$ $\delta\,(A, a) = B$ $\delta\,(A, b) = B$ $\delta\,(A, \perp) = S$	$\delta\,(B, a) = A$ $\delta\,(B, b) = A$ $\delta\,(B, b) = B$ $\delta\,(B, \perp) = S$
Функция переходов детерминированного автомата (C ≡ AB)		
$\delta'\,(H, a) = A$ $\delta'\,(H, b) = B$ $\delta'\,(H, \perp) = S$	$\delta'\,(A, a) = C$ $\delta'\,(A, b) = B$ $\delta'\,(A, \perp) = S$ $\delta'\,(B, a) = A$ $\delta'\,(B, b) = C$ $\delta'\,(B, \perp) = S$	$\delta'\,(C, a) = C$ $\delta'\,(C, b) = C$ $\delta'\,(C, \perp) = S$

G': $H \to aB \mid bA \mid \perp$
$A \to aC \mid bB \mid \perp$
$B \to aA \mid bC \mid \perp$
$C \to aC \mid bC \mid \perp$

КРИТЕРИИ: (**а**) отсутствует или неэквивалентная ДС: **–4**
(**б**) отсутствует или построен неэквивалентный ДКА: **–8**
(**в**) отсутствует или неэквивалентная праволинейная грамматика: –4

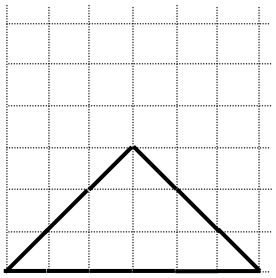
4. Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*a*, *b*, *d*}:

a — переместить перо на одну клетку вправо;

d — переместить перо по диагонали на одну клетку влево-вверх;

b — переместить перо по диагонали на одну клетку влево-вниз.

Например, по последовательности *aaaaaadddbbb* интерпретатор построит прямоугольный равнобедренный треугольник с основанием длины 6 (см. рисунок).



Построить грамматику с терминальным алфавитом {*a*, *b*, *d*}, описывающую подобные изображенному на рисунке треугольники с основанием длины $2k$, $k \geq 1$. В грамматике должно быть **не более** 4 правил вывода (считая альтернативы).

Ответ: Треугольник с основанием длины *2n* описывается цепочкой $a^{2n}d^nb^n$. Следовательно, требуется построить грамматику, порождающую язык $\{ a^{2n}d^nb^n \mid n \geq 1\}$:

$S \to aaSDb \mid aadb$
$bD \to Db$
$dD \to dd$

КРИТЕРИИ: порождается пустая цепочка: **–5**, имеется лишняя (непустая) или недостающая цепочка: **–10**, за каждое лишнее правило: **–4**. Если верно указан язык, но грамматика составлена неверно или имеет слишком много правил (превышение на три и более правила), за задачу ставить **2** балла.

5. Дана однозначная КС-грамматика $G_{parenthesis}$, порождающая язык *L* сбалансированных скобочных систем. *Неделимой* называется скобочная система, которую нельзя представить в виде конкатенации двух других скобочных систем. Например, цепочка (( )( )) – неделимая скобочная система. Вставить в грамматику действия вида ⟨*cout* << 'символ' ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(x, 1^{f(x)}) \mid x \in L$, f(*x*) – количество неделимых скобочных подсистем в *x*. Например, для *x* = (( )( )( ))( ) f(*x*) = 5 }.

$G_{parenthesis}$:
$S \to TS \mid \varepsilon$
$T \to (S)$

Ответ: Соглашение: запись ⟨*1*⟩ является сокращением ⟨*cout* << *'1'*;⟩

$S \to T\langle 1\rangle S \mid \varepsilon$
$T \to (S)$

$S \to TS \mid \varepsilon$
$T \to (S)\langle 1\rangle$

$S \to TS \mid \varepsilon$
$T \to (\langle 1\rangle S)$

КРИТЕРИИ: Перевод реализован без ошибок: **10** баллов. Есть ошибки: **0** баллов.

6. От каких компонентов интегрированных систем программирования получают и каким компонентам передают данные для обработки входящие в них редакторы связей? Что подвергается обработке в редакторах связей и с чем связана необходимость этой обработки?

Ответ: Редакторы связей получают свои исходные данные от компиляторов и ассемблеров, а также из библиотек. Редакторы связей передают основные результаты своей работы загрузчикам. Таблицы, строящиеся при редактировании связей, передаются отладчикам. Редакторами связей обрабатываются именованные ссылки из одних модулей в другие, например, ссылки на внешние процедуры или глобальные объекты, определенные в других модулях. Компилятор (и ассемблер) не в состоянии обрабатывать такие ссылки, так как обычно используется стратегия раздельной трансляции модулей программы.

КРИТЕРИИ: За пропуск компонентов: **-3** (за компилятор), **-1** (за ассемблер), **-3** (за библиотеки), **-3** (за загрузчик), **-2** (за отладчик). За отсутствие или неправильное описание обрабатываемой информации и/или причины необходимости редактирования связей снижать на **5**.

7. Какие области памяти выделяются компиляторами для хранения локальных данных процедур? Какая дисциплина распределения памяти при этом реализуется? Какой способ использования выбирается для этих областей? Какая ещё информация должна размещаться в тех же областях памяти, с той же дисциплиной распределения и с тем же способом использования?

Ответ: Для хранения локальных данных процедур используется (1) динамическая память (2) со стековой дисциплиной распределения, (3) относящаяся к локальным областям памяти. (4) В тех же областях памяти, с той же дисциплиной распределения и с тем же способом использования хранятся записи о текущем состоянии процесса выполнения программ, а также записи о входах в блоки операторов, которые можно рассматривать в качестве процедур без параметров.

КРИТЕРИИ: (**а**) Неправильный ответ на первые три вопроса: **-2** (за каждый).
(**б**) За пропуск ответа о записях состояния процесса выполнения: **-3**.
(**в**) За пропуск ответа о входах в блоки: **-1**.

8. По заданной грамматике $G = \{\{a, b, c\}, \{S, T, U\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

P: $S \to aTb \mid UT \mid cS$
$T \to aT \mid bU \mid \varepsilon$
$U \to bT \mid aU \mid \varepsilon$

Ответ: $S' \to S \mid \varepsilon$
$S \to aTb \mid ab \mid UT \mid T \mid U \mid cS \mid c$
$T \to aT \mid bU \mid a \mid b$
$U \to bT \mid aU \mid a \mid b$

КРИТЕРИИ: За любую ошибку в ответе ставить **0** за всю задачу.

9. Дана грамматика G. Примени́м ли к ней метод рекурсивного спуска? Ответ обосновать.

G: $S \to aSb \mid Yb \mid bb$
$Y \to cSY \mid B \mid \varepsilon$
$B \to d \mid \varepsilon$

Ответ: неприменим, так у нетерминала Y есть две альтернативы, из которых выводится ε.

КРИТЕРИИ: Нет обоснования или неверный ответ: **-10**. За каждую ошибку в обосновании: **-4**

10. По приведенной обратной польской записи фрагмента программы, написанной на языке Си++, восстановите этот фрагмент двумя способами: сначала, пользуясь операторами условного и безусловного перехода на метку, а затем (если это возможно), пользуясь только операторами структурного программирования без переходов. Операция ПОЛИЗ '@' соответствует унарной операции изменения знака.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	a	23	!F	&a	&a	b	8	—	7	*	b	a	@	23	/	/	—

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
=	+=	;	28	!	&a	&b	+#	=	;	&b	#+	a	b	—	<	38	!F	1	!	

Ответ:

L: *if (! a)* *goto M;*
*a += a = (b – 8) * 7 – b / (– a / 23);*
goto N;
M: *a = ++b;*
N: *if (! (b++ < a – b))* *goto E;*
goto L;
E:
*do if (a) a += a = (b – 8) * 7 – b / (– a / 23); else a = ++ b; while (++ b < a – b);*

КРИТЕРИИ: за неверный ответ на первый вопрос: **-7**.
за неверный ответ на второй вопрос, если первый вариант верен: **-3**, иначе: **-10**

2013 год
Коллоквиум

Вариант 1_2013 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. Для **каждого имени** из реализации функций `B::g()` и `main()` укажите (используя операцию "`::`") область видимости имён так, чтобы программа была верной. **Подчеркните** вставленные расширения имён, без которых программа ошибочна. Если вариантов расширения имён несколько, укажите все.

```
static int x;
namespace N1 { int f (int a, int b){ return   x = a + b; }
               class A { int x;
                   public: A (int n = 13) { x = n; }
                           int f(){ return ::x = x; }
               };
               namespace N2 { class B:  public  A { float x;
                                  public: int f (int a) { return ::x = x; }
                                          void g ();
                                };
               }
}
void        B::g() {        x = 20.13;      f ();
                            f (2013);       f (13, 20132013);
}
int main () {           B b; b.       f ();  b.       f (13);
                        x =       f ('2', 013);
                        return 0;
}
```

2. Есть ли ошибки в функции `main()` приведённого фрагмента на Си++? Если есть, то **объясните,** в чём они заключаются. Если нужно, удалите ошибочные вызовы из функции `main()`. Что будет выдано в стандартный поток вывода при работе исправленной (если нужно) программы?

```
class A { public: virtual void f()      { h (65); cout << "A::f"       << endl; }
                  virtual void g(int x) { h (x);  cout << "A::g," << x << endl; }
                           void h(int x) {        cout << "A::h," << x << endl; }
};
class B:  public A { public:
                           void g (int x) { h (x);  cout << "B::g," << x << endl; }
                  virtual void h (int x) {         cout << "B::h," << x << endl; }
};
class C:  public B { float f;
                  public: void g ()      { h (67); cout << "C::g"       << endl; }
                  virtual void h (int x) {         cout << "C::h," << x << endl; }
};
int main( ){
     C c, * pc = & c;   B * pb = & c;   A * pa = & c;
     pa -> f ();                        pa -> f (2013);
     pa -> g (13);                      pb -> f ();
     pb -> g ();                        pb -> g (13);
     pb -> h (0);                       pc -> f ();
     pc -> g ();                        pc -> h ();
     pc -> h (0);                       return 1;
}
```

3. Что такое функция преобразования в языке Си++? Какие ограничения накладываются на использование функций преобразования при выборе одной из перегруженных функций? Привести пример использования функции преобразования в программах на Си++.

4. Используя описания классов A и B из задания **1**, определить, тела каких конструкторов и деструкторов (возможно явно не определённые) и в каком порядке будут исполнены при работе следующего фрагмента программы, расположенного в пространстве имён *N2* из задания **1**:

```
void h () { struct C { A a; B b; }; A a(2013); C c; }
```

5. Для класса рациональных дробей

```
template<class T> class fraction { T n; T d; ... };
```

написать два варианта реализации вне класса операции `"/"`, выполняющей деление одной рациональной дроби на другую, а именно – методом класса и функцией-другом этого класса.

6. Описать алгоритм определения возможности использования имён из базовых классов в производном классе в случае множественного наследования. Привести по одному примеру, демонстрирующему случаи обнаружения ошибки на каждом из шагов алгоритма.

7. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
struct X; void f(X & x, int n);
struct X { X (){  cout << "Xc";
                  f (*this, -1);  cout << "Xf"; }
           X (X &) { cout << "Xk"; }
};
struct Y: X { Y (){ try { cout << "Yt";  f (*this, 1);
                          cout << "Yp";
                        }
                        catch (Y)  { cout << "Yy";  throw; }
                        catch (int){ cout << "Yi";  throw; }
              }
              Y (Y &) { cout << "Yk"; }
};
void f (X & x, int n) { try { cout << "ft";
                              if (n < 0)     throw x;
                              cout << "f0";
                              if (n > 0)     throw 1;
                              cout << "f1";                              }
                        catch (int)   { cout << "fi";  throw; }
```

```
                              catch (X& a)  { cout << "fc";
                                              f(x, 1);
                                              cout << "fr";
                                              throw;                          }
}
int main() { try { Y a; } catch (...){ cout << "mc"; return 0; }
             cout << "mr";
             return 0;
}
```

8. В приведённой программе возможно наличие синтаксических ошибок в определении класса *A*. Если ошибки есть, исправьте их заменой, исключением или добавлением нужных служебных слов языка Си++. Обоснуйте сделанные исправления.

```
class A { int i;
          int f (int & x) { return g (x); }
          int g (int & x) { if (x >= 0) f (-i); return i; }  };
int A::i = 2013;
int main () { const A a;   A::i = 201;  a.f (20);  return a.i = 1; }
```

9. Для приведённой ниже программы описать функцию B* *f(A*);* , которая, получая в качестве параметра указатель типа *A**, возвращает его значение, наиболее безопасным образом преобразованное к типу *B**, а в случае невозможности преобразования корректно завершает работу программы.

```
struct A { };
struct B:A {int x; B (int y = 5){ x = y; } };
int main () { try { B b, *pb = f(&b); cout << pb -> x << '\n';
                    A a;  pb = f(&a); cout << pb -> x << '\n'; }
              catch (...){ }    return 0;
            }
```

10.Даны описания:

```
typedef vector<int> V;
struct Weight_t { V::size_type Index;  // индекс элемента вектора
                  float Weight;   };   // вес элемента вектора
typedef list <Weight_t>  L;
```

Написать функцию *g()*, которая по заданному вектору типа *V* и соответствующему ему списку типа *L* вычисляет средневзвешенное значение элементов вектора (средний результат умножения элементов на их веса), выдавая в выходной поток значения и веса элементов.

Ответы_Вариант 1_2013

1. **Ответ:**

```
void N1::N2::B::g() {
                        N1::N2::B::x = 20.13;
                        N1::A::f ();
                        N1::N2::B::f (2013);
                        N1::f (13, 20132013); }

 int main (){           N1::N2::B b;
                        b.N1::A::f ();
                        b.N1::N2::B::f (13);
                        ::x =N1::f ('2', 013);  }
```

Отмечены исправляющие (обязательные) изменения.

Критерии: За отсутствие каждого **необходимого (из 6)** расширения имени (без которого ошибка) **-2** .
За каждое другое не отмеченное уточнение области видимости (из 6) **-1**.
При указании хотя бы одного правильного расширения имени (где это необходимо) за отсутствие вариантов расширений не наказывать.
Для остальных расширений имен считать достаточным указание ближайшей области видимости, за отсутствие полной цепочки вложенных областей видимости – не наказывать.

2. Неверные вызовы: Будет выдано в поток: `A::h,65`

pc -> f()	(в классе *C* нет функции с именем *f*, поле *f* не доступно),	A::f
pa -> f(2013)	(в классе *A* нет функции *f()* с параметром),	C::h,13
pb -> g()	(в классе *B* нет функции *g()* без параметров).	B::g,13
pc -> h()	(в классе *C* нет функции *h()* без параметров)	A::h,65
		A::f
		C::h,13
		B::g,13
	C::h,0	
		C::h,67
		C::g
		C::h,0

Критерии: За каждую не найденную\лишнюю ошибку или неправильно вызванную функцию **– 1**.

3. Функция преобразования – метод класса специального вида (***operator*** <*mun*> *()*), которые преобразуют объект класса к типу <*mun*>, например, "***operator int()***". Алгоритм преобразования кодируется в теле функций преобразования.
1). Пользовательские преобразования автоматически применяются только, если они однозначны. 2). Допускается не более одного пользовательского преобразования для обработки одного параметра одного вызова.

```
struct S { long s;
           S(){ s = 0L; }                    // конструктор умолчания
           operator long (){ return s; } }; //   S --> long

long f(long l) { return l; }
```

Теперь разрешено: `S s; long nl = f (s);`

Критерии: За неправильное объяснение **–5**.
За каждое из двух неправильно описанных или пропущенных ограничений **–3**.
За отсутствие примера **–2**.

4. Ответ: `A(2013), A(), A(), B(), C(), ~C(), ~B(), ~A(), ~A(), ~A().`

Критерии: За каждую ошибку **–3**.
За указание `A()` вместо `A(2013)` снижать на **1** балл.

5.
```
template<class T> fraction<T>  fraction<T>::operator/ (const fraction<T> & x)
          { fraction<T> w;  w.n = n * x.d;     w.d = d * x.n;    return w; }
template<class T> fraction<T> operator/ (const fraction<T>& x, const fraction<T>& y)
          { fraction<T> w;  w.n = x.n * y.d;    w.d = x.d * y.n;    return w; }
```

Критерий: За отсутствие\ошибку каждого примера (за однотипные ошибки наказывать 1 раз): **-5.**

6. При исследовании возможности использования имени проводится (1) проверка однозначности, (2) выбор перегруженной функции, (3) контроль доступа.

```
class A { int g(int); public: double x;... };
class B {             public: double x; void f (short); void f (char); ... };
class C: public A, public B { public: ... };
C c; C * p = & c; p -> x = 1.0;   // неоднозначность наследования поля A::x
p -> f (1);                       // неоднозначность выбора функции f()
p -> g (2);                       // запрет доступа к функции A::g(int)
```

Критерий: За пропуск одного из трёх шагов: **-3**.
За указание их неправильного порядка: **-3**.
За отсутствие каждого из трёх примеров: **-1**.

7. **Ответ:** `Xc ft Xk fc ft f0 fi mc`

Критерии: За каждую неверную (лишнюю или пропущенную) печать **-2**.

8. Ответ:

1. Заменить слово *"class"* словом *"struct"*.
2. Перед определением типа поля *"i"* вставить слово *"static"*.
3. В определениях методов *"f()"* и *"g()"* вставить после пустого списка формальных параметров слово *"const"*.
4. В определениях методов *"f()"* и *"g()"* вставить перед типом формального параметра слово *"const"*.

Критерий: За пропуск ошибок доступа к закрытым членам класса: **-5**.
За пропуск указания статичности поля *"i"*: **-8.**
За пропуск каждой из двух ошибок вызова константных методов: **-4.**
За пропуск каждой из двух ошибок в заголовках функций: **-4.**
За указание ошибки там, где её нет: **-5.**

9. **Ответ:**

```
B* f(A* p){
   B* pb = dynamic_cast<A*>(p);
   if(pb)  return pb;
   else throw 0;/*exit(0);*/
}
```

Критерий: За утверждение о невозможности безопасного преобразования ставить полный балл: **10**.
За безошибочное решение с использованием динамического приведения (как здесь написано, как будто класс полиморфен): **-1**.
За любую другую ошибку: **-10** (включая использование статического приведения).

10. **Решение:**

```
float g (const V & vect, const L & lst)
{ L::const_iterator lp = lst.begin ();
  V::size_type t, n = 0; float vw;
  float w = 0.0f;
  while (lp != lst.end ())
     {      t = (* lp).Index;
            vw = (* lp).Weight; ++ lp;
            if (t < 0 || t >= vect.size ()) continue;
            cout << vect [t] << ' ' << vw << endl;
            w += vect [t] * vw;
            ++ n;
     }
  return n ? w / n : 0.0f;
}
```

Критерий: За каждую серьёзную ошибку: **-3**.
За использование обычного итератора вместо константного, отсутствие проверок на пустой список и выход за пределы диапазона индексов не наказывать.

Вариант 2_2013 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

NB! **Во всех задачах, где это требуется, предполагается наличие директив**
`#include <iostream>` и `using namespace std`.

1. Если в реализации функций *g()* и *main()* есть ошибки, исправьте их, используя только операцию расширения области видимости ***"::"***. Для всех обращений к функциям *f()* и *g()* укажите, из какой области видимости они вызываются (если вариантов расширения имён несколько, укажите все). Подчеркните те расширения имён, без которых программа ошибочна.

```
int x;
int f(){ return x = 2; }
struct A { int f() { return x = 0; } };

namespace S1 {
        class B: public A {
                int x;
         public: B(int n = 1){ x = n; }
                 short int f(){ return x = 3; }
                 int f(int a, int b) { return x = a + b; } };
        namespace S2 {
             class C: public B {
                   int x;
              public: int f(int a) { return a; }
                      int f(int a, int b) { return b; }
                      int g() { x =           f ();
                                x =           f (2);
                                x =           f (0, 1);  return 3; } };
        }
}
int main () {             C c;        x =         c.          f ();
                          x =         c.          f (2013);
               return     c.          g ();
}
```

2. Есть ли ошибки в функции *main()* приведённого фрагмента на Си++? Если есть, то **объясните,** в чём они заключаются. Если нужно, удалите ошибочные вызовы из функции *main()*. Что будет выдано в стандартный поток вывода при работе исправленной (если нужно) программы?

```
class T { virtual void p ()      { z (85); cout << "T::p"       << endl; }
          public: void u (int x) { z (x);  cout << "T::u," << x << endl; }
                  void z (int x) { z ();   cout << "T::z," << x << endl; }
                  void z ()      {         cout << "T::z"       << endl; }
};
class U: public T { public:
          virtual void p ()      { z (86); cout << "U::p"       << endl; }
                  void u (int x) { z (x);  cout << "U::u," << x << endl; }
          virtual void z (int x) {         cout << "U::z," << x << endl; }
};
class W: public U { public: unsigned u;
                  void p (int x) { z ();   cout << "W::p," << x << endl; }
                  void z ()      {         cout << "W::z"       << endl; }
};
int main( ){
     W w, * pw = &w;     T * pt = &w;       U * pu = &w;
     pt -> p ();         pt -> u (201);     pt -> z ();      pu -> p ();
     pu -> u (201);      pu -> z (20);      pw -> p ();      pw -> p (2013);
     pw -> u (201);      pw -> z ();        pw -> z (20);    return 2;     }
```

3. Что такое конструктор преобразования? Какие ограничения накладываются на использование конструкторов преобразования при выборе одной из перегруженных функций? Привести пример использования конструктора преобразования в программах на Си++.

4. Используя описания классов A, B и C из задания **1**, определить, тела каких конструкторов и деструкторов (возможно явно не определённые) и в каком порядке будут исполнены при работе следующего фрагмента программы, расположенного в пространстве имён *S2* из задания **1**:

```
void h () { struct D { A a; C c; D (int x = 0) {}; }; D d(2013); }
```

5. Для класса рациональных дробей

```
template <class T> class fraction { T n; T d; ... };
```

написать два варианта реализации вне класса операции "<", выполняющей сравнение двух рациональных дробей, а именно – методом класса и функцией-другом класса.

6. В некоторой программе, написанной на языке Си++, имеется *try-catch* – блок, внутри которого генерируется исключение типа *E*. Каким должен быть тип *H* параметра обработчика исключения, чтобы возникшая исключительная ситуация могла быть передана ему на обработку?

7. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
struct A;
void g(A & a, int i);
struct A    { A () { cout << "Ak";
                     g (*this, -1); cout << "Ag"; }
              A (A &) { cout << "Ap"; }
};
struct B: A { B () { try { cout << "Bt";  g (*this, -1); cout << "Bf";
                         }
                     catch (B)  { cout << "Bb";  throw; }
                     catch (int) { cout << "Bi";  throw; }
              }
              B (B &) { cout << "Bp"; }
};
void g (A & a, int i) { try { cout << "gt";
                              if (i < 0)     throw a;
                              cout << "g0";
                              if (i > 0)     throw 1;
                              cout << "g1";
                        }
                        catch (int)    { cout << "gi";  throw; }
                        catch (A& a)   { g(a, 0); cout << "gr"; throw; }
}
int main() try { B a; }
           catch (...) { cout << "mc";   return 0;  }
           cout << "mr";   return 0;
}
```

8. В приведённой программе возможно наличие синтаксических ошибок в определении класса *B*. Если ошибки есть, исправьте их заменой, исключением или добавлением нужных служебных слов языка Си++. Обоснуйте сделанные исправления.

```
class B {      int x;
               int y;
               int p ()  { return y >= 0 ? y = -1 : q (); }
               int q () const { return p (); }
               int r ()  { return x = y; }
               B (int z) { x = y > 0 ? z % y : -y; }
};
int B::y = 13;
int main () {  B::y = 1;  B::p();
               const B b1 (2013), b2 (b1);
               b1.q(); return b2.x = 2;  }
```

9. В системе классов, описанной в задаче 1, в пространстве имён *S2* описать функцию **C& *f(A&);*** , которая, получая в качестве параметра ссылку на объект базового класса *A*, возвращает ссылку на объект производного класса *C*, полученную наиболее безопасным образом, а в случае невозможности приведения типов корректно завершает программу.

10.Даны описания:

```
typedef vector<double> V;
struct Signif_t { V::size_type Index;  // индекс элемента вектора
                  bool Signif; // значимость элемента (true – да, false – нет)
};
typedef list<Signif_t> S;
```

Написать функцию *g()*, которая по заданному вектору типа *V* и соответствующему ему списку типа *S* вычисляет сумму значащих элементов вектора, выдавая в выходной поток индексы и значения суммируемых элементов.

Вариант 3_2013 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие директив `#include <iostream>` и `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. Если в реализации функций *g()* и *main()* есть ошибки, исправьте их, используя только операцию расширения области видимости *"::"*. Для всех обращений к функциям *f()* и *g()* укажите, из какой области видимости они вызываются (если вариантов расширения имён несколько, укажите все). Подчеркните те расширения имён, без которых программа ошибочна.

```
int x = 0;
namespace P1 { int x = 2;      int f (int a) { return x = a; }
               class A { int x;
                  public:   A (int n = 0)   {      x        = n; }
                                 int f ()      { return     x; }
                                 int f (int a) { return x = a; }
               };
               class B { int x;
                  public:   B (int n = 1)   {      x        = n; }
                                 int f (int a) { return x = a; }
                                void g () {      x =        f (3); }
               };
               namespace P2 { struct C: A, B { int x;
                                 int f (int a) { return x = a; }
                             };
               }
}
int  main () {           A a, * pa;        B b, * pb;        C c, * pc;
              x =                f (2);  pa = &a;       x = pa ->       f (0);
pa = &c;      x = pa ->          f (3);                 x = pa ->       f ();
pb = &b;      x = pb ->          f (0);  pb = &c;       x = pb ->       f (1);
pc = &c;      x = pc ->          f (3);                 x = pc ->       f ();
                                        c.g ();        return 2013;
}
```

2. Есть ли ошибки в функции *main()* приведённого фрагмента на Си++? Если есть, то **объясните,** в чём они заключаются. Если нужно, удалите ошибочные вызовы из функции *main()*. Что будет выдано в стандартный поток вывода при работе исправленной (если нужно) программы?

```
class P { public:
          virtual void m ()      { n (0); cout << "P::m"       << endl; }
          virtual void n (int x) { l (x); cout << "P::n," << x << endl; }
                   void n ()     {        cout << "P::n"       << endl; }
          virtual void l (int x) {        cout << "P::l," << x << endl; }
                   void l ()     {        cout << "P::l"       << endl; }
};
class Q: public P { public:
          virtual void m (int x) { n (x); cout << "Q::m," << x << endl; }
          virtual void n (int x) { l (x); cout << "Q::n," << x << endl; }
          virtual void n ()      {        cout << "Q::n"       << endl; }
          virtual void l (int x) {        cout << "Q::l," << x << endl; }
          virtual void l ()      {        cout << "Q::l"       << endl; }
};
class R: public Q { public: long l;
                   void m (int x) { n (x); cout << "R::m," << x << endl; }
```

```
                void n (int x) {        cout << "R::n," << x << endl; }
};
int main( ){  R r, * pr = & r; P * pp = & r; Q * pq = & r;
              pp -> m ();       pp -> n ();          pp -> n (2013);
              pp -> l ();       pp -> l (13);        pq -> m (2013);
              pq -> n ();       pq -> l ();          pr -> m (2013);
              pr -> l ();                            return 3;       }
```

3. Какие виды расширения типов имеются в языке Си++? Приведите пример программы на языке Си++, в котором существенно используется отличие расширения от стандартного преобразования типа.

4. Используя описания классов *A* и *B* из задания **1**, определить, тела каких конструкторов и деструкторов (возможно явно не определённые) и в каком порядке будут исполнены при работе следующего фрагмента программы, расположенного в пространстве имён *P2* из задания **1**:

```
void h () { A a; class D { B b; A a; public: D (): b(2013),a() {} }; D d; }
```

5. Для класса рациональных дробей

```
template<class T> class fraction { T n; T d; ... };
```

написать два варианта реализации вне класса префиксной операции увеличения "++", выполняющей увеличение на 1 значения рациональной дроби, а именно – методом класса и функцией-другом класса.

6. В некоторой программе на языке Си++ имеются определения классов, каждый из которых содержит *f()*. В функции *main()* этой программы определен указатель *p*, который может указывать на объекты, обладающие методом *f()*. Может ли вызов *p->f()*, используемый в цикле, давать разные результаты на разных итерациях цикла, если все функции в данной программе работают только с нестатическими локальными переменными и не используют средств получения данных из внешней среды (то есть ввод, обращение к таймеру и тому подобное). Если нет, объяснить почему. Если да, указать условия, при которых это возможно и привести пример такой программы.

7. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
struct T;
void f(T & t, int n);
struct T    { T () { f(*this, 0);  cout << "Te"; }
              T (T &){ cout << "Tk"; } };
struct U: T { U (){ try { cout << "Ut"; f (*this, 1);  cout << "Uf"; }
                    catch (U)  { cout << "Uc";  throw; }
                    catch (int){ cout << "Ui";  throw; } }
              U (U &) { cout << "Uk"; }
};
void f (T & t, int a) {
   try  { cout << "ft";
          if (a < 0) throw t;
          if (a > 0) throw 1;
          cout << "fr"; }
    catch (int)    { cout << "fi";  throw; }
    catch (T& t)   { cout << "fk"; f(t, 0); cout << "ff"; throw; }
}
int main( try { U a; }
          catch (...){ cout << "mc";  return 0; }
          cout << "mr";  return 0; }
```

8. В приведённой программе возможно наличие синтаксических ошибок в определении класса *M*. Если ошибки есть, исправьте их заменой, исключением или добавлением нужных служебных слов языка Си++. Обоснуйте сделанные исправления.

```
class M {     int m;
              void m1() { if (m < 0) m2(m); }
              int  m2(int & n) const { return m1(),n; }
              void m3(int & n) { m = m2(n); } };
int M::m = 1;
int main() {  M::m3 (2013);   M mm;   return mm.m2 (3); }
```

9. В системе классов, описанной в задаче 1, в пространстве имён *P2* описать функцию **C* *f(B&);*** , которая, получая в качестве параметра ссылку на объект класса *B*, возвращает указатель на объект класса *C*, полученный наиболее безопасным образом, а в случае невозможности приведения типов корректно завершает программу.

10.Даны описания:

```
typedef vector<bool> B;
struct Value_t { B::size_type Index;  // индекс элемента вектора
                 int Value;       };  // значение элемента
typedef list<Value_t> T;
```

Написать функцию *g()*, которая по заданному вектору значимости типа *B* и соответствующему ему списку типа *T* в порядке обратном порядку элементов списка выдаёт в выходной поток значения полей элементов списка, сопровождаемое их значимостью, вычисляет сумму значимых целочисленных полей списка и возвращает это значение.

Ответы_Вариант 3_2013

1.**Ответ:**

```
void g () { x = P1::B::f (3); }

int main {   P1::A a, * pa;      P1::B b, * pb;      P1::P2::C c, * pc;
              x = P1:: f (2);
     pa = &a;   x = pa ->  P1::A:: f (0);
     pa = &c;   x = pa ->  P1::A:: f (3);
                x = pa ->  P1::A:: f ();
     pb = &b;   x = pb ->  P1::B:: f (0);
     pb = &c;   x = pb ->  P1::B:: f (1);
     pc = &c;   x = pc ->  P1::C:: f (3);
                x = pc ->  P1::A:: f ();
                c.P1::B:: g (); }
```

Отмечены исправляющие (обязательные) изменения.

Критерии: За отсутствие каждого **необходимого** расширения имени (без которого ошибка) **-2** (за каждое из 5).

За каждое другое не отмеченное уточнение области видимости (из 8) **-1**.

При указании хотя бы одного правильного расширения имени (где это необходимо) за отсутствие вариантов расширений не наказывать.

Для остальных расширений имен считать достаточным указание ближайшей области видимости, за отсутствие полной цепочки вложенных областей видимости – не наказывать.

2. Ошибочный вызов – pr -> l() :
(имя функции скрывает имя поля-данного l)

Будет выдано в поток:

```
R::n,0
P::m
P::n
R::n, 2013
P::l
Q::l, 13
R::n, 2013
R::m, 2013
Q::n
Q::l
R::n, 2013
R::m, 2013
```

Критерии: За каждую не найденную ошибку, лишнюю ошибку или неправильно вызванную функцию **– 1**.

3. Расширение типа – некоторая часть стандартных преобразований (часто – наиболее безопасная часть). Конкретно к расширениям относятся:

- целочисленные расширения:
  (1) *char* --> *int* (*unsigned int*);
  (2) *short* --> *int* (*unsigned int*);
  (3) *enum* --> *int* (*unsigned int*);
  (4) *bool* --> *int*;
  (5) *enum* --> *long int* (*unsigned long int*), если тип *int* не подходит;
  (6) если можно, то битовые поля могут расширяться до *int* (*unsigned int*), иначе они не расширяются.
- расширения чисел с плавающей точкой: *float* --> *double*.

```
void f (double) /* … */; void f (int) /* … */;
void g () { short aa = 1;     // тип short расширяется до int
            float ff = 1.0;
            f (ff);           // f (double)
            f (aa);           // f (int)
          } // Здесь расширение избавляет от неоднозначности выбора функции f().
```

Критерии: За пропуск пунктов (5) и (6) не наказывать.
За пропуск любого другого из 5 оставшихся видов расширения **–2**.
За пропуск беззнаковых вариантов – не наказывать.
За совсем неправильное объяснение **–10**.
За отсутствие примера **–2**.

4. Ответ: `A (), B(2013), A (), D (), ~D (), ~A (), ~B (), ~A ().`

Критерии: За каждую ошибку **–3**.
За указание `B()` вместо `B(2013)` снижать на **1** балл.

5.
```
template<class T> fraction<T> & fraction<T>::operator++ ()
                          { r += d; return * this; }
template<class T> fraction<T> & operator++ (fraction<T>& x)
                          { x.r += x.d; return x; }
```

Критерий: За отсутствие\ошибку каждого примера (за однотипные ошибки наказывать 1 раз): **-5.**

6. Да, может. В программе должна быть определена иерархия классов, в которую входят классы с виртуальным методом *f()*. Каждый из методов должен реализовывать разные алгоритмы, возвращающие разные результаты. При обращении к методу *f()* указатель *p* на разных итерациях должен содержать значения, относящиеся к разным классам иерархии.

```
struct A   { virtual int f(int) { return 0; } ... };
struct B:A { virtual int f(int) { return 1; } ... };
int main () { int i; A * a [5], * p; B b [5]; A a2;
              for (i = 0; i < 5; ++i) { a [i] = & b [i]; }
              a [2] = & a2;
              for (i = 0; i < 5; ++i) { p = a [i]; p -> f (); }
}
```

Критерий: За ответ “нет”: **-10**.
За неправильное объяснение и пример: **-10**.
За отсутствие объяснения при правильном примере: **-2**.
За отсутствие примера при правильном подробном объяснении: **-3**.

7. `ft fr Te Ut ft fi Ui mc`

Критерии: За каждую неверную (лишнюю или пропущенную) печать **-2**.

8. **1.** Заменить слово *"class"* словом *"struct"*.
2. Перед определением типа поля *"m"* вставить слово *"static"*.
3. В определениях методов *"m1()"*, *"m2()"* и *"m3()"* перед типом возвращаемого значения вставить слово *"static"*.
4. В определении метода *"m2()"* удалить слово *"const"*.
5. В определениях методов *"m2()"* и *"m3()"* вставить перед типом формального параметра слово *"const"*.

Критерий: За пропуск ошибок доступа к закрытым членам класса: **-5**.

За пропуск указания статичности поля *"m"*: **-8.**
За пропуск любой ошибки определения статических методов *"m1()"*, *"m2()"* и *"m3()"*: **-6**.
За пропуск ошибки указания константности статического метода *"m2()"*: **-5.**
За пропуск каждой из двух ошибок в заголовках функций: **-4.**
За указание ошибки там, где её нет: **-5.**

9.
```
C * f (B & rb){
     try { C & rc = dynamic_cast<C &>(rb); return & rc; }
     catch (bad_cast) { exit (0); }
}
```

Критерий: За утверждение о невозможности безопасного преобразования ставить полный балл: **10**.
За безошибочное решение с использованием динамического приведения (как здесь написано, как будто класс полиморфен): **-1**.
За любую другую ошибку: **-10** (включая использование статического приведения).

10.
```
int g (const V & vect, const T & lst)
{ T::const_reverse_iterator lp = lst.rbegin ();
  B::size_type t; int vl;
  int w = 0;
  while (lp != lst.rend ())
     { t = (* lp).Index; vl = (* lp ++).Value;
       if (t < 0 || t >= vect.size ()) continue;
       if (vect [t]) w += vl;
       cout << t << ' ' << vl << ' ' << vect [t] << endl;
     }
  return w;
}
```

Критерий: За каждую серьёзную ошибку: **-3**.
За использование обычного итератора вместо константного, отсутствие проверок на пустой список и выход за пределы диапазона индексов не наказывать.

2013 год
Экзамен

Зариант 1_2013 Ф.И.О._____ № группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to aA$
$aA \to aaB \mid a$
$B \to abb \mid aCbb \mid b$
$C \to ab \mid aaD$
$D \to bb \mid abbb$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы.
Ответ:
$L(G)$=

(б) Каким из перечисленных классов грамматик принадлежит G?
Ответ:

Класс П	$G \in$ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

(в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком типа k+1.
Ответ: k = _____

2. Дать определение системы программирования.
Из перечисленных компонентов вычеркните те, которые не входят в состав системы программирования:

текстовый редактор,
макрогенератор,
драйвер клавиатуры,
редактор связей.

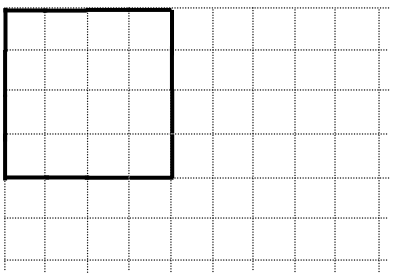
3. 1) Дать определение языка, порождаемого грамматикой G=(T,N,P,S).
2) Привести пример грамматики (задав все элементы порождающей грамматики), которая порождает пустой язык.

4.

Имеется клетчатый лист бумаги, бесконечный вправо и вниз. В левом верхнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита {*a*, *b* ,*c* ,*d*}:

a — переместить перо на одну клетку вправо;

b — вниз; *c* — влево; *d* — вверх.



Например, по последовательности *aaaabbbbccccdddd* интерпретатор построит квадрат, изображенный на рисунке, со стороной длины 4.

Построить грамматику с терминальным алфавитом {*a*, *b, c, d*}, описывающую все квадраты со стороной длины n ($n \geq 1$), начерченные по часовой стрелке. В грамматике должно быть **не более** 6 правил вывода (считая альтернативы). Решения с б**о**льшим числом правил не рассматриваются. Грамматика не должна описывать другие фигуры.

5. 1) Привести по одному примеру ситуаций, когда под объекты программы выделяется:

а) **статическая**, б) **динамическая** (произвольного распределения), в) **стековая память**.

2) В какой памяти располагаются таблицы виртуальных методов полиморфных классов?

5. Дана грамматика G: $S \to aA|b \mid \varepsilon$
$A \to AA \mid B$
$B \to bBB$
$C \to d$

а) Является ли G приведенной? **Обоснуйте ответ**. Если нет, примените к ней алгоритм приведения грамматики, и выпишите получившуюся грамматику G'.

б) Является ли исходная грамматика G однозначной? **Обоснуйте ответ**.

7. Дана автоматная леволинейная грамматика:
$S \to C\perp$
$C \to A1$
$A \to B1 \mid C0 \mid 0$
$B \to B1 \mid 0$

а) Детерминирован ли разбор по данной грамматике (**ответ обосновать**)?

б) Построить конечный автомат (КА) по заданной грамматике.

в) Если КА недетерминированный, преобразовать его к ДКА.

г) Построить **праволинейную** грамматику, соответствующую ДКА.

8. Построить неукорачивающую КС-грамматику, эквивалентную заданной КС грамматике, пользуясь **алгоритмом преобразования «КС -> неукорачивающая КС»**.

$S \to ABc \mid c \mid A$
$A \to aA \mid \varepsilon$
$B \to bB \mid \varepsilon$

9. Применим ли метод рекурсивного спуска к заданной грамматике. **Ответ обосновать**.

$S \to ABc \mid c$
$A \to aA \mid \varepsilon$
$B \to bB \mid \varepsilon$

10. Дана польская инверсная запись фрагмента программы (на языке Си). Вставьте пропущенные в польской записи команды перехода (‘ ! ’, ‘ ! F’) и метки и восстановите фрагмент программы, если известно, что в исходном фрагменте не было операторов перехода.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
x	0	>			y	x	<			&y	y	1	x	+	+	=	;			27		&y

24	25	26	27	
x	=	;		

Ответы_Вариант 1_2013

1.
Тип гр. – 0 (другие - нет),
$L(G) = \{ a^n b^{n-1} \mid n=1,2,3,4,5,6 \}$. Тип языка 3 (регулярный).

Критерии: Неверный язык - **-4**
Неверный тип языка - **-4**
Неверный основной тип грамматики- **-3**
За каждую неверную принадлежность другим типам - **-1**

2. **Система программирования** - это комплекс программных инструментов и библиотек, который поддерживает **весь** технологический (жизненный) цикл создания программного продукта.
Не входит - драйвер клавиатуры.

Критерии: Нет\Неверное определение – **-6**
Нет библиотек – не наказывать
Нет технологического цикла - **-4**
Любая ошибка(и) в вычеркивании - **-5**

3. 1) Язык, порождаемый грамматикой G=(T,N,P,S), это **множество терминальных** цепочек конечной длины, **выводимых из начального символа грамматики**.
Или **$L(G) = \{ \alpha \in T^* \mid S \Rightarrow \alpha \}$**
2) G = {{a}, {S, A}, {S -> Aa}, S} , или G = { {a}, {S}, ∅, S}, или ...

Критерии: Неверное определение - **-5**,
За отсутствие слов «конечной длины» – не наказывать,
выделенное жирным - **обязательно**.
Нет\неверный пример - **-5**.

4. Нужно описать язык $\{a^n b^n c^n d^n \mid n \geq 1\}$. $S \to aBSCd \mid abcd$
$Ba \to aB$
$dC \to Cd$
$Bb \to bb$
$cC \to cc$

Критерии: Ввыводится лишняя цепочка, не выводится нужная, превышение количества правил: **-10**
***Внимание**: в корректном решении не может быть правил вида $B \to b$, когда нетерминал превращается в терминал независимо от контекста, за подобные правила сразу **-10**.*
Если грамматика неверная или отсутствует, но верно указан язык $\{a^n b^n c^n d^n \mid n \geq 1\}$ – всего **+2**.

5. 1) а) Статическая – при описании, например, глобальных переменных.
б) Динамическая – например, при выделении памяти с помощью new, при работе с контейнерами STL.
в) Стековая – например, при работе рекурсивных функций, при передаче фактических параметров по значению в функции.
2) В статической.

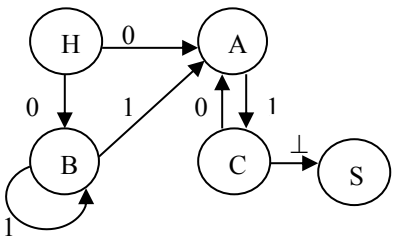
Критерии: За каждый неверный пример - **-3**
За неверный ответ на вопрос - **-3**..

6. (а) Не является приведенной, т.к. содержит недостижимый символ *C*. (Возможно указание других бесполезных символов, достаточно указать хотя бы один).
G': $S \to b \mid \varepsilon$
(б) Является однозначной, т.к. в ней выводятся всего две цепочки: *b* и ε, обе выводятся за один шаг вывода, так что дерево вывода каждой цепочки единственно.

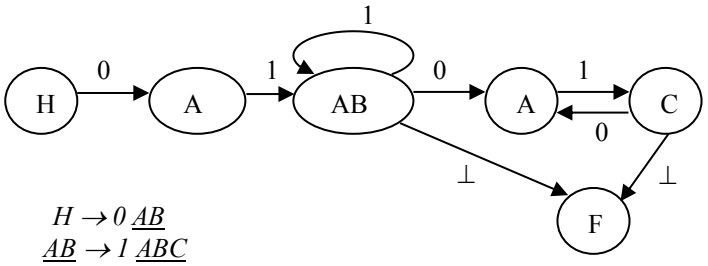
Критерии: неверный (необоснованный) ответ или ошибки в пункте (а) : **-5**
неверный (необоснованный) ответ или ошибки в пункте (б): **-5**

7. а) Нет, т.к. правила вывода содержат повторяющиеся правые части. Обоснование возможно также через автомат.

б)



в)



г) $H \to 0\,\underline{AB}$
$\underline{AB} \to 1\,\underline{ABC}$
$\underline{ABC} \to 1\,\underline{ABC} \mid 0A \mid \perp$
$A \to 1C$
$C \to 0A \mid \perp$

Критерии: Неверный ответ на вопрос а): **-5,** нет обоснования: **-3.**

Неверный КА: **-4**
Неверный ДКА: **-5**
Неверная грамматика для ДКА: **-2**

8. $S' \to S \mid \varepsilon$
$S \to ABc \mid Ac \mid Bc \mid c \mid A$
$A \to aA \mid a$
$B \to bB \mid b$

Критерии: Пропуск **каждой альтернативы** - **-5.** (Пропуск первой строки - **-10**)

9. Нет, т.к. пересечение множеств first (ABc) = {a,b,c} и first (c) = {c} непусто.

Критерии: Неверный ответ или неверное обоснование - **-10.**
Неточности в множествах first(A) и/или follow(A): **-2**

10.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
x	0	>	23	!F	y	x	<	21	!F	&y	y	1	x	+	+	=	;	6	!	27	!	&y

24	25	26	27	
x	=	;		

if (x > 0) while (y < x) y = y + (1 + x) ; else y = x;

Критерии: Есть ошибки в ПОЛИЗЕ: -5;
Есть ошибки в восстановленном фрагменте: -5
Внимание: скобки в выражении **y+(1+x)** обязательны, без них ПОЛИЗ был бы другим.
Если без ошибок *только* операции перехода расставлены, даем **2 балла** за задачу.

Вариант 2_2013 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

S → 0A1
0A → 0B1 | 01
B1 → 0C111 | 011
C → 0D | 00D1 | 0
D → 01

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы.
Ответ:
$L(G)$=

(б) Каким из перечисленных классов грамматик принадлежит G?
Ответ:

Класс П	$G \in$ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком типа k+1.
Ответ: k = _____

2. Привести общую схему работы компилятора с указанием результатов работы каждого из этапов.
Из перечисленных действий вычеркните те, которые не участвуют в компиляции:
лексический анализ,
контроль контекстных условий,
верификация,

генерация машинного кода.

3. 1) Дать определение эквивалентных грамматик.
2) Привести пример эквивалентных грамматик.

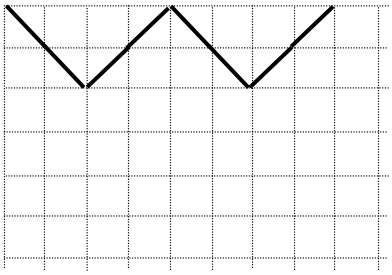
4.

Имеется клетчатый лист бумаги, бесконечный вправо и вниз. В левом верхнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита $\{a, b\}$:

a — переместить перо по диагонали на одну клетку вправо вниз;

b — переместить перо по диагонали на одну клетку вправо вверх.

Например, по последовательности *aabbaabb* интерпретатор построит ломаную из четырех равных отрезков длины 2, похожую на букву W (см. рисунок).



Построить грамматику с терминальным алфавитом $\{a, b\}$, описывающую все буквы W с длиной отрезка n ($n \geq 1$). В грамматике должно быть **не более** 6 правил вывода (считая альтернативы). Решения с бо**л**ьшим числом правил не рассматриваются. Грамматика не должна описывать другие фигуры.

5. Каковы основные свойства языка внутреннего представления программы?
Привести **два примера** таких языков.

6. Дана грамматика G:

$$S \to B \mid C$$
$$C \to c \mid cC\,B$$
$$D \to d$$
$$B \to bcB \mid B$$

а) Является ли G приведенной? **Обоснуйте ответ**. Если нет, примените к ней алгоритм приведения грамматики, и выпишите получившуюся грамматику G'.
б) Является ли исходная грамматика G однозначной? **Обоснуйте ответ**.

7. Дана автоматная праволинейная грамматика:

$S \to 0A \mid 0B \mid 1C$
$A \to 0B \mid 1C$
$B \to 1B \mid 1A$
$C \to \perp$

а) Детерминирован ли разбор по данной грамматике (**ответ обосновать**)?
б) Построить конечный автомат (КА) по заданной грамматике.
в) Если КА недетерминированный, преобразовать его к ДКА.
г) Построить **леволинейную** грамматику, соответствующую ДКА.

8. Построить неукорачивающую КС-грамматику, эквивалентную заданной КС грамматике, пользуясь **алгоритмом преобразования «КС -> неукорачивающая КС»**.

$S \to ABc \mid A$
$A \to aAb \mid \varepsilon$
$B \to bBa \mid \varepsilon$

9. Применим ли метод рекурсивного спуска к заданной грамматике. **Ответ обосновать**.

$S \to AB \mid \varepsilon$
$A \to aAb \mid \varepsilon$
$B \to bBa \mid \varepsilon$

10. Дана польская инверсная запись фрагмента программы (на языке Си). Вставьте пропущенные в польской записи команды перехода (‘ ! ’, ‘ ! F’) и метки и восстановите фрагмент программы, если известно, что в исходном фрагменте не было операторов перехода.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
y	x	<			x	0	>			&y	y	2	x	+	+	=	;	25		&y	x	=

24	25	26	27
;	1		

Ответы_Вариант 2_2013

1.
Тип гр. – КЗ, НУ, тип 0 (другие - нет),
$L(G) = \{0^n 1^{n+1} \mid n=1,2,3,4,5\}$. Тип языка 3 (регулярный).

Критерии: Неверный язык - **-4**
Неверный тип языка - **-4**
Неверный основной тип грамматики- **-3**
За каждую неверную принадлежность другим типам - **-1**

2. Фаза анализа: (исходная программа на ЯП) ⇒ лексический анализ ⇒ (последовательность лексем) ⇒ синтаксический анализ ⇒ (промежуточное представление программы) ⇒ семантический анализ//контроль контекстных условий.
Фаза синтеза: подготовка к генерации объектного модуля ⇒ генерация объектного модуля ⇒ (объектный модуль)
Не участвует – верификация.

Критерии: Нет\Неверная схема – **-6**
Нет фазы синтеза - **-4**
Нет промежуточных результатов - **-2 за каждый.**
Любая ошибка(и) в вычеркивании - **-5**

3. 1) Грамматики эквивалентны, если порождают один и тот же язык.
2) Пример: $S \to 0S1 \mid 01$ и $S \to 0A1$
$A \to 0A1 \mid \varepsilon$

Критерии: Неверное определение - **-5**,
Нет\неверный пример - **-5**.

4. Нужно описать язык $\{ a^n b^n a^n b^n \mid n \geq 1 \}$.

$S \to aBSAb \mid abab$
$Ba \to aB$
$bA \to Ab$
$aA \to aa$
$Bb \to bb$

Критерии: Ввыводится лишняя цепочка, не выводится нужная, превышение количества правил: **-10 .**
Внимание: в корректном решении не может быть правил вида $B \to b$, когда нетерминал превращается в терминал независимо от контекста, за подобные правила сразу **-10.**
Если грамматика неверная или отсутствует, но верно указан язык $\{a^n b^n a^n b^n \mid n \geq 1\}$,
даем всего **2** балла за задачу.

5. Свойства:
а) позволяет фиксировать синтаксическую структуру исходной программы;

b) текст на нем можно автоматически генерировать во время синтаксического анализа;

c) его конструкции относительно просто транслируются в объектный код либо достаточно эффективно интерпретируются.

Примеры: Р-код, байт-код виртуальной Java-машины, ПОЛИЗ...

Критерии: За пункт а) - **-4**,
За пункты b) и c) по **-2 за каждый**
За **каждый** неверный\отсутствующий пример - **-2**

6. (а) Не является приведенной, т.к. содержит недостижимый символ *D*. (Возможно указание других бесполезных символов, достаточно указать хотя бы один).

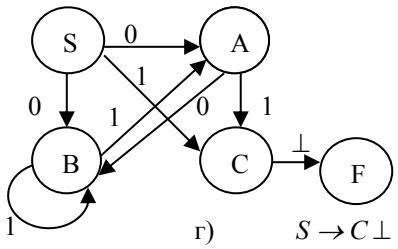
G': $S \to C$
$C \to c$

(б) Является однозначной, т.к. в ней выводятся всего одна цепочка: *c*, для которой существует единственный вывод $S \to C \to c$, а значит, единственно дерево вывода.

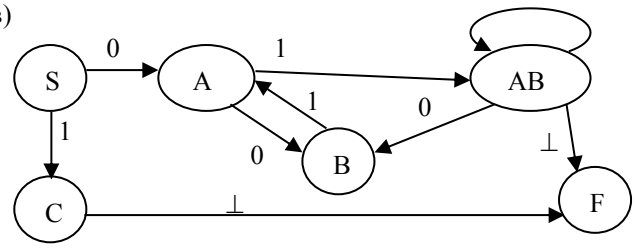
Критерии: неверный (необоснованный) ответ или ошибки в пункте (а) : **-5**
неверный (необоснованный) ответ или ошибки в пункте (б): **-5**

7. а) Нет, т.к. есть альтернативы правил вывода, начинающиеся одинаковыми нетерминалами. Обоснование возможно также через автомат.

б)



в)



г)

$S \to C \perp \mid \underline{ABC} \perp$
$C \to 1$
$\underline{ABC} \to \underline{ABC}\ 1 \mid \underline{AB}\ 1$
$B \to \underline{ABC}\ 0 \mid \underline{AB}\ 0$
$\underline{AB} \to B1 \mid 0$

Критерии: Неверный ответ на вопрос а): **-5,** нет обоснования: **-3.**
Неверный КА: **-4**
Неверный ДКА: **-5**
Неверная грамматика для ДКА: **-2**

8. $S' \to S \mid \varepsilon$
$S \to ABc \mid Ac \mid Bc \mid c \mid A$
$A \to aAb \mid ab$
$B \to bBa \mid ba$

Критерии: Пропуск **каждой альтернативы** - **-5.** (Пропуск первой строки - **-10**)

9. Нет, т.к. $AB \Rightarrow \varepsilon$ и $\varepsilon \Rightarrow \varepsilon$.

Критерии: Неверный ответ или неверное обоснование - **-10.**

10.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
y	x	<	27	!F	x	0	>	21	!F	&y	y	2	x	+	+	=	;	25	!	&y	x	=

24	25	26	27
;	1	!	

while (y < x) **if** (x > 0) y = y + (2 + x); **else** y = x;

Критерии: Есть ошибки в ПОЛИЗЕ: **-5;**
Есть ошибки в восстановленном фрагменте: **-5**

Внимание: скобки в выражении **y+(2+x)** обязательны, без них ПОЛИЗ был бы другим.

Если без ошибок *только* операции перехода расставлены, даем **2 балла** за задачу.

1	2	3	4	5	6	7	8	9	10

Вариант 3_2013

Ф.И.О._____№ группы_____

1. Дана грамматика G:

$S \to aAb \mid \varepsilon$
$A \to aBbb \mid ab \mid \varepsilon$
$aB \to aaaCb \mid aa$
$C \to D \mid \varepsilon$
$D \to ab$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы.
Ответ:
$L(G)$=

(б) Каким из перечисленных классов грамматик принадлежит G?
Ответ:

Класс П	$G \in$ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

(в) Тип языка: найти такое целое k, что язык $L(G)$ является языком типа k и не является языком типа k+1.
Ответ: k = _____

2. В чем отличие компиляторов и интерпретаторов?
Из перечисленных действий вычеркните те, которые не участвуют в интерпретации:

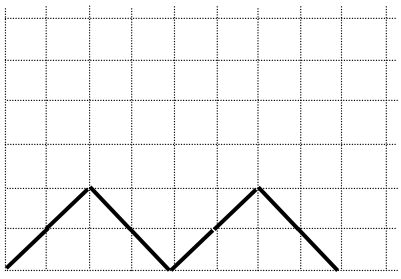
синтаксический анализ,
генерация объектного модуля,
работа с таблицей имен,
ввод входных данных.

3. 1) Дать определение почти эквивалентных грамматик.
2) Привести пример почти эквивалентных грамматик.

4.
Имеется клетчатый лист бумаги, бесконечный вправо и вверх. В левом нижнем углу расположено перо, оставляющее на бумаге след при перемещении. Перо управляется интерпретатором-графопостроителем. На вход интерпретатору подается последовательность символов-команд из алфавита $\{c, d\}$:

c — переместить перо по диагонали на одну клетку вправо вверх;

d — переместить перо по диагонали на одну клетку вправо вниз.



Например, по последовательности *ccddccdd* интерпретатор построит ломаную из четырех равных отрезков длины 2, похожую на букву М (см. рисунок).

Построить грамматику с терминальным алфавитом $\{c, d\}$, описывающую все буквы М с длиной отрезка n ($n \geq 1$). В грамматике должно быть **не более** 6 правил вывода (считая альтернативы). **Решения с большим числом правил не рассматриваются**. Грамматика не должна описывать другие фигуры.

5. Перечислить основные функции редактора связей.
Привести пример ситуации, в которой редактор связей выдаст сообщение об ошибке.

6. Дана грамматика *G*: $S \to aSB \mid c$
$A \to aA \mid Aa \mid a$
$C \to \varepsilon$
$B \to bB$

а) Является ли *G* приведенной? **Обоснуйте ответ**. Если нет, примените к ней алгоритм приведения грамматики, и выпишите получившуюся грамматику *G'*.
б) Является ли исходная грамматика *G* однозначной? **Обоснуйте ответ**.

7. Дана автоматная леволинейная грамматика:

$S \to C \perp$
$C \to Ab \mid Ba$
$A \to Ab \mid a$
$B \to Bb \mid a$

а) Детерминирован ли разбор по данной грамматике (**ответ обосновать**)?

б) Построить конечный автомат (КА) по заданной грамматике.
в) Если КА недетерминированный, преобразовать его к ДКА.
г) Построить **праволинейную** грамматику, соответствующую ДКА.

8. Построить неукорачивающую КС-грамматику, эквивалентную заданной КС грамматике, пользуясь **алгоритмом преобразования «КС -> неукорачивающая КС»**.

$S \to aSBb \mid bA \mid B$
$A \to aB \mid b$
$B \to bB \mid \varepsilon$

9. Применим ли метод рекурсивного спуска к заданной грамматике. **Ответ обосновать**.

$S \to aSb \mid bA$
$A \to aB \mid b$
$B \to bB \mid \varepsilon$

10. Дана польская инверсная запись фрагмента программы (на языке Си). Вставьте пропущенные в польской записи команды перехода (‘ ! ’, ‘ ! F’) и метки и восстановите фрагмент программы, если известно, что в исходном фрагменте не было операторов перехода.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
x	0	>			&y	y	3	x	+	+	=	;	y	x	<			6		27		&y

24	25	26	27
x	=	;	

Ответы_Вариант 3_2013

1.
Тип гр. – 0 (другие - нет), L(G) = { $a^n b^n \mid n=0,1,2,3,4,5$ }. Тип языка 3 (регулярный).

Критерии: Неверный язык - **-4**
Неверный тип языка - **-4**
Неверный основной тип грамматики- **-3**
За каждую неверную принадлежность другим типам - **-1**

2.
Компилятор преобразует программу на ЯП (немашинном) в эквивалентную программу в машинных кодах, которая затем может использоваться многократно независимо от компилятора, получая во время выполнения входные данные и выдавая результат.

Интерпретатор, получая на вход исходные данные, читает предложение за предложением исходной программы на ЯП, переводит в машинные коды и непосредственно выполняет его, получая в итоге результат выполнения программы.

Не участвует – генерация ОМ.

Критерии: Неверный ответ на первый вопрос - **-6**
Неточности (индивидуально)- от **-1** до **-2 за каждую.**
Любая ошибка(и) в вычеркивании - **-5**

3. 1) Грамматики почти эквивалентны, если порождаемые ими языки отличаются не более, чем на ε. Или Грамматики G1 и G2 ***почти эквивалентны***, если $L(G1) \cup \{\varepsilon\} = L(G2) \cup \{\varepsilon\}$.

2) Пример: $S \to 0S1 \mid \varepsilon$ и $S \to 0A1$

$A \to 0A1 \mid \varepsilon$

Критерии: Неверное определение - **-5**,
За «не более, чем» не наказывать.
Нет\неверный пример - **-5**.

4. Нужно описать язык $\{c^n d^n c^n d^n \mid n \geq 1\}$.

$S \to cDSCd \mid cdcd$

$Dc \to cD$

$dC \to Cd$

$Dd \to dd$

$cC \to cc$

Критерии: Выводится лишняя цепочка, не выводится нужная, превышение количества правил: **-10**
Внимание: в корректном решении не может быть правил вида $B \to b$, когда нетерминал превращается в терминал независимо от контекста, за подобные правила сразу **-10**.
Если грамматика неверная или отсутствует, но верно указан язык $\{c^n d^n c^n d^n \mid n \geq 1\}$ – всего **+2**.

5. 1. связывает между собой по внешним данным объектные модули, порождаемые компилятором и составляющие единую программу,
2. связывает файлы статически подключаемых библиотек с целью получения единого исполняемого модуля,
3. готовит таблицу трансляции относительных адресов для загрузчика,
4. готовит таблицу точек вызова функций динамически подключаемых библиотек.

Пример: отсутствие описания объявленного имени функции при его использовании.

Критерии: За отсутствующий или неверный пункт 1-3 - **-3**,
За отсутствующий или неверный пункт 4 - **-1**,
Нет\неверный пример - **-3**.

6. (а) Не является приведенной, т.к. содержит недостижимый символ *C*. (Возможно указание других бесполезных символов, достаточно указать хотя бы один).

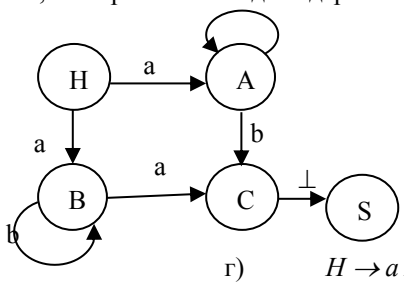
G': $S \to c$

(б) Является однозначной, т.к. в ней выводятся всего одна цепочка: *c*, для которой существует единственный вывод $S \to c$, а значит, единственно дерево вывода.

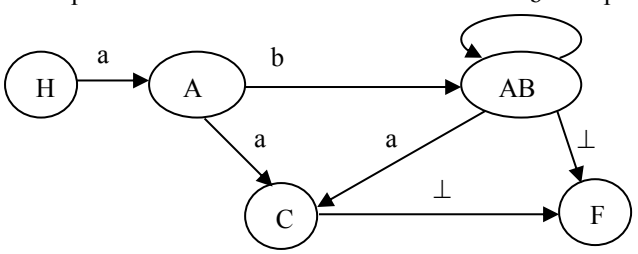
Критерии: неверный (необоснованный) ответ или ошибки в пункте (а) : **-5**
неверный (необоснованный) ответ или ошибки в пункте (б): **-5**

7. а) Нет, т.к. правила вывода содержат повторяющиеся правые части. Обоснование возможно через автомат.

б)



в)



г) $H \to a\,\underline{AB}$

$\underline{AB} \to aC \mid b\,\underline{ABC}$

$\underline{ABC} \to b\,\underline{ABC} \mid aC \mid \perp$

$C \rightarrow \perp$

Критерии: Неверный ответ на вопрос а): **-5,** нет обоснования: **-3.**
Неверный КА: **-4**
Неверный ДКА: **-5**
Неверная грамматика для ДКА: **-2**

8. $S' \rightarrow S \mid \varepsilon$
$S \rightarrow aSBb \mid aBb \mid aSb \mid ab \mid bA \mid B$
$A \rightarrow aB \mid a \mid b$
$B \rightarrow bB \mid b$

Критерии: Пропуск **каждой альтернативы** - **-5.** (Пропуск первой строки - **-10**)

9. Нет, т.к. пересечение множеств first (bB) = {b} и follow (B) = {b} непусто.

Критерии: Неверный ответ или неверное обоснование - **-10.**
Неточности в множествах first(A) и/или follow(A): **-2**

10.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
x	0	>	23	!F	&y	y	3	x	+	+	=	;	y	x	<	21	!F	6	!	27	!	&y

24	25	26	27
x	=	;	}

if (x > 0) **do** y = y + (3 + x) **while** (y < x); **else** y = x;

Критерии: Есть ошибки в ПОЛИЗЕ: **-5**;
Есть ошибки в восстановленном фрагменте: **-5**
Внимание: скобки в выражении **y+(3+x)** обязательны, без них ПОЛИЗ был бы другим.
Если без ошибок *только* операции перехода расставлены, даем **2 балла** за задачу.

2014 год
Коллоквум

Вариант 1_2014 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. При компиляции функции main () следующей программы фиксируется ошибка. Объясните, в чем она заключается, и исправьте ее, модифицируя только структуру B, ничего в ней не удаляя.

```
struct A {
      int y;
      A (int n){ y = n;}
};
struct B:A {};
int main () {
      B b;
       cout << b.y << '\n';
       return 0;
 }
```

2. Вычеркните неразрешимые вызовы функции f(...). Что напечатает получившаяся программа?

```
 struct A {  int n;
             A (int  m) { n = m; }
 };
void  f ( int i,  char c )  { cout << "f(int, char)\n";}
void  f ( int i,  int  j )  { cout << "f(int, int)\n";}
void  f ( const char * p,  char c )  { cout << "f(const char *, char)\n";}
void  f ( const char * p,  A a )  { cout << "f(const char *, A)\n";}
int main () {
     A a (1);
     f (0, a);
     f ('*', a);
     f (0, 'a');
     return 0;   }
```

3. C++11. В каких ситуациях использование move-семантики (семантики переноса) позволяет повысить эффективность работы программы?

6. Описать **<u>класс</u>** A и необходимые функции так, чтобы при компиляции заданной функции *main ()* не были обнаружены ошибки.

```
int main ( ) {
        A a (1),  b (2),  c (3);
        cout << (a += b += c) << endl;
        return 0;
}
```

7. Что напечатает программа?

```
struct B {
        virtual void f (int a) { g ();   cout << a << " - B::f (int) \n";}
        void g () { cout << "B::g () \n";}
};
class D: public B {
        void f (int b = 7) { g ();   cout << b << " - D::f (int) \n";}
        virtual void g () {cout << "D::g () \n";}
};
int main () {
        D d;
        B  b,  & rb = d;
        b. f (5);
        rb. f (6);     return 0;     }
```

6. Что такое функция преобразования в языке Си++? Привести **пример** использования функции преобразования.

7. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
struct A {
       A () { cout << "A_Constr\n"; }
       A (const A & a) { cout << "A_Copy\n"; }
       ~A () { cout << "A_Destr\n"; }
};

struct B : A {
       B () { cout << "B_Constr\n"; }
       ~B () { cout << "B_Destr\n"; }
};
```

```
void g() { B bg; throw bg;}

int main () {
       try {  A a; g(); }
       catch ( B & ) { cout << "B&_Catch\n"; }
       catch ( A &) { cout << "A&_Catch\n"; }
       return 0;
}
```

8. Исправьте ошибки в нижеследующей программе, ничего в ней **не удаляя**.
Объясните, в чем заключаются исправленные ошибки.
Что напечатает программа после внесенных исправлений?

```
struct A {
         int y;
         static int f () { return (y = 7); }
};
struct B : A {
          static int f () { return 9; }
};
int main () {
         B b,d;
         b.y = b.f();
         cout << B::f() << ' ' << B::y << ' ' << d.y << A::f() << endl;
         return 0; }
```

9. Модифицируйте программу, **ничего в ней не удаляя**, так, чтобы использование операции **dynamic_cast** было корректно, а программа в целом завершилась нормально (не аварийно).

```
struct B {
       void f () { cout << "B::f()\n"; }
};
struct D:B {};

int main () {
       D d, &rd  = d;
       B b, & rb = b, &rbd = d;
       rd = dynamic_cast <D&> (rbd);  rd.f();
       rd = dynamic_cast <D&> (rb);   rd.f();
       return 0; }
```

10. Написать **ОДНУ функцию (beforelast)**, которая для своего аргумента-контейнера целочисленных элементов возвращает значение предпоследнего элемента, если таковое имеется, или 0. Использование **beforelast () должно быть корректно** в следующей функции:

```
int  ss (const vector <int> & V, const list <int> & L){
          return   beforelast(V) + beforelast(L);
       }
```

Ответы_Вариант 1_2014

1. Надо добавить в класс B, например, конструктор умолчания со списком инициализации для вызова конструктора класса A с одним параметром.

Критерии: Ошибка не найдена или неправильно объяснена – **0**.
Неверное или с ошибками только исправление **- 5**

2. Неразрешимый вызов – f (**'*'** , a);

Будет напечатано: f (**const char** * , A)
f (**int**, **char**)

Критерии: За каждое неверное вычеркивание или неправильно вызванную функцию **– 5**.

3. При копировании временных объектов неплоских классов.

Критерии: За отсутствие уточнения «неплоских» **– -5**.
В остальных случаях – **0**, но! копирование ≡ работа КК и\или операции= ...

4.
```
struct A {                                  // есть и другой вариант решения: в А перегрузить +=
        int n;                              // дважды (с параметром А и int и результатом int), тогда
        A(int i) {n = i;}           // не нужно перегружать <<
        A& operator += (A & a) { n += a.n;  return * this; }
}
ostream & operator << (ostream & s, A& a) {
        s << a.n;
        return s;
}
```

Критерии: За каждую неверную\отсутствующую операцию **–5**.
За другие мелкие погрешности - **-2** за каждую.
За **typedef int** A – **0**.

5.
```
B::g()
5 - B::f(int)
D::g()
6 - D::f(int)
```

Критерий: Ошибки в ответе – **0**, иначе – **10**.
Ошибки, не связанные с виртуальностью - **-2**.

6. Функция преобразования – **метод класса** специального вида (***operator*** <*mun*> *()*), который задает правило приведения объектов класса к типу <*mun*>, например, "***operator int()***". Алгоритм преобразования кодируется в теле функций преобразования.

Пример:

```
struct S {
      long s;
     S(){ s = 0L; }
     operator long (){ return s; } //   S --> long
};
long f(long l) { return l; }
..... S s; long nl = f (s); // O.K.!
```

Критерии: За неправильное определение - **–5**.
За отсутствие\неверный пример - **–5**.

7.
```
A_Constr
A_Constr
B_Constr
A_Copy
B_Destr
A_Destr
A_Destr
B&_Catch
B_Destr
A_Destr
```

Критерии: За одну ошибку – **7**,
За две ошибки – **3**,
За большее количество ошибок – **0**.

Если пропущен конструктор, за парный пропущенный деструктор не наказывать.

8. Исправления: 1) перед описанием поля *"y"* вставить ***"static"***,
2) добавить описание статического поля вне класса: int A::y;
Объяснение: статические функции не могут пользоваться нестатическими членами-данными.
Напечатается: **9 7 7 7 или 9 9 9 7**.

Критерии: Не найдена ошибка – отсутствие **static** - или неверное объяснение – **0**.
Пропущено описание статической переменной вне класса - **-5**.
Неверная печать - **-5**.
За указание ошибки там, где её нет - **-5.**

9. Модификация: 1) Например, добавить **virtual** перед f() в классе B,
2) Заключить в **try-catch** блок использование операций (и) **dynamic_cast**, используя **catch** (bad_cast){....}

Критерий: За отсутствие каждого исправления и каждое лишнее исправление - **-5**.

10.
```
template <class T>
int beforelast ( T & t){ // const не помешает
      if (t.size() < 2)
            return 0;
      typename T::const_reverse_iterator ri = t.rbegin ();
      ri++;
      return *ri;
}
```

Критерии: За пропуск ***typename*** - **-5**.
За использование неконстантного итератора - **-5**.
За ошибки в алгоритме - **-5**.
За другие ошибочки - **-2** за каждую.

Вариант 2_2014 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std.`**

1. При компиляции функции main () следующей программы фиксируется ошибка. Объясните, в чем она заключается, и исправьте ее, модифицируя только структуру B, ничего в ней не удаляя.

```
struct A {
      int x,y;
      A (int m, int n){ x = m; y = n;}
};
struct B {A a;};

int main () {
      B b;
      cout << b.a.y << '\n';
      return 0;
 }
```

2. Вычеркните неразрешимые вызовы функции f(...). Что напечатает получившаяся программа?

```
struct A {
      operator int () { return 1; }
};
void  f (double d, char c)   { cout << "f(double, char)\n";}
```

```
void f (double d, int j) { cout << "f(double, int)\n";}
void f (A a, const char * p) { cout << "f(A, const char *)\n";}
void f (int i, const char * p) { cout << "f(int, const char *)\n";}

int main () {
    A a ;
    f (a, 0);
    f (a, 'a');
    f ('a', 0);
    return 0;  }
```

3. C++11. Каково назначение служебного слова ***auto*** в C++11? Привести **пример** его использования.

4. Описать **класс** B и необходимые функции так, чтобы при компиляции заданной функции main () в ней не были обнаружены ошибки.

```
int main () {
        B b (1);
        cout << (b ++) ++ << endl;
        return 0;
}
```

5. Что напечатает программа?

```
struct B {
        virtual void f (int a = 0) { cout << a << "B::f (int) \n";}
        void g () { f ();   cout << "B::g () \n";}
};
class D: public B {
        virtual void f () { cout << "D::f()";}
        virtual void g () { f ();   cout << "D::g()\n";}
};
int main () {
        D d;
        B  b,  & rb = d;
        b. g ();
        rb. g ();     return 0;      }
```

6. Что такое функция-друг в языке Си++? Привести пример описания и использования функции-друга.

7. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
struct A {
       A () { cout << "A_Constr\n"; }
       A (const A & a) { cout << "A_Copy\n"; }
       ~A () { cout << "A_Destr\n"; }
};
struct B : A {
       B () { cout << "B_Constr\n"; }
       B (const B & a) { cout << "B_Copy\n"; }
       ~B () { cout << "B_Destr\n"; }
};

void g() { B bg; throw bg;}

int main () {
       try {  A a; g(); }
       catch ( A& ) { cout << "A&_Catch\n"; }
       catch ( B&) { cout << "B&_Catch\n"; }
       return 0;
}
```

8. Исправьте ошибки в нижеследующей программе, ничего в ней **не удаляя**.
Объясните, в чем заключаются исправленные ошибки.
Что напечатает программа после внесенных исправлений?

```
struct A {
        int y;
        int f () { return (y = 2); }
};
struct B : A {
         static int g () { return 9 + f(); }
};
int A::y;

int main () {
        B b,d;
        b.y = b.f();
        cout << B::f() << ' ' << B::y << ' ' << d.y << B::g() << endl;
        return 0; }
```

9. Модифицируйте программу, **ничего в ней не удаляя**, так, чтобы использование операции **dynamic_cast** было корректно, а программа в целом завершилась нормально (не аварийно).

```
struct B {
        void g1 () { cout << "B::g1()\n"; }
};
struct D:B {};

int main () {
        D d, * pd;
        B b, * pb = &b, * pbd = &d;
        pd = dynamic_cast <D*> (pbd);  pd -> g1();
        pd = dynamic_cast <D*> (pb);   pd -> g1();
        return 0; }
```

10. Написать **ОДНУ функцию (first_last)**, которая для своего аргумента-контейнера целочисленных элементов возвращает сумму первого и последнего элементов (если таковые имеются) или 0. Использование **first_last () должно быть корректно** в следующей функции:

```
int  f2 ( vector <int> & V,  const list <int> & L ) {
         return   first_last (V) + first_last (L);
}
```

Ответы_Вариант 2_2014

1.
Надо добавить в класс B, например, конструктор умолчания со списком инициализации для вызова конструктора класса A с двумя параметрами.

Критерии: Ошибка не найдена или неправильно объяснена – **0**.
Только исправление неверное или с ошибками **- 5**

2.

Неразрешимые вызовы – f (a, 0); и f ('a', 0);

Будет напечатано: f (double, char)

Критерии: За каждое неверное вычеркивание или неправильно вызванную функцию **– 5**.

3.

Описание *явно инициализируемой* переменной может содержать ключевое слово **auto:** при этом типом созданной переменной будет тип инициализирующего выражения.
Например, **auto** i = 7;

Критерии: По сути неверный ответ – **0**.
Отсутствие\неверный пример - **-5**.

4.

```
struct B {
        int n;
        B (int i) {n = i;}
        B& operator ++ (int) { B b = *this; n++;  return b; } // можно также вернуть int & на n !,
}                                                              // тогда не обязательно перегружать <<
ostream & operator << (ostream & s, const B& a) {
        s << a.n;
        return s;
}
```

Критерии: За каждую неверную\отсутствующую операцию **–5**.
За другие мелкие погрешности - **-2** за каждую.
За **typedef int** B – **0**.

5.

0 - B::f(int)
B::g()
0 - B::f(int)
B::g()

Критерий: Ошибки в ответе – **0**, иначе – **10**.
Случайные неточности, не связанные с виртуальностью - **-2**.

6.

Функция-друг класса – внешняя по отношению к классу функция, которой разрешен доступ к закрытым и защищенным членам класса.
Пример: ....

Критерий: Неверное определение - **0**.
Неверный\отсутствует пример **- -5**.

7.

```
A_Constr
A_Constr
B_Constr
A_Constr
B_Copy
B_Destr
A_Destr
A_Destr
A& Catch
B_Destr
A_Destr
```

Критерии: За каждую неверную (лишнюю или пропущенную) печать **-2**.

8.

Исправления: 1) перед описанием поля "*y*" вставить "***static***",
2) перед описанием функции f() вставить "***static***".

Объяснение: статические функции не могут пользоваться нестатическими членами класса.

Напечатается: **2 2 2 11**.

Критерии: Не найдена ошибка – отсутствие **static** – за каждую – **-5**.
Неверная печать - **-5**.
За указание ошибки там, где её нет - **-5.**

9.

Модификация: 1) Например, добавить **virtual** перед g1 () в классе B,
2) Вставить проверку на NULL возвращаемого результата операций (и) **dynamic_cast**.

Критерий: За отсутствие каждого исправления и каждое лишнее исправление - **-5**.

10.

```
template <class T>
int first_last ( T & t){ // const не помешает
      if (t.size() == 0)
            return 0;
      typename T::const_iterator bi = t.begin(), ei = t.end();
      ei--;
      return *bi + *ei;
}
```

Критерии: За пропуск ***typename*** - **-5**.
За использование неконстантного итератора - **-5**.
За ошибки в алгоритме - **-5**.
За другие ошибочки - **-2** за каждую.

Вариант 3_2014 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. При компиляции функции main () следующей программы фиксируется ошибка. Объясните, в чем она заключается, и исправьте ее, модифицируя только структуру B, ничего в ней не удаляя.

```
struct A {
      int y;
      A (int n){ y = n;}
};
struct B:A {
      int x;
      B(int m){x = m;}
};
int main () {
      B b(10);
      cout << b.y << '\n';
      return 0;
}
```

2. Вычеркните неразрешимые вызовы функции f(...). Что напечатает получившаяся программа?

```
struct A {  int n;
            A (int  i) { n = i; }
};
void f (char a, char b)  { cout << "f(char, char)\n";}
void f (A  a,  A  b)  { cout << "f(A, A)\n";}
void f (const int & r, char c)  { cout << "f(const int &, char)\n";}
void f (char c, const char * p)  { cout << "f(char, const char *)\n";}

int main () {
```

```
A a (1);
f ('a', 0);
f (a, 0);
f (1, 0);
return 0;   }
```

3. C++11. Каково специальное назначение идентификатора ***final*** в C++11? Привести по одному **примеру для каждого варианта** его специального использования.

4. Описать **класс** C и необходимые функции так, чтобы при компиляции заданной функции main () в ней не были обнаружены ошибки.

```
int main () {
        C c (3);
        cout << -- (-- c) << endl;
        return 0;
}
```

5. Что напечатает программа?

```
struct B {
        virtual void f (float a) {g ();  cout << a << " - B::f(float)\n";}
        virtual void g () { cout << "B::g()\n";}
};
class D: public B {
        virtual void f (double d) { cout << d << " - D::f(double)  ";}
        virtual void g () { f (4);   cout << "D::g()\n";}
};
int main () {
        D d;
        B  b,  & rb = d;
        b. f (2);
        rb. f (3);     return 0;     }
```

6. Что такое чистая виртуальная функция в языке Си++? Привести пример описания и использования такой функции.

7. Что будет выдано в стандартный канал вывода при работе следующей программы?

```
struct A {
       A () { cout << "A_Constr\n"; }
       A (const A & a) { cout << "A_Copy\n"; }
       ~A () { cout << "A_Destr\n"; }
};
struct B : A {
       B () { cout << "B_Constr\n"; }
       B (const B & a) { cout << "B_Copy\n"; }
       ~B () { cout << "B_Destr\n"; }
};

void g() { B bg; throw &bg;}

void f() { A af; g();}

int main () {
       try {  A a; f(); }
       catch ( A* ) { cout << "A*_Catch\n"; }
       catch ( B* ) { cout << "B*_Catch\n"; }
       return 0;
}
```

8. Исправьте ошибки в описаниях **классов A и B**, чтобы программа стала правильной.
Объясните, в чем заключаются исправленные ошибки.
Что напечатает программа после внесенных исправлений?

```
struct A {
        static int y;
        static int f()const { return y; }
};
struct B : A {
        virtual static int g () { return 4 * f(); }
};
int A::y = 3;
int main () {
        B b,d;
        b.y = b.f();
        cout << B::f() << ' ' << B::y << ' ' << d.y << B::g() << endl;
        return 0; }
```

9. Модифицируйте программу, **ничего в ней не удаляя**, так, чтобы использование операции **dynamic_cast** было корректно, а программа в целом завершилась нормально (не аварийно).

```
struct B {
        void ttt () { cout << "B::ttt () \n"; }
};
struct D : B { };

int main () {
        D d, * pd1, *pd2;
        B b, * pb = &b, * pbd = &d;
        pd1 = dynamic_cast < D* > (pbd);
        pd2 = dynamic_cast < D* > (pb);
        if ( typeid (*pd1) == typeid (*pd2) )    pb -> ttt () ;
        return 0; }
```

10. Написать **ОДНУ функцию (max_first_last)**, которая для своего аргумента-контейнера целочисленных элементов возвращает наибольшее из значений первого и последнего элементов (если таковые имеются) или 0. Использование **max_first_last () должно быть корректно** в следующей функции:

```
int  f3 (const vector <int> & V,  list <int> & L ) {
        return   max_first_last (V) + max_first_last (L);
}
```

Ответы_Вариант 3_2014

1. Надо добавить список инициализации в конструктор класса B для вызова конструктора класса A с одним параметром.

Критерии: Ошибка не найдена или неправильно объяснена – **0**.
Неверное\с ошибками только исправление **- 5**

2. Неразрешимый вызов – f ('a', 0);
Будет напечатано: f (A , A)
f (**const int** &, **char**)

Критерии: За каждое неверное вычеркивание или неправильно вызванную функцию **– 5**.

3. Идентификатор *final* обозначает следующее:

- ***- в описании классов*** - то, что они не могут быть базовыми для новых классов,
  ex: **struct** F **final** {}
- ***в описании виртуальных функций*** - то, что возможные производные классы (от рассматриваемого) не могут иметь виртуальные функции, которые бы замещали финальные функции;
  ex: **virtual void** g() **final**;

Критерии: За **каждое** неверное объяснение использования – **-5**.
За **каждый** неверный\отсутствующий пример использования – **-3**.

4.
```
struct C {
        int n;
        C (int i) {n = i;}
        C& operator -- ( ) { --n;  return *this; } // можно также вернуть int & на n!, тогда
}                                                  // тогда не обязательно перегружать <<
ostream & operator << (ostream & s, C& a) {
        s << a.n;
        return s;
}
```

Критерии: За каждую неверную\отсутствующую операцию –**5**.
За другие мелкие погрешности - **-2** за каждую.
За **typedef int** C – **0**.

5.
B::g()
2 - B::f(float)
4 - D::f(double)
D::g()
3 – B::f(float)

Критерий: Ошибки в ответе – **0**, иначе – **10**.
Случайные неточности, не связанные с виртуальностью - **-2**.

6. Чистая виртуальная функция – виртуальный метод класса вида:

virtual < тип> <имя_ф> (.....) = 0;

В примере чистая виртуальная функция должна быть переопределена в производном классе.

Критерий: Неверное определение - **0**.
За отсутствие\неверный - **-5**.

7.
```
A_Constr
A_Constr
A_Constr
B_Constr
B_Destr
A_Destr
A_Destr
A_Destr
A*_Catch
```

Критерии: За одну ошибку – **7**,
За две ошибки – **3**,
За большее количество ошибок – **0**.
Если пропущен конструктор, за парный пропущенный деструктор не наказывать.

8.

Исправления: 1) в описании функции f () убрать **"*const*",**
2) в описании функции g () убрать **"*virtual*".**

Объяснение: статические функции не могут быть виртуальными и константными.

Напечатается: **3 3 3 12**.

Критерии: Не найдены ошибки – за каждую – **-5**.
Неверная печать - **-5**.
За указание ошибки там, где её нет - **-5.**

9.

Модификация: 1) Например, добавить **virtual** перед ttt () в классе B,
2) Заключить в **try-catch** блок условный оператор в функции main () , используя **catch** (bad_typeid){....}

Критерий: За отсутствие каждого исправления и каждое лишнее исправление - **-5**.

10.

```
template <class T>
int max_first_last ( T & t){ // const не помешает
      if (t.size() == 0)
            return 0;
      typename T::const_iterator bi = t.begin(), ei = t.end();
      ei--;
      return *bi > *ei ? *bi : *ei;
}
```

Критерии: За пропуск ***typename*** - **-5**.
За использование неконстантного итератора - **-5**.
За ошибки в алгоритме - **-5**.
За другие ошибочки - **-2** за каждую.

2014 год
Экзамен

Вариант 1_2014 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to aS\,|Sb|aAb|\,\varepsilon$
$A \to aaAbb|aabb$
$A \to A$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы:

Ответ:
$L(G)$ =

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

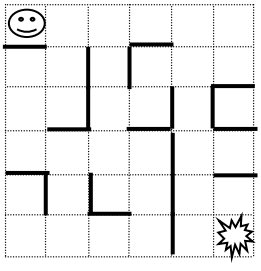
Класс П	$G \in$ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

(в) Тип языка: найти такое целое k, что $L(G)$ является языком типа k, но не языком типа k+1.
Ответ: k =

2. Является ли однозначной данная грамматика G, порождающая язык цепочек в алфавите {0,1}, в которых символов 0 и 1 поровну? Ответ обосновать.

G: $S \to 0S1S\,|1S0S\,|\,01BB|\,\varepsilon$
$B \to BB|B01$

3. Тесей должен пройти из левого верхнего угла лабиринта в правый нижний, чтобы убить Минотавра. Нить Ариадны оставляет след перемещений Тесея. Он умеет за один шаг перемещаться на клетку вниз (след обозначается символом *a*), вверх (след *b*) или вправо (след *c*). Влево не умеет. Каждая клетка посещается не более одного раза. Постройте грамматику, порождающую язык в алфавите {*a, b, c*} всевозможных путей Тесея, приводящих к цели. Например, цепочка *ccccaaaaac* приведет Тесея к Минотавру. В грамматике должно быть не более 4-х правил вывода, считая альтернативы.



4. Дан список слов: *в*, *и*, *исправление*, *обнаружение*, *ошибки*, *программа*. Составьте из него, употребив слова в нужном порядке и форме, определение термина (понятия) из раздела СП и назовите сам термин (он не входит в заданный список).

5. Можно ли считать утилиту *make* системы *Unix* компилятором? Обоснуйте ответ.

6. По заданной грамматике $G = \{\{a, b\}, \{B, S\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

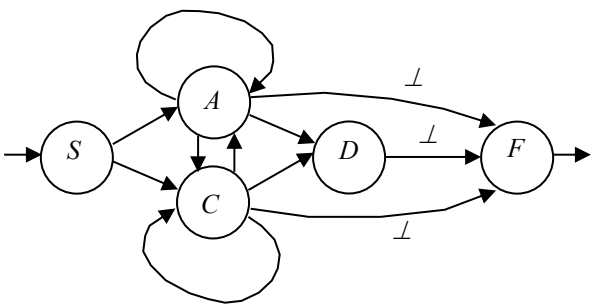
P: $S \to aBb \mid \varepsilon$
$B \to aB \mid bS \mid SS$

Ответ:

7. Даны 1) функция-анализатор на языке Си++ для леволинейной грамматики G_L:

```
bool scan_G ()
{   enum state { H, B, S, ER };  // множество состояний
    state CS= H;                 // CS -- текущее состояние
    do { gc ();   // считывает символ в глобальный объект c
         switch (CS) {
             case H: if ( c == 'a' ) CS = B;
                     else CS = ER; break;
             case B: if ( c == 'b' );
                 else if ( c == '⊥' ) CS = S;
                      else CS = ER; break;
                           }
         }
    while ( CS != S && CS != ER);
    return CS == S; // true, если CS != ER,  иначе false
}
```

2) заготовка диаграммы состояний праволинейной грамматики G_R:



(а) Расставить в заготовке диаграммы для G_R терминальные символы так, чтобы грамматики G_R и G_L были эквивалентны.

(б) Восстановить грамматику G_R. **Ответ:**

(в) Сколько состояний будет иметь конечный автомат, полученный алгоритмом детерминизации диаграммы для G_R ? **Ответ:____**

8. Дана КС-грамматика G, порождающая язык L.

Вставить в грамматику действия вида ⟨*cout* << "символы" ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(\omega, a^k b^{k+n}) \mid \omega \in L,\ k = |\omega|_a,\ n = |\omega|_b\}$.

G:

$S \to aA \mid bB \mid \varepsilon$

$A \to cA \mid S$

$B \to cB \mid S$

9. Дана грамматика G. Докажите, что метод рекурсивного спуска неприменим к ней. Можно ли вычеркнуть одно вхождение терминального символа в правилах грамматики так, чтобы к получившейся грамматике метод был бы применим?

G: $S \to aSb \mid Yb \mid bb$
$Y \to cSY \mid bd \mid \varepsilon$

10. Постройте ПОЛИЗ для фрагмента программы на Си. Префиксный ++ в польской записи обозначается как +#, постфиксный как #+.

```
for ( i = 0, j = 10;  i == j ? 0 : 1; --j) a += 2 < i++ > j;
```

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

Вариант 1_2014 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to aS \mid Sb \mid aAb \mid \varepsilon$
$A \to aaAbb \mid aabb$
$A \to A$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы:

Ответ:
$L(G) = \{ a^n b^m \mid m,n \geq 0 \}$

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

Класс П	$G \in$ П? (да/нет)
регулярные	**нет (-2)**
контекстно-зависимые	**нет (-2)**
контекстно-свободные	**да (-4)**
грамматики типа 0	**да (-4)**
неукорачивающие	**нет (-2)**

(в) Тип языка: найти такое целое k, что $L(G)$ является языком типа k, но не языком типа $k+1$.
Ответ: $k = 3$

Критерии: ошибка в формуле для (а): –4
за каждую неверную строчку в (б): см. таблицу
ошибка в (в): –3

2. Является ли однозначной данная грамматика G, порождающая язык цепочек в алфавите {0,1}, в которых символов 0 и 1 поровну? Ответ обосновать.

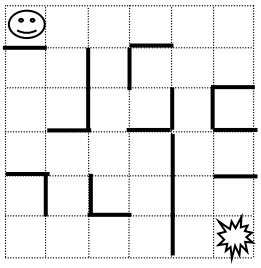
G: $S \to 0S1S \mid 1S0S \mid 01BB \mid \varepsilon$
$B \to BB \mid B01$

Ответ: Грамматика не приведенная. Символ B и правила его содержащие -- бесполезны и не участвуют в выводах терминальных цепочек из S, а значит не влияют на однозначность. Удалим их:
$S \to 0S1S \mid 1S0S \mid \varepsilon$

Имеем два различных левых вывода для *0101*:
$S \to 0S1S \to 01S \to 010S1S \to 0101S \to 0101$ и
$S \to 0S1S \to 01S0S1S \to 010S1S \to 0101S \to 0101$
Следовательно, грамматика неоднозначна.
Можно также привести два различных дерева вывода или два различных правых вывода для цепочки *0101*. Можно найти и другие цепочки, демонстрирующие нарушение определения однозначности.

Критерии: неверный ответ или отсутствует верное обоснование: 0

3. Тесей должен пройти из левого верхнего угла лабиринта в правый нижний, чтобы убить Минотавра. Нить Ариадны оставляет след перемещений Тесея. Он умеет за один шаг перемещаться на клетку вниз (след обозначается символом *a*), вверх (след *b*) или вправо (след *c*). Влево не умеет. Каждая клетка посещается не более одного раза. Постройте грамматику, порождающую язык в алфавите {*a*, *b, c*} всевозможных путей Тесея, приводящих к цели. Например, цепочка *ccccaaaaac* приведет Тесея к Минотавру. В грамматике должно быть не более 4-х правил вывода, считая альтернативы.



Ответ: Существует всего четыре возможных пути при заданных ограничениях: *ccccaaaaac, ccccaaaaca, ccaacbcaaaac, ccaacbcaaaca.* Поэтому возможна следующая регулярная грамматика:
$S \to ccccaaaaac \mid ccccaaaaca \mid ccaacbcaaaac \mid ccaacbcaaaca$

Возможны и другие варианты, например:
$S \to ccccaaaaA \mid ccaacbcaaaA$
$A \to ac \mid ca$

Критерии: за каждый неверный (отсутствующий) путь в порождаемом языке: –5

4. Дан список слов: *в*, *и*, *исправление*, *обнаружение*, *ошибки*, *программа*. Составьте из него, употребив слова в нужном порядке и форме, определение термина (понятия) из раздела СП и назовите сам термин (он не входит в заданный список).
Ответ: *Отладка – обнаружение и исправление ошибок в программе.*
Критерии: неверный ответ: 0

5. Можно ли считать утилиту *make* системы *Unix* компилятором? Обоснуйте ответ.
Ответ: нет. *make* не переводит текст из *Makefile* в исполняемый файл, а интерпретирует его.
Критерии: неверный (не обоснованный) ответ: 0

6. По заданной грамматике $G = \{\{a, b\}, \{B, S\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

P: $S \to aBb \mid \varepsilon$
$B \to aB \mid bS \mid SS$

Ответ: $S' \to S \mid \varepsilon$
$S \to aBb \mid ab$
$B \to a \mid aB \mid bS \mid b \mid SS \mid S$

Критерии: за каждое лишнее (отсутствующее) правило: -5

7. Даны 1) функция-анализатор на языке Си++ для леволинейной грамматики G_L:

```
bool scan_G ()
{   enum state { H, B, S, ER };  // множество состояний
    state CS= H;                 // CS -- текущее состояние
    do { gc ();   // считывает символ в глобальный объект c
         switch (CS) {
```

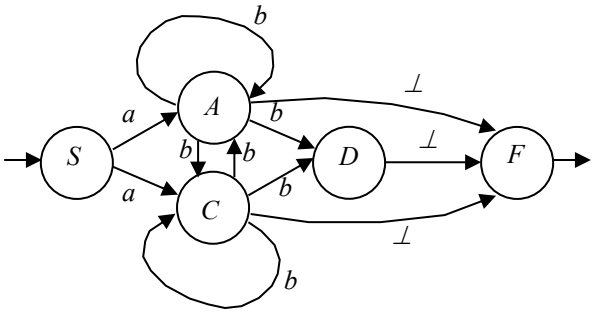
```
                case H: if ( c == 'a' ) CS = B;
                        else CS = ER; break;
                case B: if ( c == 'b' );
                     else if ( c == '⊥' ) CS = S;
                         else CS = ER; break;
                              }
            }
        while ( CS != S && CS != ER);
        return CS == S; // true, если CS != ER,  иначе false
    }
```

Восстановим грамматику G_L : $S \to B \perp$

$B \to Bb \mid a$

$L(G_L) = \{ a\, b^m \perp \mid m \geq 0 \}$

2) заготовка диаграммы состояний праволинейной грамматики G_R:



(а) Расставить в заготовке диаграммы для G_R терминальные символы так, чтобы грамматики G_R и G_L были эквивалентны.

Ответ: Переход из S в A или C возможен только по a согласно формуле языка. Далее все незаполненные дуги помечаются b, так как a стоит только в начале цепочки, а $\perp$ -- только в конце.

(б) Восстановить грамматику G_R. **Ответ:**

$S \to aA \mid aC$

$A \to bA \mid bD \mid bC \mid \perp$

$C \to bC \mid bD \mid bA \mid \perp$

$D \to \perp$

(в) Сколько состояний будет иметь конечный автомат, полученный алгоритмом детерминизации диаграммы для G_R ? **Ответ: 4**

Состояния ДКА: $\{S\}$, $\{A,C\}$,$\{A,C,D\}$, $\{F\}$

Критерии: (а) нет эквивалентности: –3
(пустых переходов не должно быть, в определении КА их нет)

(б) восстановленная G_R не соответствует диаграмме студента (сама диаграмма может быть ошибочной, этот пункт оценивается независимо) : -4

(в) неверное число: –3 (обоснование не обязательно)

8. Дана КС-грамматика G, порождающая язык L.

Вставить в грамматику действия вида ⟨*cout* << "символы" ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(x, a^k b^{k+n}) \mid x \in L\,,\ k = |x|_a\,,\ n = |x|_b\}$.

G:

$S \to aA \mid bB \mid \varepsilon$

$A \to cA \mid S$

$B \to cB \mid S$

Ответ:

$S \to a\langle a\rangle A\langle b\rangle \mid bB\langle b\rangle \mid \varepsilon$	$S \to aA\langle b\rangle \mid bB\langle b\rangle \mid \varepsilon$	$S \to aA \mid bB \mid \varepsilon$ Есть еще варианты.
$A \to cA \mid S$	$A \to cA \mid \langle a\rangle S$	$A \to cA \mid \langle a\rangle S\langle b\rangle$
$B \to cB \mid S$	$B \to cB \mid S$	$B \to cB \mid S\langle b\rangle$

Запись ⟨*cout* << *"a"* ;⟩ сокращается до ⟨*a*⟩

Критерии: за любую ошибку: 0

9. Дана грамматика G. Докажите, что метод рекурсивного спуска неприменим к ней. Можно ли вычеркнуть один терминальный символ в правилах грамматики так, чтобы к получившейся грамматике метод был бы применим?

G: $S \to aSb \mid Yb \mid bb$
$Y \to cSY \mid dd \mid \varepsilon$

Ответ: *first* (*Yb*)∩ *first* (*bb*)={*b*}≠∅ . Нет: какой бы символ ни вычеркнуть, метод будет неприменим. Обоснование не обязательно. (Пустая цепочка, обозначенная *ε, не является* терминальным символом –ее вычеркивать нельзя.)
Критерии: ошибочное или отсутствующее доказательство: -7
ошибочный или отсутствующий ответ на второй вопрос: -3

10. Постройте ПОЛИЗ для фрагмента программы на Си. Префиксный ++ в польской записи обозначается как +#, постфиксный как #+.

for *(i=0, j=10; i==j?0:1; -- j) a+=2<i++>j;*

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
&i	0	=	;	&j	10	=	;	i	j	==	17	!F	0	18	!	1

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
36	!F	26	!	&j	-#	9	!	&a	2	&i	#+	<	j	>	+=	22	!				

Критерии: за каждую ошибку: -4

Вариант 2_2014 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика *G*:

$S \to cA \mid cAb \mid cb \mid \varepsilon$
$cAb \to cAbb \mid cb \mid ccAb$
$ccAbb \to ccAbb$

(а) Описать язык *L*(*G*) в виде теоретико-множественной формулы:

Ответ:
L(*G*) =

(б) Каким из перечисленных классов грамматик принадлежит *G*? **Ответ:**

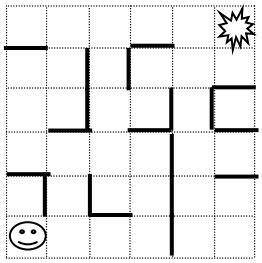
Класс П	*G* ∈ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

(в) Тип языка: найти такое целое *k*, что *L(G)* является языком типа *k*, но не языком типа *k*+1.
Ответ: *k* =

2. Является ли однозначной данная грамматика *G* , порождающая язык цепочек в алфавите {*a*,*b*}? Ответ обосновать.

G: $S \to aSaS \mid aBa \mid aSa \mid A$
$B \to BB \mid Ba$
$A \to b \mid \varepsilon$

3. Баба-Яга дала Иванушке клубочек, который через лесной лабиринт проведет его к сундуку, где находится смерть Кащеева. Клубочек (а за ним и Иванушка) умеет за один шаг перемещаться на клетку вниз (оставляемый след обозначается символом *a*), вверх (след *b*) или вправо (след *c*). Влево не умеет. Каждая клетка посещается не более одного раза. Постройте грамматику, порождающую язык в алфавите {*a, b, c*} всевозможных путей Иванушки, приводящих к цели. Иванушка начинает путь из левого нижнего угла. Цель – в правом верхнем. Например, цепочка *cbbcbbbccc* приведет Иванушку к цели. В грамматике должно быть не более 4-х правил вывода, считая альтернативы.



4. Дан список слов: *входные*, *заранее*, *данные*, *для*, *известный, программа, работа*, *результат, с*. Составьте из него, употребив слова в нужном порядке и форме, определение термина (понятия) из раздела СП и назовите сам термин (он не входит в заданный список).

5. Можно ли считать утилиту *make* системы *Unix* интерпретатором? Обоснуйте ответ.

6. По заданной грамматике $G = \{\{a, c\}, \{B, S\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

Ответ:

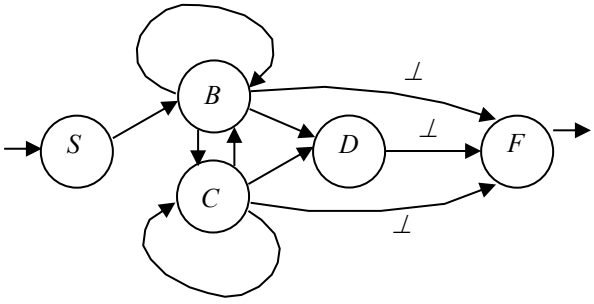
P: $S \to aBc \mid \varepsilon$

$B \to aB \mid cS \mid SaS$

7. Даны 1) функция-анализатор на языке Си++ для леволинейной грамматики G_L:

```
bool scan_G ()
{   enum state { H, A, S, ER };  // множество состояний
    state CS= H;                 // CS -- текущее состояние
    do { gc (); // считывает символ в глобальный объект c
         switch (CS) {
             case H: if ( c == 'a'||c ==’b’ ) CS = A;
                     else CS = ER; break;
             case A: if ( c == 'a' );
                   else if ( c == '⊥' ) CS = S;
                      else CS = ER; break;
                           }
         }
    while ( CS != S && CS != ER);
    return CS == S; // true, если CS != ER,  иначе false
}
```

2) заготовка диаграммы состояний праволинейной грамматики G_R:



(а) Расставить в заготовке диаграммы для G_R терминальные символы так, чтобы грамматики G_R и G_L были эквивалентны.

(б) Восстановить грамматику G_R. **Ответ:**

(в) Сколько состояний будет иметь конечный автомат, полученный алгоритмом детерминизации диаграммы для G_R ? **Ответ:**____

8. Дана КС-грамматика G, порождающая язык L.

Вставить в грамматику действия вида ⟨*cout* << "символы" ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(\omega, a^k d^{k-n}) \mid \omega \in L,\ k = |\omega|,\ n = |\omega|_d\}$.

G:

$S \to aA \mid dD \mid \varepsilon$

$A \to cA \mid S$

$D \to cD \mid S$

9. Дана грамматика G. Докажите, что метод рекурсивного спуска неприменим к ней. Можно ли вычеркнуть одно вхождение терминального символа в правилах грамматики так, чтобы к получившейся грамматике метод был бы применим?

G: $S \to aSb \mid Xa \mid bb$
$X \to cSX \mid bd \mid \varepsilon$

10. Постройте ПОЛИЗ для фрагмента программы на Си. Префиксный ++ в польской записи обозначается как +#, постфиксный как #+.

```
i = 0, j = 10; do a += 2 < i++ > --j; while ( i == j ? 0 : 1 );
```

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

Вариант 2_2014 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to cA \mid cAb \mid cb \mid \varepsilon$
$cAb \to cAbb \mid cb \mid ccAb$
$ccAbb \to ccAbb$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы:

Ответ:
$L(G) = \{ c^n b^m \mid m,n \geq 2 \} \cup \{ c^k b^k \mid 0 \leq k \leq 1 \}$
Или:
$\{ c^n b^m \mid m,n \geq 2 \} \cup \{\varepsilon\} \cup \{cb\}$

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

Класс П	$G \in$ П? (да/нет)
регулярные	**нет (-2)**
контекстно-зависимые	**нет (-2)**
контекстно-свободные	**нет (-2)**
грамматики типа 0	**да (-4)**
неукорачивающие	**нет (-2)**

(в) Тип языка: найти такое целое k, что $L(G)$ является языком типа k, но не языком типа $k+1$.
Ответ: $k = 3$

Критерии: ошибка в формуле для (а): –4
за каждую неверную строчку в (б): см. таблицу
ошибка в (в): –3

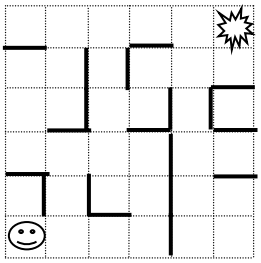
2. Является ли однозначной данная грамматика G, порождающая язык цепочек в алфавите $\{a,b\}$? Ответ обосновать.

G: $S \to aSaS \mid aBa \mid aSa \mid A$
$B \to BB \mid Ba$
$A \to b \mid \varepsilon$

Грамматика не приведенная. Символ B и правила его содержащие -- бесполезны и не участвуют в выводах терминальных цепочек из S, а значит не влияют на однозначность. Удалим их:
$S \to aSaS \mid aSa \mid A$
$A \to b \mid \varepsilon$
Имеем два различных левых вывода для *aa*:
$S \to aSaS \to aAaS \to aaS \to aaA \to aa$ и
$S \to aSa \to aAa \to aa$
Следовательно, грамматика неоднозначна.
Можно также привести два различных дерева вывода или два различных правых вывода для цепочки *aa*. Можно найти и другие цепочки, демонстрирующие нарушение определения однозначности.

Критерии: неверный ответ или отсутствует верное обоснование: 0

3. Баба-Яга дала Иванушке клубочек, который через лесной лабиринт проведет его к сундуку, где находится смерть Кащеева. Клубочек (а за ним и Иванушка) умеет за один шаг перемещаться на клетку вниз (оставляемый след обозначается символом *a*), вверх (след *b*) или вправо (след *c*). Влево не умеет. Каждая клетка посещается не более одного раза. Постройте грамматику, порождающую язык в алфавите {*a*, *b, c*} всевозможных путей Иванушки, приводящих к цели. Иванушка начинает путь из левого нижнего угла. Цель – в правом верхнем. Например, цепочка *cbbcbbbccc* приведет Иванушку к цели. В грамматике должно быть не более 4-х правил вывода, считая альтернативы.



Ответ: Существует всего четыре возможных пути при заданных ограничениях: *cbbcbbbccc, cbbcbbbccacb, cbbcbcbccb, cbbcbcbcac.* Поэтому возможна следующая регулярная грамматика:
$S \to cbbcbbbccc, \mid cbbcbbbccacb \mid cbbcbcbccb \mid cbbcbcbcac$

Критерии: за каждый неверный (отсутствующий) путь в порождаемом языке: –5

4. Дан список слов: *входной*, *заранее*, *данные*, *для*, *известный, программа, работа*, *результат, с*. Составьте из него, употребив слова в нужном порядке и форме, определение термина (понятия) из раздела СП и назовите сам термин (он не входит в заданный список).

Ответ: *Тест – входные данные для программы с заранее известным результатом работы.*
Критерии: неверный ответ: 0

5. Можно ли считать утилиту *make* системы *Unix* интерпретатором? Обоснуйте ответ.
Ответ: да. *make* интерпретирует текст из *Makefile*, запуская необходимые процессы аналогично командному интерпретатору *shell*.
Критерии: неверный (не обоснованный) ответ: 0

6. По заданной грамматике $G = \{\{a, c\}, \{B, S\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

P: $S \to aBc \mid \varepsilon$
$B \to aB \mid cS \mid SaS$

Ответ: $S' \to S \mid \varepsilon$
$S \to aBc$
$B \to aB \mid cS \mid c \mid aS \mid Sa \mid SaS \mid a$

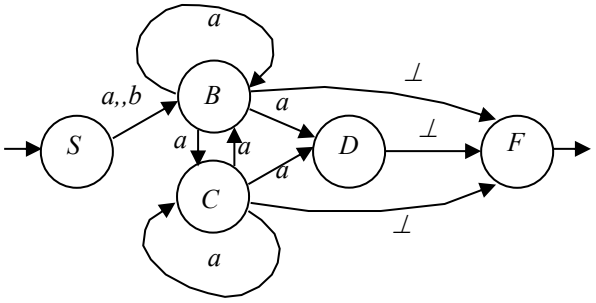
Критерии: за каждое лишнее (отсутствующее) правило: -5

7. Даны 1) функция-анализатор на языке Си++ для леволинейной грамматики G_L:

```
bool scan_G ()
{   enum state { H, A, S, ER };  // множество состояний
```

```
        state CS= H;                    // CS -- текущее состояние
        do { gc (); // считывает символ в глобальный объект c
             switch (CS) {
                 case H: if ( c == 'a'||c ==’b’ ) CS = A;
                         else CS = ER; break;
                 case A: if ( c == 'a' );
                       else if ( c == '⊥' ) CS = S;
                          else CS = ER; break;
                               }
             }
        while ( CS != S && CS != ER);
        return CS == S; // true, если CS != ER,  иначе false
}
```

2) заготовка диаграммы состояний праволинейной грамматики G_R:



(а) Расставить в заготовке диаграммы для G_R терминальные символы так, чтобы грамматики G_R и G_L были эквивалентны.

(б) Восстановить грамматику G_R. **Ответ:**

$S \to aB|\ bB$

$B \to aB\ |aC|\ aD|\perp$

$C \to aC|\ aD|\ aB|\perp$

$D \to \perp$

(в) Сколько состояний будет иметь конечный автомат, полученный алгоритмом детерминизации диаграммы для G_R ? **Ответ: 4**

Состояния ДКА: $\{S\}$, $\{B\}$,$\{B,C,D\}$, $\{F\}$

Критерии: (а) нет эквивалентности: –3
(пустых переходов не должно быть, в определении КА их нет)

(б) восстановленная G_R не соответствует диаграмме студента (сама диаграмма может быть ошибочной, этот пункт оценивается независимо) : -4

(в) неверное число: –3 (обоснование не обязательно)

8. Дана КС-грамматика G, порождающая язык L.

Вставить в грамматику действия вида ⟨*cout* << "символы" ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(x,\ a^k d^{k-n}))\ |\ x \in L\ ,\ k = |x|,\ \ n = |x|_d\ \}$.

G:

$S \to aA\ |dD\ |\ \varepsilon$

$A \to cA\ |\ S$

$D \to cD|\ S$

Ответ: $S \to a\langle a\rangle A\langle d\rangle\ |d\langle a\rangle D\ |\ \varepsilon$ $S \to aA\langle d\rangle\ |d\langle a\rangle D\ |\ \varepsilon$ Есть еще варианты.

$A \to c\langle a\rangle A\langle d\rangle\ |\ S$ $A \to c\langle a\rangle A\ \langle d\rangle|\ S$

$D \to c\langle a\rangle D\langle d\rangle|\ S$ $D \to c\langle a\rangle D\langle d\rangle|\ \langle a\rangle S$

Запись ⟨*cout* << *"a"* ;⟩ сокращается до ⟨*a*⟩

Критерии: за любую ошибку: 0

9. Дана грамматика G. Докажите, что метод рекурсивного спуска неприменим к ней. Можно ли вычеркнуть один терминальный символ в правилах грамматики так, чтобы к получившейся грамматике метод был бы применим?

G: $S \to aSb\ |\ Xa\ |\ bb$

$X \to cSX\ |bd\ |\ \varepsilon$

Ответ: *first* (*Xa*)∩ *first* (*aSb*)={*a*}≠∅ . Нет: какой бы символ ни вычеркнуть, метод будет неприменим. Обоснование не обязательно. (Пустая цепочка, обозначаемая *ε, не является* терминальным символом – ее нельзя вычеркивать)
Критерии: ошибочное или отсутствующее доказательство: -7
ошибочный или отсутствующий ответ на второй вопрос: -3

10. Постройте ПОЛИЗ для фрагмента программы на Си. Префиксный ++ в польской записи обозначается как +#, постфиксный как #+.

i=0, j=10; ***do*** *a+=2<i++>--j;* ***while*** *(i==j?0:1) ;*

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
&i	0	=	;	&j	10	=	;	&a	2	&i	#+	<	&j	-#	>	+=

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
;	i	j	==	27	!F	0	28	!	1	32	!F	9	!								

Критерии: за каждую ошибку: -4
Вариант 3_2014 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика *G*:

$S \to Sc|aSc|\varepsilon|aAc$
$aAc \to aEAc|aac$
$aE \to Ea$
$E \to a$

(а) Описать язык *L*(*G*) в виде теоретико-множественной формулы:

Ответ:
L(*G*) =

(б) Каким из перечисленных классов грамматик принадлежит *G*? **Ответ:**

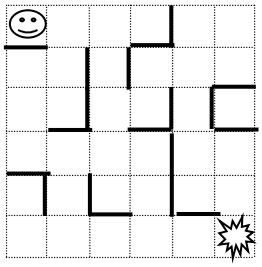
Класс П	*G* ∈ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

(в) Тип языка: найти такое целое *k*, что *L(G)* является языком типа *k*, но не языком типа *k*+1.
Ответ: *k* =

2. Является ли однозначной данная грамматика *G*, порождающая язык цепочек в алфавите {*b*,*c*}? Ответ обосновать.

G: $S \to ASS\,|SSA|\,B\,|\,A$
$B \to bB|cB|\,\varepsilon$

3. Колобок должен пройти через лесной лабиринт из левого верхнего угла в правый нижний, чтобы спеть песенку лисе. Он умеет за один шаг перемещаться на клетку вниз (след обозначается символом *a*), вверх (след *b*) или вправо (след *c*). Влево не умеет. Каждая клетка посещается не более одного раза. Постройте грамматику, порождающую язык в алфавите {*a*, *b, c*} всевозможных путей колобка, приводящих к лисе. Например, цепочка *ccaaaacacc* приведет колобка к цели. В грамматике должно быть не более 4-х правил вывода, считая альтернативы.



4. Дан список слов: *на*, *правильность*, *проверка*, *программа*, *помощь*, *с, тесты*. Составьте из него, употребив слова в нужном порядке и форме, определение термина (понятия) из раздела СП и назовите сам термин (он не входит в заданный список).

5. Можно ли язык, на котором записывается задание для утилиты *make* системы *Unix,* считать интерпретируемым языком? Обоснуйте ответ.

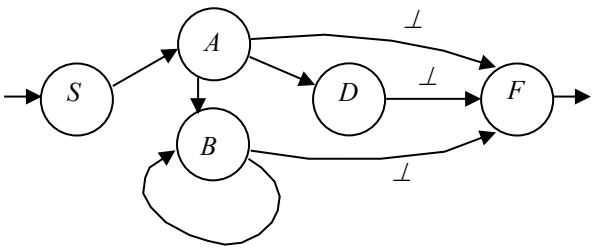
6. По заданной грамматике $G = \{\{d, b\}, \{B, S\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

P: $S \to dBb \mid S \mid \varepsilon$ **Ответ:**
$B \to dB \mid bS \mid SSd$

7. Даны 1) функция-анализатор на языке Си++ для леволинейной грамматики G_L:

```
bool scan_G ()
{   enum state { H, C, S, ER };  // множество состояний
    state CS= H;                 // CS -- текущее состояние
    do { gc ();   // считывает символ в глобальный объект c
         switch (CS) {
             case H: if ( c == 'a' ) CS = C;
                     else CS = ER; break;
             case C: if ( c == 'c' );
                   else if ( c == '⊥' ) CS = S;
                     else CS = ER; break;
                          }
         }
    while ( CS != S && CS != ER);
    return CS == S; // true, если CS != ER,  иначе false
}
```

2) заготовка диаграммы состояний праволинейной грамматики G_R:



(а) Расставить в заготовке диаграммы для G_R терминальные символы так, чтобы грамматики G_R и G_L были эквивалентны.

(б) Восстановить грамматику G_R. **Ответ:**

(в) Сколько состояний будет иметь конечный автомат, полученный алгоритмом детерминизации диаграммы для G_R ? **Ответ:____**

8. Дана КС-грамматика G, порождающая язык L.

Вставить в грамматику действия вида ⟨*cout* << "символы" ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(\omega, a^{2k}b^{k-n}) \mid \omega \in L,\ k = |\omega|,\ n = |\omega|_a\}$.

G:

$S \to aA \mid bB \mid \varepsilon$

$A \to cA \mid S$

$B \to cB \mid S$

9. Дана грамматика G. Докажите, что метод рекурсивного спуска неприменим к ней. Можно ли вычеркнуть одно вхождение терминального символа в правилах грамматики так, чтобы к получившейся грамматике метод был бы применим?

G: $S \to aSb \mid YD \mid db$
$Y \to cSY \mid bd \mid \varepsilon$

$D \to dd$

10. Постройте ПОЛИЗ для фрагмента программы на Си. Префиксный ++ в польской записи обозначается как +#, постфиксный как #+.

```
i = 0, j = 10; while ( i == j ? 0 : 1 ) a += 2 < ++i > j--;
```

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

Вариант 3_2014 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Дана грамматика G:

$S \to Sc|aSc|\varepsilon|aAc$
$aAc \to aEAc|aac$
$aE \to Ea$
$E \to a$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы:

Ответ:
$L(G) = \{ a^n c^m \mid m, n \geq 1 \} \cup \{ c^k \mid k \geq 0 \}$

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

Класс П	$G \in$ П? (да/нет)
регулярные	**нет (-2)**
контекстно-зависимые	**нет (-2)**
контекстно-свободные	**нет (-2)**
грамматики типа 0	**да (-4)**
неукорачивающие	**нет (-2)**

(в) Тип языка: найти такое целое k, что $L(G)$ является языком типа k, но не языком типа k+1.
Ответ: k =3

Критерии: ошибка в формуле для (а): –4
за каждую неверную строчку в (б): см. таблицу
ошибка в (в): –3

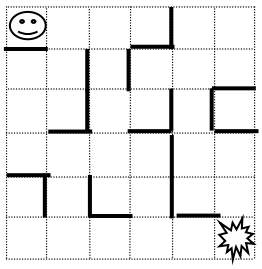
2. Является ли однозначной данная грамматика G, порождающая язык цепочек в алфавите $\{b,c\}$? Ответ обосновать.

G: $S \to ASS|SSA|B|A$
$B \to bB|cB|\varepsilon$

Грамматика не приведенная. Символ A и правила его содержащие -- бесполезны и не участвуют в выводах терминальных цепочек из S, а значит не влияют на однозначность. Удалим их:
$S \to B$
$B \to bB|cB|\varepsilon$
Заметим, что правило $S \to B$ применяется в любом выводе на первом шаге, а на следующих шагах работают правила из подграмматики $B \to bB|cB|\varepsilon$.Данная подграмматика автоматная праволинейная и ей соответствует детерминированная диаграмма состояний, которая «строит» левый выовод единственным способом. Следовательно, подграмматика $B \to bB|cB|\varepsilon$ однозначна, и исходная грамматика тоже однозначна.
Другое обоснование: к преобразованной приведенной грамматике применим метод рекурсивного спуска, следовательно, она однозначна. (Метод РС неприменим к неоднозначным грамматикам).
Возможны и другие неформальные рассуждения, подтверждающие единственность вывода в данной грамматике для любой цепочке, а следовательно, единственность левого, правого выводов и дерева вывода.

Критерии: неверный ответ или отсутствует верное обоснование: 0

3. Колобок должен пройти через лесной лабиринт из левого верхнего угла в правый нижний, чтобы спеть песенку лисе. Он умеет за один шаг перемещаться на клетку вниз (след обозначается символом *a*), вверх (след *b*) или вправо (след *c*). Влево не умеет. Каждая клетка посещается не более одного раза. Постройте грамматику, порождающую язык в алфавите {*a*, *b, c*} всевозможных путей колобка, приводящих к лисе. Например, цепочка *ccaaaacacc* приведет колобка к цели. В грамматике должно быть не более 4-х правил вывода, считая альтернативы.



Ответ: Существует всего четыре возможных пути при заданных ограничениях: *ccaaacaacc, ccaaaacacc, ccaacbcaaaca, ccaacbcaacaa.* Поэтому возможна следующая регулярная грамматика:
$S \rightarrow ccaaacaacc \mid ccaaaacacc \mid ccaacbcaaaca \mid ccaacbcaacaa$

Критерии: за каждый неверный (отсутствующий) путь в порождаемом языке: –5

4. Дан список слов: *на*, *правильность*, *проверка*, *программа*, *помощь*, *с, тесты*. Составьте из него, употребив слова в нужном порядке и форме, определение термина (понятия) из раздела СП и назовите сам термин (он не входит в заданный список).

Ответ: *Тестирование – проверка программы на правильность с помощью тестов.*
Критерии: неверный ответ: 0

5. Можно ли язык, на котором записывается задание для утилиты *make* системы *Unix,* считать интерпретируемым языком? Обоснуйте ответ.
Ответ: да. *make* интерпретирует текст на языке для *Makefile*, запуская необходимые процессы аналогично командному интерпретатору *shell*.
Критерии: неверный (не обоснованный) ответ: 0

6. По заданной грамматике $G = \{\{d, b\}, \{B, S\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

P: $S \rightarrow dBb \mid S \mid \varepsilon$
$B \rightarrow dB \mid bS \mid SSd$

Ответ: $S' \rightarrow S \mid \varepsilon$
$S \rightarrow dBb \mid S$
$B \rightarrow dB \mid bS \mid b \mid d \mid Sd \mid SSd$

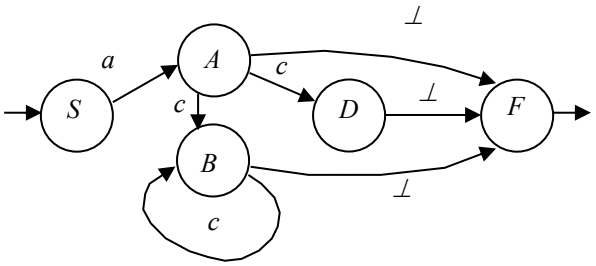
Критерии: за каждое лишнее (отсутствующее) правило: -5

7. Даны 1) функция-анализатор на языке Си++ для леволинейной грамматики G_L:

```
bool scan_G ()
{   enum state { H, C, S, ER };  // множество состояний
    state CS= H;                 // CS -- текущее состояние
    do { gc ();   // считывает символ в глобальный объект c
         switch (CS) {
             case H: if ( c == 'a' ) CS = C;
                     else CS = ER; break;
             case C: if ( c == 'c' );
                   else if ( c == '⊥' ) CS = S;
                      else CS = ER; break;
                          }
         }
    while ( CS != S && CS != ER);
```

```
        return CS == S; // true, если CS != ER,  иначе false
    }
```

2) заготовка диаграммы состояний праволинейной грамматики G_R:



(а) Расставить в заготовке диаграммы для G_R терминальные символы так, чтобы грамматики G_R и G_L были эквивалентны.

(б) Восстановить грамматику G_R. **Ответ:**

$S \to aA$

$A \to cB \mid cD \mid \perp$

$B \to cB \mid \perp$

$D \to \perp$

(в) Сколько состояний будет иметь конечный автомат, полученный алгоритмом детерминизации диаграммы для G_R ? **Ответ: 5**

Состояния ДКА: $\{S\}$, $\{A\}$,$\{B,D\}$, $\{B\}$, $\{F\}$

Критерии: (а) нет эквивалентности: –3
(пустых переходов не должно быть, в определении КА их нет)
(б) восстановленная G_R не соответствует диаграмме студента (сама диаграмма может быть ошибочной, этот пункт оценивается независимо) : -4
(в) неверное число: –3 (обоснование не обязательно)

5. Дана КС-грамматика G, порождающая язык L.

Вставить в грамматику действия вида ⟨*cout* << "символы" ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(x, a^{2k} b^{k-n}) \mid x \in L\ ,\ k = |x|,\ n = |x|_a\}$.

G:

$S \to aA \mid bB \mid \varepsilon$

$A \to cA \mid S$

$B \to cB \mid S$

Ответ: $S \to a\langle aa\rangle A \mid b\langle aa\rangle B\langle b\rangle \mid \varepsilon$ | $S \to aA \mid b\langle aa\rangle B\langle b\rangle \mid \varepsilon$ Есть еще варианты.

$A \to c\langle aa\rangle A\langle b\rangle \mid S$ | $A \to c\langle aa\rangle A\ \langle b\rangle \mid \langle aa\rangle S$

$B \to c\langle aa\rangle B\langle b\rangle \mid S$ | $B \to c\langle aa\rangle B\langle b\rangle \mid S$

Запись ⟨*cout* << "*a*" ;⟩ сокращается до ⟨*a*⟩

Критерии: за любую ошибку: 0

9. Дана грамматика G. Докажите, что метод рекурсивного спуска неприменим к ней. Можно ли вычеркнуть один терминальный символ в правилах грамматики так, чтобы к получившейся грамматике метод был бы применим?

G: $S \to aSb \mid YD \mid db$
$Y \to cSY \mid bd \mid \varepsilon$
$D \to dd$

Ответ: $first(YD) \cap first(db) = \{d\} \neq \varnothing$. Нет: какой бы символ ни вычеркнуть, метод будет неприменим. Обоснование не обязательно. (Пустая цепочка, обозначаемая ε, *не является* терминальным символом – ее нельзя вычеркивать).
Критерии: ошибочное или отсутствующее доказательство: -7
ошибочный или отсутствующий ответ на второй вопрос: -3

10. Постройте ПОЛИЗ для фрагмента программы на Си. Префиксный ++ в польской записи обозначается как +#, постфиксный как #+.
i=0, j=10; ***while*** *(i==j?0:1) a+=2<++i>j--;*

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&i	0	=	;	&j	10	=	;	i	j	==	17	!F	0	18	!	1

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
32	!F	&a	2	&i	+#	<	&j	#-	>	+=	;	9	!								

Критерии: за каждую ошибку: -4

2015 год
Коллоквиум

Вариант 1_2015 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. Приведите фрагмент программы с примером ***ошибочного*** обращения к видимой, но недоступной функции.

2. Что будет напечатано данной программой? Можно ли удалить хотя бы одну операцию разрешения области видимости так, чтобы печать не изменилась?

```
namespace N { int f= 2;
              int g=-2;
}
namespace M { int f= 3;
              int g=-3;
              int h=0;
}
int f=-5;
```

```
int main() { int f=1;
    cout<<::f<<N::f<<M::f<<M::h<<endl;
  using namespace N;
    cout<<f<<g<<M::h<<endl;
  using namespace M;
    cout<<f<<M::g<<h<<endl;
    return 0;
}
```

3. Можно ли продемонстрировать параметрический полиморфизм с помощью оператора-выражения `a+(b+a);` ? Обоснуйте ответ. Перечислите другие виды полиморфизма.

4. Данная программа выводит на печать несколько строк. Вычеркните ***фрагменты*** из функции ***main*** так, чтобы напечаталось верное утверждение об ООП и C++.

```
class A {
public:
   A() { cout<<"абстрактный "; }
   virtual ~A() {cout<<  endl;}
   void t(int n){cout<<"конструктор ";}
   virtual void t(){cout<<"метод ";}
};
class B : public A {
public:
    B(){}
    B(int x) {cout<<"класс ";}
    void t() { cout<<"тип ";}
    void t(int n) { cout<<"интерфейс
";}
    ~B() {cout<< "данных";}
};
```

```
int main() {
    cout<<"класс, не содержащий открытых \
          членов-данных, -- это " ;
    A * qa= new A;
    qa->t(5);
    delete qa;
    qa=new B;
    qa->A::t();
    delete qa;
    qa= new B;
    { A & ra= *qa;
      ra.t();
    }
    delete qa;
    B * qb= new B;
    qb->t(5);
```

```
    delete qb;
    B b(5);
}
```

5. Может ли одна и та же функция одновременно участвовать в перегрузке (*overloading*), скрытии (*hiding*) и замещении (*overriding*) ? Обоснуйте ответ.

Примечание: замещение проявляется при работе механизма виртуальных функций.

6. Добавьте реализацию метода `sum` (из класса `C`), который вычисляет сумму всех числовых полей объекта класса `C` (каждое поле учитывается в сумме один раз). Операцию `::` можно использовать не более одного раза в каждом первичном выражении (т.е. `Y::x` возможно, `Z::Y::x` – нет).

Примечание: у операции `->` приоритет выше, чем у операции приведения типа `(mun) x`.

```
struct Base {  int a; };
```

```
struct A1 : Base {  int a;
};
```

```
struct A2 : Base { int a;};
```

```
struct B1 :  virtual
Base {  int a; };
```

```
struct B2 :  virtual Base
{  int a; };
```

```
struct C : A1,  A2, B1,  B2
{ int sum();  int a; };
```

7. Что будет напечатано в результате выполнения программы? Компилятор ***не*** оптимизирующий.

```
class A {
    static int c;
    int n;
public:
    A():n(++c){
       cout<<" A()="<< n;
    }

    A(const A&a):n(++c){
       cout<<" A(A&)="<< n;
    }
    ~A(){
       cout<<" ~A()="<< n;
    }
};
```

```
void f(){
    try {throw A(); throw 1;}
    catch (int) {cout << "int";}
    catch (A a) {throw;}
    cout<< " f() ";
}
int A::c=0;
int main() {
    try {
        try { f(); cout<< " after_f ";}
        catch(A & a) {throw a;}
       catch(...) { cout << "inner ..."; }
    }
    catch (...) {cout << " ... ";}
}
```

8. Опишите прототипы двух перегруженных функций `f` из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
f (-6);
f ('+', 6);
f ();
f ("abc");
```

9. Объясните, какой смысл имеет выражение `typeid(*p)`.

10. Перепишите данный фрагмент, не используя средства из C++11:

```
long  m;
decltype(m) n = 50000;
vector<long> v;
…
for (auto p=v.begin(); p!=v.end(); ++p) {n+=*p;}
```

Вариант 2_2015 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. Приведите пример ***ошибочного*** обращения к доступному, но не видимому (без операции `::`) методу класса.

2. Что будет напечатано данной программой? Можно ли удалить хотя бы одну операцию разрешения области видимости так, чтобы печать не изменилась?

```
namespace M {
    int f(){return 2;}
    int g=-2;
}
namespace N {
    int f(){return 3;}
    int g=-3;
    int h=0;
}
```

```
int f() {return -7;}
int main() { int g=-1;
    cout<<f()<<N::f()<<M::f()<<N::h<<endl;
  using namespace N;
    cout<<N::f()<<g<<h<<endl;
  using namespace M;
    cout<<M::f()<<M::g<<h<<endl;
    return 0;
}
```

3. Можно ли продемонстрировать параметрический полиморфизм с помощью оператора-выражения `(b+a)+b;` ? Обоснуйте ответ. Перечислите другие виды полиморфизма.

4. Данная программа выводит на печать несколько строк. Вычеркните ***фрагменты*** из функции ***main*** так, чтобы напечаталось верное утверждение об ООП и C++.

```
class A {
public:
    A() { cout<<"абстрактный "; }
    virtual ~A() {cout<<  endl;}
    void g(int n){cout<<"класс ";}
    virtual void g(){cout<<"метод ";}
};

class B : public A {
public: B(){}
    B(int x) {cout<<"деструктор ";}
    void g() { cout<<"конструктор ";}
    void g(int n) { cout<<"интерфейс
";}
    ~B() {cout<< "данных";}
};
```

```
int main() {
    cout<<" класс, содержащий чистую \
            виртуальную функцию, -- это ";
    A * qa= new B;
    qa->A::g();
    delete qa;
    qa= new B;
    { A & ra= *qa;
      ra.g();
    }
    delete qa;
    B * qb= new B;
    qb->g(7);
    delete qb;
    qa= new A;
    qa->g(7);
    delete qa;
    B b(7);
```

```
}
```

5. Может ли функция одновременно участвовать в скрытии (*hiding*) и замещении (*overriding*) и при этом не участвовать в перегрузке (*overloading*)? Обоснуйте ответ.

Примечание: замещение проявляется при работе механизма виртуальных функций.

6. Добавьте реализацию метода `prod` (из класса `C`), который вычисляет произведение всех числовых полей объекта класса `C` (каждое поле учитывается в произведении один раз). Операцию `::` можно использовать не более одного раза в каждом первичном выражении (т.е. `Y::x` возможно, `Z::Y::x` – нет).

Примечание: у операции `->` приоритет выше, чем у операции приведения типа `(mun) x`.

```
class Base
{ public: int a; };

class B1 :
  virtual public Base
{ public:  int a; };
```

```
class A1 : public Base
{ public: int a; };

class B2 :
   virtual public Base
{ public:  int a; };
```

```
class A2 : public Base
{ public: int a;};

class C : public A1, public A2,
          public B1, public B2
{ public: int prod();  int a;
};
```

7. Что будет напечатано в результате выполнения программы? Компилятор ***не*** оптимизирующий.

```
class A {
    static int c;
    int n;
public:
    A():n(++c){
       cout<<" A()="<< n;
    }

    A(const A&a):n(++c){
       cout<<" A(A&)="<< n;
    }
    ~A(){
       cout<<" ~A()="<< n;
    }
};
```

```
void f(){ try {throw A(); throw 1;}
          catch (int) {cout << "int";}
          catch (A &) {throw;}
          cout<< " f() ";
        }
int A::c=0;
int main(){
    try {
        try { f(); cout<< " after_f ";}
        catch(A  a) {throw a;}
        catch(...) { cout << "inner ..."; }
    }
    catch (...) {cout << " ... ";}
    return 0;
}
```

8. Опишите прототипы двух перегруженных функций `h` из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
h (-6,"bcd");
h('-', 6);
h ();
h (3.5);
```

9. Объясните, какой смысл имеет выражение **`typeid`**`(`*`имя_типа`*`)`.

10. Перепишите данный фрагмент, не используя средства из C++11:

```
double  t;
decltype(t) x = 50.000;
list<double> l;
…
auto p=l.rbegin();
while (p!=l.rend()) {
    x-= *p; p++;
}
```

Вариант 3_2015 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. Что будет напечатано данной программой? Можно ли удалить хотя бы одну операцию разрешения области видимости так, чтобы печать не изменилась?

```
namespace D { int f= 2;
              int g=-2;
}
namespace C { int f= 3;
              int g=-3;
              int h=0;
}
int f=-5;
```

```
int main() { int f=1;
    cout<<::f<<D::f<<C::f<<C::h<<endl;
  using namespace D;
    cout<<f<<g<<C::h<<endl;
  using namespace C;
    cout<<f<<C::g<<h<<endl;
    return 0;
}
```

2. Опишите прототипы двух перегруженных функций `g` из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
g (-6);
g ('+', 6);
g ();
g ("abc");
```

3. Перепишите данный фрагмент, не используя средства из C++11:

```
long  w;
decltype(w) n = 70000;
vector<long> v;
…
for (auto p=v.begin(); p!=v.end(); ++p) {n+=*p;}
```

4. Объясните, какой смысл имеет выражение `typeid(*p)`.

5. Что будет напечатано в результате выполнения программы? Компилятор ***не*** оптимизирующий.

```
class Y {
    static int c;
    int n;
public:
    Y():n(++c){
       cout<<" Y()="<< n;
    }

    Y(const Y&a):n(++c){
       cout<<" Y(Y&)="<< n;
    }
    ~Y(){
       cout<<" ~Y()="<< n;
    }
};
```

```
void h(){
    try {throw Y(); throw 1;}
    catch (int) {cout << "int";}
    catch (Y a) {throw;}
    cout<< " h() ";
}
int Y::c=0;
int main() {
    try {
        try { h(); cout<< " after_h ";}
        catch(Y & a) {throw a;}
       catch(...) { cout << "inner ..."; }
    }
    catch (...) {cout << " ... ";}
}
```

6. Можно ли продемонстрировать параметрический полиморфизм с помощью оператора-выражения `a*(b*a);` ? Обоснуйте ответ. Перечислите другие виды полиморфизма.

7. Данная программа выводит на печать несколько строк. Вычеркните ***фрагменты*** из функции ***main*** так, чтобы напечаталось верное утверждение об ООП и C++.

```
class X {
public:
   X() { cout<<"абстрактный "; }
   virtual ~X() {cout<<  endl;}
   void t(int n){cout<<"конструктор ";}
   virtual void t(){cout<<"метод ";}
};
class Y : public X {
public:
    Y(){}
    Y(int x) {cout<<"класс ";}
    void t() { cout<<"тип ";}
    void t(int n) { cout<<"интерфейс
";}
    ~Y() {cout<< "данных";}
};
```

```
int main() {
    cout<<"класс, не содержащий открытых \
          членов-данных, -- это " ;
    X * qx= new X;
    qx->t(5);
    delete qx;
    qx=new Y;
    qx->X::t();
    delete qx;
    qx= new Y;
    { X & rx= *qx;
      rx.t();
    }
    delete qx;
    Y * qy= new Y;
    qy->t(5);
    delete qy;
    Y y(5);
}
```

8. Может ли одна и та же функция одновременно участвовать в перегрузке (*overloading*), скрытии (*hiding*) и замещении (*overriding*) ? Обоснуйте ответ.

Примечание: замещение проявляется при работе механизма виртуальных функций.

9. Добавьте реализацию метода `sum` (из класса `C`), который вычисляет сумму всех числовых полей объекта класса `C` (каждое поле учитывается в сумме один раз). Операцию `::` можно использовать не более одного раза в каждом первичном выражении (т.е. `Y::x` возможно, `Z::Y::x` – нет).

Примечание: у операции `->` приоритет выше, чем у операции приведения типа `(mun) x`.

```
struct Base {  int b; };
```

```
struct A1 : Base {  int b;
};
```

```
struct A2 : Base { int b;};
```

```
struct B1 :  virtual
Base {  int b; };
```

```
struct B2 :  virtual Base
{  int b; };
```

```
struct C : A1,  A2, B1,  B2
{ int sum();  int b; };
```

10. Приведите фрагмент программы с примером ***ошибочного*** обращения к видимой, но недоступной функции.

Вариант 4_2015 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. Что будет напечатано данной программой? Можно ли удалить хотя бы одну операцию разрешения области видимости так, чтобы печать не изменилась?

```
namespace L {
    int f(){return 2;}
    int g=-2;
}
namespace K {
    int f(){return 3;}
    int g=-3;
    int h=0;
}
```

```
int f() {return -7;}
int main() { int g=-1;
    cout<<f()<<K::f()<<L::f()<<K::h<<endl;
  using namespace K;
    cout<<K::f()<<g<<h<<endl;
  using namespace L;
    cout<<L::f()<<L::g<<h<<endl;
    return 0;
}
```

2. Опишите прототипы двух перегруженных функций `p` из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
p();
p(-6,"qxq");
p(-0.5);
p('*', 9);
```

3. Перепишите данный фрагмент, не используя средства из C++11:

```
double  r;
decltype(r) x = 70.000;
list<double> l;
…
auto p=l.rbegin();
while (p!=l.rend()) {
    x-= *p; p++;
}
```

4. Объясните, какой смысл имеет выражение `typeid(имя_типа)`.

5. Что будет напечатано в результате выполнения программы? Компилятор ***не*** оптимизирующий.

```
class B {
    static int c;
    int n;
public:
    B():n(++c){
       cout<<" B()="<< n;
    }

    B(const B&a):n(++c){
       cout<<" B(B&)="<< n;
    }
    ~B(){
       cout<<" ~B()="<< n;
    }
};
```

```
void s(){ try {throw B(); throw 1;}
          catch (int) {cout << "int";}
          catch (B &) {throw;}
          cout<< " s() ";
        }
int B::c=0;
int main(){
    try {
        try { s(); cout<< " after_s ";}
        catch(B  a) {throw a;}
        catch(...) { cout << "inner ..."; }
    }
    catch (...) {cout << " ... ";}
    return 0;
}
```

6. Можно ли продемонстрировать параметрический полиморфизм с помощью оператора-выражения `(b*a)*b;` ? Обоснуйте ответ. Перечислите другие виды полиморфизма.

7. Данная программа выводит на печать несколько строк. Вычеркните ***фрагменты*** из функции ***main*** так, чтобы напечаталось верное утверждение об ООП и C++.

```
class Y {
public:
    Y() { cout<<"абстрактный "; }
    virtual ~Y() {cout<<  endl;}
    void g(int n){cout<<"класс ";}
    virtual void g(){cout<<"метод ";}
};

class Z : public Y {
public: Z(){}
    Z(int x) {cout<<"деструктор ";}
    void g() { cout<<"конструктор ";}
    void g(int n) { cout<<"интерфейс
";}
    ~Z() {cout<< "данных";}
};
```

```
int main() {
    cout<<" класс, содержащий чистую \
            виртуальную функцию, -- это ";
    Y * qy= new Z;
    qy->Y::g();
    delete qy;
    qy= new Z;
    { Y & ry= *qy;
      ry.g();
    }
    delete qy;
    Z * qz= new Z;
    qz->g(7);
    delete qz;
    qy= new Y;
    qy->g(7);
    delete qy;
    Z z(7);
}
```

8. Может ли функция одновременно участвовать в скрытии (*hiding*) и замещении (*overriding*) и при этом не участвовать в перегрузке (*overloading*)? Обоснуйте ответ.

Примечание: замещение проявляется при работе механизма виртуальных функций.

9. Добавьте реализацию метода `prod` (из класса `C`), который вычисляет произведение всех числовых полей объекта класса `C` (каждое поле учитывается в произведении один раз). Операцию `::` можно использовать не более одного раза в каждом первичном выражении (т.е. `Y::x` возможно, `Z::Y::x` – нет).

Примечание: у операции `->` приоритет выше, чем у операции приведения типа `(тип) x`.

```
class Base
{ public: int b; };

class B1 :
  virtual public Base
{ public:  int b; };
```

```
class A1 : public Base
{ public: int b; };

class B2 :
   virtual public Base
{ public:  int b; };
```

```
class A2 : public Base
{ public: int b;};

class C : public A1, public A2,
          public B1, public B2
{ public: int prod();  int b;
};
```

10. Приведите пример ***ошибочного*** обращения к доступному, но не видимому (без операции `::`) методу класса.

Коллоквиум 2015 Условия, ответы и решения, критерии

За каждую задачу максим 10 баллов.

Вариант 1_2015

1. Приведите фрагмент программы с примером ***ошибочного*** обращения к видимой, но недоступной функции.

Ответ: `class A {void f (){}; }; class B: public A { public: void g(){ f(); } };`

```
            // метод f () в приватной части A  недоступен для наследника
```

Критерий: пример не удовлетворяет условию или отсутствует : -10

2. Что будет напечатано данной программой? Можно ли удалить хотя бы одну операцию разрешения области видимости так, чтобы печать не изменилась?

```
namespace N { int f= 2;
              int g=-2;
}
namespace M { int f= 3;
              int g=-3;
              int h=0;
}
int f=-5;
```

```
int main() { int f=1;
    cout<<::f<<N::f<<M::f<<M::h<<endl;
  using namespace N;
    cout<<f<<g<<M::h<<endl;
  using namespace M;
    cout<<f<<M::g<<h<<endl;
    return 0;
}
```

Ответ :
```
    -5230
    1-20
    1-30
Удалить :: нельзя.
```

Критерий: за каждую неверно напечатанную строчку: -3
за неправильный ответ на второй вопрос: -4

3. Можно ли продемонстрировать параметрический полиморфизм с помощью оператора-выражения `a+(b+a);` ? Обоснуйте ответ. Перечислите другие виды полиморфизма.

Ответ: другие виды полиморфизма – статический, динамический.

Параметрический продемонстрировать можно: нужно привести такое описание шаблонной операции, чтобы разные вхождения + в выражение `a+(b+a);` настраивались на разные типы операндов (получаем, что одна и та же операция означает разное в зависимости от контекста – это и есть проявление полиморфизма). Однако, если приведен пример с настройкой на один и тот же тип, то такой ответ тоже засчитываем без снятия баллов. Обоснование может быть схематичным, например, без реализации тел функций.

```
#include <iostream>
using namespace std;
template <typename T1, typename T2>
T1 operator+ (T1 a, T2 b) {cout<<a<<"+"<<b<<" | "; return a;
}
class A{
        char c;
        public:
```

```
        A():c('A'){}
        friend ostream & operator<<(ostream & out, const A & a);
};
ostream & operator<<(ostream & out, const A & a ){
        out<<a.c;
        return out;
}

class B{
        char c;
        public:
        B():c('B'){}
        friend ostream & operator<<(ostream & out, const B & b);
};
ostream & operator<<(ostream & out, const B & b ){
        out<<b.c;
        return out;
}

int main() {
A a; B b;
a+(b+a); // operator+ (a, operator+(b,a));
         // Напечатается : B+A | A+B |
        return 0;
}
```

Критерий:
за каждый не названный (или не существующий) вид полиморфизма: -2
за неверный ответ или отсутствующее обоснование про параметрический полиморфизм: -6

4. Данная программа выводит на печать несколько строк. Вычеркните ***фрагменты*** из функции ***main*** так, чтобы напечаталось верное утверждение об ООП и C++.

```
class A {
public:
   A() { cout<<"абстрактный "; }
   virtual ~A() {cout<<  endl;}
   void t(int n){cout<<"конструктор ";}
   virtual void t(){cout<<"метод ";}
};
class B : public A {
public:
    B(){}
    B(int x) {cout<<"класс ";}
    void t() { cout<<"тип ";}
    void t(int n) { cout<<"интерфейс
";}
    ~B() {cout<< "данных";}
};
```

```
int main() {
    cout<<"класс, не содержащий открытых \
           членов-данных, -- это " ;
    A * qa= new A;
    qa->t(5);
    delete qa;
    qa=new B;
    qa->A::t();
    delete qa;
    qa= new B;
    { A & ra= *qa;
      ra.t();
    }
    delete qa;
    B * qb= new B;
    qb->t(5);
    delete qb;
    B b(5);
}
```

Ответ: напечатаются строки

```
Класс, не содержащий открытых членов-данных, -- это абстрактный конструктор
абстрактный метод данных
абстрактный тип данных
абстрактный интерфейс данных
абстрактный класс данных
```

Единственный правильный вариант: *класс, в котором нет открытых членов, -- это абстрактный тип данных.*
Поэтому в теле main нужно оставить только:

```
cout<<  "класс, не содержащий открытых членов-данных, -- это " ;
A * qa;
qa= new B;
{ A & ra= *qa;
ra.t();
}
delete qa;
```

Критерий: за неверный ответ или вычеркивание не в main: -10

5. Может ли одна и та же функция одновременно участвовать в перегрузке (*overloading*), скрытии (*hiding*) и замещении (*overriding*) ? Обоснуйте ответ.

Примечание: замещение проявляется при работе механизма виртуальных функций.
Ответ : да, в задаче 4 таковой является функция void t(); из класса B.
Критерий: -10 за неправильный или не обоснованный ответ.

6. Добавьте реализацию метода `sum` (из класса `C`), который вычисляет сумму всех числовых полей объекта класса `C` (каждое поле учитывается в сумме один раз). Операцию `::` можно использовать не более одного раза в каждом первичном выражении (т.е. `Y::x` возможно, `Z::Y::x` – нет).

Примечание: у операции `->` приоритет выше, чем у операции приведения типа `(тип) x`.

```
struct Base {  int a; };
```

```
struct A1 : Base {  int a;
};
```

```
struct A2 : Base { int a;};
```

```
struct B1 :  virtual
Base {  int a; };
```

```
struct B2 :  virtual Base
{  int a; };
```

```
struct C : A1,  A2, B1,  B2
{ int sum();  int a; };
```

Решение

Для начала выясним, сколько всего полей в объекте класса *C* ?
1) *a* из *C* (объявлено непосредственно в *C*)
2) *a* из *A1* (объявлено в *A1*)
3) *a* из *A2*
4) *a* из *B1*
5) *a* из *B2*
6) *a* из *Base*, который является частью *A1*
7) *a* из *Base*, который является частью *A2*
8) *a* из *Base*, который является общей частью *B1* и *B2* («виртуальное» наследование)

Как добраться в методе *sum*( ) до каждого поля "*a*" из каждого класса? С первыми пятью из перечисленных в списке проблем нет. Например: *A1::a* означает поле "*a*", объявленное в классе *A1*.
Выражение *Base::a* в классе *C* неоднозначно, т.к. *Base* присутствует в классе *C* в трех экземплярах — непонятно, какое из трех "*a*" имеется ввиду.
[Компилятор Visual C++ умеет различать их таким способом:
A1::Base::a, *A2::Base::a*, *B1::Base::a* . Вместо последнего можно было также написать *B2::Base::a* (оба способа именуют одно и то же поле).
Компилятор g++ считает имена *A1::Base::a*, *A2::Base::a*, *B1::Base::a* неоднозначными. Он, скорее всего, прав, поскольку грамматика C++ так устроена, что :: получается правоассоциативной операцией. Поэтому, чтобы не сталкиваться с разным поведением компиляторов, просто запрещаем в условии задачи дважды использовать :: в одном первичном выражении]

Как быть?
Вспомним, что у нас есть указатель на объект *this*. Если привести его явно к указателю на *A2*, получим указатель на «подобъект» типа *A2* из нашего объекта, далее приводим полученный указатель к указателю на *Base* и получаем указатель на «подобъект» типа *Base*, причем именно тот *Base*, который «находится» в *A2*. Также и до остальных двух *Base*'овых полей можно «добраться». Другой допустимый вариант: привести *this* сначала к указателю на *A1* (соответственно *A2*, *B1*) и использовать доступ к полю *Base:: a* через указатель, например:
((*A1**) *this*)->*Base:: a*
Итого:

```
int C::sum( ) { return a + A1::a+ A2::a+ B1::a + B2::a +
                        ( (Base*) (A1*) this)->a  +      // другой вариант ( (A1*) this)->Base:: a
                        ( (Base*) (A2*) this)->a  +
                        ( (Base*) (B1*) this)->a  ; }
```

Чтобы проверить, можно воспользоваться следующей функцией:

```
int main() {
```

```
        C c, *p=&c; c.A1::a=1; c.A2::a=1; c.a=1;c.B1::a=1; c.B2::a=1; ((A1*)p)->Base::a=1;
        ((B1*)p)->Base::a=1;  ((A2*)p)->Base::a=1;   // ((B2*)p)->Base::a=1;
        cout<<c.sum();
        return 0;
}
```

Ответ:

```
int C::sum( ) { return a + A1::a+ A2::a+ B1::a + B2::a +
                        ( (Base*) (A1*) this)->a  +     // другой  вариант: ( (A1*) this)->Base:: a
                        ( (Base*) (A2*) this)->a  +     // другой  вариант: ( (A2*) this)->Base:: a
                                                        // или   ( (B2*) this)->Base:: a
                        ( (Base*) (B1*) this)->a  ; }   // другой  вариант: ( (B1*) this)->Base:: a
```

Критерий: за каждое неверное слагаемое : -3

7. Что будет напечатано в результате выполнения программы? Компилятор ***не*** оптимизирующий.

```
class A {
    static int c;
    int n;
public:
    A():n(++c){
       cout<<" A()="<< n;
    }

    A(const A&a):n(++c){
       cout<<" A(A&)="<< n;
    }
    ~A(){
       cout<<" ~A()="<< n;
    }
};
```

```
void f(){
    try {throw A(); throw 1;}
    catch (int) {cout << "int";}
    catch (A a) {throw;}
    cout<< " f() ";
}
int A::c=0;
int main() {
    try {
        try { f(); cout<< " after_f ";}
        catch(A & a) {throw a;}
       catch(...) { cout << "inner ..."; }
    }
    catch (...) {cout << " ... ";}
}
```

Конструктор A() в операторе throw A(); будет вызываться для создания временного объекта, с которого будет «сниматься копия» конструктором копирования для создания объекта-исключения, после чего исключение «полетит», а временный объект, созданный A(), исчезнет. Компиляторы обычно оптимизируют эту ситуацию, не делая лишнего копирования. Однако в условии явно сказано, что оптимизации нет. Поэтому ответ не совсем похож на то, что выдаст скомпилированная программа. В случае throw a; временный объект не создается, копия при создании объекта-исключения «снимается» с уже существующего объекта a.

Ответ: `A()=1 A(A&)=2 ~A()=1  A(A&)=3  ~A()=3 A(A&)=4 ~A()=2 ...  ~A()=4`

Критерий: за каждую ошибку -3

8. Опишите прототипы двух перегруженных функций `f` из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
f (-6);
f ('+', 6);
f ();
f ("abc");
```

Ответ: `f(int i=5, int j=9); f(const char * s);` //Возможны другие решения

Критерий: -3 за каждый неподходящий вызов из четырех, но не меньше 0.
Если больше двух функций описано: 0

9. Объясните, какой смысл имеет выражение `typeid(*p)`.

Ответ : динамическое определение типа объекта, на который указывает указатель p.

Критерий: - 10 за отсутствие или неправильный ответ.

10. Перепишите данный фрагмент, не используя средства из C++11:

```
long  m;
decltype(m) n = 50000;
vector<long> v;
…
for (auto p=v.begin(); p!=v.end(); ++p) {n+=*p;}
```

Ответ:
long m, n = 50000;
vector <long> v;
…
for (vector <long>::iterator p = v.begin(); p != v.end(); ++p) {n+= *p;}
Критерий: неправильно заменен decltype: -5
неправильно заменен auto: -5

Вариант 2_2015

1. Приведите пример ***ошибочного*** обращения к доступному, но не видимому (без операции ::) методу класса.
Ответ: `class A {public: void f (){}; }; class B: public A { public: void g(int f) {f();};`
// параметр f скрыл метод f() из A
Или `class A {public: void f (){}; }; class B: public A { int f; public: void g(){f();}`
// поле f скрыло метод f() из A
Критерий: пример не удовлетворяет условию или отсутствует : -10

2. Что будет напечатано данной программой? Можно ли удалить хотя бы одну операцию разрешения области видимости так, чтобы печать не изменилась?

```
namespace M {
    int f(){return 2;}
    int g=-2;
}
namespace N {
    int f(){return 3;}
    int g=-3;
    int h=0;
}
```

```
int f() {return -7;}
int main() { int g=-1;
    cout<<f()<<N::f()<<M::f()<<N::h<<endl;
  using namespace N;
    cout<<N::f()<<g<<h<<endl;
  using namespace M;
    cout<<M::f()<<M::g<<h<<endl;
    return 0;
}
```

Ответ : -7320
3-10
2-20

Удалить :: нельзя.

Критерий: за каждую неверно напечатанную строчку: -3
за неправильный ответ на второй вопрос: -4

3. Можно ли продемонстрировать параметрический полиморфизм с помощью оператора-выражения (b+a)+b; ? Обоснуйте ответ. Перечислите другие виды полиморфизма.
Ответ: другие виды полиморфизма – статический, динамический.
Параметрический продемонстрировать можно: нужно привести такое описание шаблонной операции, чтобы разные вхождения + в выражение (b+a)+b; настраивались на разные типы операндов (получаем, что одна и та же операция означает разное в зависимости от контекста – это и есть проявление полиморфизма). Однако, если приведен пример с настройкой на один и тот же тип, то такой ответ тоже засчитываем без снятия баллов. Обоснование может быть схематичным, например, без реализации тел функций.

```
#include <iostream>
using namespace std;
template <typename T1, typename T2>
T2 operator+ (T1 a, T2 b) {cout<<a<<"+"<<b<<" | "; return b;
}
class A{
        char c;
        public:
        A():c('A'){}
        friend ostream & operator<<(ostream & out, const A & a);
};
ostream & operator<<(ostream & out, const A & a ){
        out<<a.c;
```

```
        return out;
}

class B{
        char c;
        public:
        B():c('B'){}
        friend ostream & operator<<(ostream & out, const B & b);
};
ostream & operator<<(ostream & out, const B & b ){
        out<<b.c;
        return out;
}

int main() {
A a; B b;
(b+a)+b; // operator+ ( operator+(b,a),b);
         // Напечатается : B+A | A+B |
        return 0;
}
```

Критерий:
за каждый не названный (или не существующий) вид полиморфизма: -2
за неверный ответ или отсутствующее обоснование про параметрический полиморфизм: -6

4. Данная программа выводит на печать несколько строк. Вычеркните ***фрагменты*** из функции ***main*** так, чтобы напечаталось верное утверждение об ООП и C++.

```
class A {
public:
    A() { cout<<"абстрактный "; }
    virtual ~A() {cout<<  endl;}
    void g(int n){cout<<"класс ";}
    virtual void g(){cout<<"метод ";}
};

class B : public A {
public: B(){}
    B(int x) {cout<<"деструктор ";}
    void g() { cout<<"конструктор ";}
    void g(int n) { cout<<"интерфейс
";}
    ~B() {cout<< "данных";}
};
```

```
int main() {
    cout<<" класс, содержащий чистую \
            виртуальную функцию, -- это ";
    A * qa= new B;
    qa->A::g();
    delete qa;
    qa= new B;
    { A & ra= *qa;
      ra.g();
    }
    delete qa;
    B * qb= new B;
    qb->g(7);
    delete qb;
    qa= new A;
    qa->g(7);
    delete qa;
    B b(7);
}
```

Ответ: напечатаются строки

```
класс, содержащий чистую     виртуальную функцию, -- это абстрактный метод данных
абстрактный конструктор данных
абстрактный интерфейс данных
абстрактный класс
абстрактный деструктор данных
```

Единственный правильный вариант: *класс, содержащий чистую виртуальную функцию, -- это абстрактный класс*. Поэтому в теле main нужно оставить только:

```
cout<<  "класс, содержащий чистую виртуальную функцию, -- это " ;
A * qa;
qa=new A;
qa-> g(7);
delete qa;
```

Критерий: за неверный ответ или вычеркивание не в main: -10

5. Может ли функция одновременно участвовать в скрытии (*hiding*) и замещении (*overriding*) и при этом не участвовать в перегрузке (*overloading*)? Обоснуйте ответ.

Примечание: замещение проявляется при работе механизма виртуальных функций.

Ответ : да, если в задаче 4 в классе B вычеркнуть g(int); то оставшаяся функция void g(); из класса B будет удовлетворять условию задачи.
Критерий: -10 за неправильный или не обоснованный ответ.

6. Добавьте реализацию метода `prod` (из класса `C`), который вычисляет произведение всех числовых полей объекта класса `C` (каждое поле учитывается в произведении один раз). Операцию `::` можно использовать не более одного раза в каждом первичном выражении (т.е. `Y::x` возможно, `Z::Y::x` – нет).

Примечание: у операции `->` приоритет выше, чем у операции приведения типа `(тип) x`.

```
class Base
{ public: int a; };

class B1 :
  virtual public Base
{ public:  int a; };
```

```
class A1 : public Base
{ public: int a; };

class B2 :
   virtual public Base
{ public:  int a; };
```

```
class A2 : public Base
{ public: int a;};

class C : public A1, public A2,
          public B1, public B2
{ public: int prod();  int a;
};
```

Ответ:
*int C::prod( ) { return a * A1::a* A2::a* B1::a * B2::a *
((Base*) (A1*) this)->a * // другой вариант: ((A1*) this)->Base:: a
((Base*) (A2*) this)->a *
((Base*) (B1*) this)->a ; }*
Критерий: за каждый неверный сомножитель : -3

7. Что будет напечатано в результате выполнения программы? Компилятор ***не*** оптимизирующий.

```
class A {
    static int c;
    int n;
public:
    A():n(++c){
       cout<<" A()="<< n;
    }

    A(const A&a):n(++c){
       cout<<" A(A&)="<< n;
    }
    ~A(){
       cout<<" ~A()="<< n;
    }
};
```

```
void f(){ try {throw A(); throw 1;}
          catch (int) {cout << "int";}
          catch (A &) {throw;}
          cout<< " f() ";
        }
int A::c=0;
int main(){
    try {
        try { f(); cout<< " after_f ";}
        catch(A  a) {throw a;}
        catch(...) { cout << "inner ..."; }
    }
    catch (...) {cout << " ... ";}
    return 0;
}
```

Ответ: `A()=1 A(A&)=2 ~A()=1 A(A&)=3 A(A&)=4 ~A()=3 ~A()=2 ... ~A()=4`
Критерий: за каждую ошибку -3

8. Опишите прототипы двух перегруженных функций `h` из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
h (-6,"bcd");
h('-', 6);
h ();
h (3.5);
```

Ответ: `h(double d=5, int j=9); h(int i, const char * s);` //Возможны другие решения
Критерий: -3 за каждый неподходящий вызов из четырех, но не меньше 0.
Если больше двух функций описано: 0

9. Объясните, какой смысл имеет выражение **`typeid`**`(`*`имя_типа`*`)`.

Ответ : в этом случае оператор typeid возвращает ссылку на объект типа **type_info**, представляющий тип именно с этим именем.

Критерий: - 10 за отсутствие или неправильный ответ.

10. Перепишите данный фрагмент, не используя средства из C++11:

```
double  t;
decltype(t) x = 50.000;
list<double> l;
…
auto p=l.rbegin();
while (p!=l.rend()) {
    x-= *p; p++;
}
```

Ответ:

```
double  t,   x = 50.000;
list  <double> l;
…
list <double>::reverse_iterator p = l.rbegin();
while  (p != l.rend())   {x-= *p;  p++;}
```

Критерий: неправильно заменен decltype: -5
неправильно заменен auto: -5

Вариант 3_2015

1. Что будет напечатано данной программой? Можно ли удалить хотя бы одну операцию разрешения области видимости так, чтобы печать не изменилась?

```
namespace D { int f= 2;
              int g=-2;
}
namespace C { int f= 3;
              int g=-3;
              int h=0;
}
int f=-5;
```

```
int main() { int f=1;
    cout<<::f<<D::f<<C::f<<C::h<<endl;
  using namespace D;
    cout<<f<<g<<C::h<<endl;
  using namespace C;
    cout<<f<<C::g<<h<<endl;
    return 0;
}
```

Ответ :
```
    -5230
   1-20
   1-30
Удалить :: нельзя.
```

Критерий: за каждую неверно напечатанную строчку:-3
за неправильный ответ на второй вопрос: -4

2. Опишите прототипы двух перегруженных функций g из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
g (-6);
g ('+', 6);
g ();
g ("abc");
```

Ответ: `g(int i=5, int j=9);  g(const char * s);` //Возможны другие решения
Критерий: -3 за каждый неподходящий вызов из четырех, но не меньше 0.
Если больше двух функций описано: 0

3. Перепишите данный фрагмент, не используя средства из C++11:

```
long  w;
decltype(w) n = 70000;
vector<long> v;
…
for (auto p=v.begin(); p!=v.end(); ++p) {n+=*p;}
```
Ответ:
long w, n = 70000;
vector <long> v;
…
for (vector <long>::iterator p = v.begin(); p != v.end(); ++p) {n+= *p;}
Критерий: неправильно заменен decltype: -5
неправильно заменен auto: -5

4. Объясните, какой смысл имеет выражение `typeid(*p)`.
Ответ : динамическое определение типа объекта, на который указывает указатель p.
Критерий: - 10 за отсутствие или неправильный ответ.

5. Что будет напечатано в результате выполнения программы? Компилятор ***не*** оптимизирующий.

```
class Y {
    static int c;
    int n;
public:
    Y():n(++c){
       cout<<" Y()="<< n;
    }

    Y(const Y&a):n(++c){
       cout<<" Y(Y&)="<< n;
    }
    ~Y(){
       cout<<" ~Y()="<< n;
    }
};
```

```
void h(){
    try {throw Y(); throw 1;}
    catch (int) {cout << "int";}
    catch (Y a) {throw;}
    cout<< " h() ";
}
int Y::c=0;
int main() {
    try {
        try { h(); cout<< " after_h ";}
        catch(Y & a) {throw a;}
       catch(...) { cout << "inner ..."; }
    }
    catch (...) {cout << " ... ";}
}
```

Ответ: `Y()=1 Y(Y&)=2 ~Y()=1 Y(Y&)=3 ~Y()=3 Y(Y&)=4 ~Y()=2 ... ~Y()=4`
Критерий: за каждую ошибку -3

6. Можно ли продемонстрировать параметрический полиморфизм с помощью оператора-выражения `a*(b*a);` ? Обоснуйте ответ. Перечислите другие виды полиморфизма.
Ответ: другие виды полиморфизма – статический, динамический.
Параметрический продемонстрировать можно: нужно привести такое описание шаблонной операции, чтобы разные вхождения * в выражение `a*(b*a);` настраивались на разные типы операндов (получаем, что одна и та же операция означает разное в зависимости от контекста – это и есть проявление полиморфизма). Однако, если приведен пример с настройкой на один и тот же тип, то такой ответ тоже засчитываем без снятия баллов. Обоснование может быть схематичным, например, без реализации тел функций.

```
#include <iostream>
using namespace std;
template <typename T1, typename T2>
T1 operator* (T1 a, T2 b) {cout<<a<<"*"<<b<<" | "; return a;
```

```
}
class A{
        char c;
        public:
        A():c('A'){}
        friend ostream & operator<<(ostream & out, const A & a);
};
ostream & operator<<(ostream & out, const A & a ){
        out<<a.c;
        return out;
}

class B{
        char c;
        public:
        B():c('B'){}
        friend ostream & operator<<(ostream & out, const B & b);
};
ostream & operator<<(ostream & out, const B & b ){
        out<<b.c;
        return out;
}

int main() {
A a; B b;
a*(b*a); // operator* (a, operator*(b,a));
         // Напечатается : B*A | A*B |
        return 0;
}
```

Критерий:
за каждый не названный (или не существующий) вид полиморфизма: -2
за неверный ответ или отсутствующее обоснование про параметрический полиморфизм: -6

7. Данная программа выводит на печать несколько строк. Вычеркните ***фрагменты*** из функции ***main*** так, чтобы напечаталось верное утверждение об ООП и C++.

```
class X {
public:
   X() { cout<<"абстрактный "; }
   virtual ~X() {cout<<  endl;}
   void t(int n){cout<<"конструктор ";}
   virtual void t(){cout<<"метод ";}
};
class Y : public X {
public:
    Y(){}
    Y(int x) {cout<<"класс ";}
    void t() { cout<<"тип ";}
    void t(int n) { cout<<"интерфейс
";}
    ~Y() {cout<< "данных";}
};
```

```
int main() {
    cout<<"класс, не содержащий открытых \
          членов-данных, -- это " ;
    X * qx= new X;
    qx->t(5);
    delete qx;
    qx=new Y;
    qx->X::t();
    delete qx;
    qx= new Y;
    { X & rx= *qx;
      rx.t();
    }
    delete qx;
    Y * qy= new Y;
    qy->t(5);
    delete qy;
    Y y(5);
}
```

Ответ: напечатаются строки

```
Класс, не содержащий открытых членов-данных, -- это абстрактный конструктор
абстрактный метод данных
абстрактный тип данных
абстрактный интерфейс данных
абстрактный класс данных
```

Единственный правильный вариант: *класс, в котором нет открытых членов, -- это абстрактный тип данных*.
Поэтому в теле main нужно оставить только:

```
cout<<  "класс, не содержащий открытых членов-данных, -- это " ;
X * qx;
qx= new Y;
```

```
{ X & rx= *qx;
rx.t();
}
delete qx;
```

Критерий: за неверный ответ или вычеркивание не в main: -10

8. Может ли одна и та же функция одновременно участвовать в перегрузке (*overloading*), скрытии (*hiding*) и замещении (*overriding*) ? Обоснуйте ответ.

Примечание: замещение проявляется при работе механизма виртуальных функций.
Ответ : да, в задаче 7 таковой является функция void t(); из класса Y.
Критерий: -10 за неправильный или не обоснованный ответ.

9. Добавьте реализацию метода `sum` (из класса `C`), который вычисляет сумму всех числовых полей объекта класса `C` (каждое поле учитывается в сумме один раз). Операцию `::` можно использовать не более одного раза в каждом первичном выражении (т.е. `Y::x` возможно, `Z::Y::x` – нет).

Примечание: у операции `->` приоритет выше, чем у операции приведения типа `(тип) x`.

```
struct Base {  int b; };
struct B1 :  virtual
Base {  int b; };
```

```
struct A1 : Base {  int b;
};
struct B2 :  virtual Base
{  int b; };
```

```
struct A2 : Base { int b;};
struct C : A1,  A2, B1,  B2
{ int sum();  int b; };
```

Ответ:

```
int C::sum( ) { return b + A1::b+ A2::b+ B1::b + B2::b +
                        ( (Base*) (A1*) this)->b  +   // другой  вариант: ( (A1*) this)->Base:: b
                        ( (Base*) (A2*) this)->b  +
                        ( (Base*) (B1*) this)->b  ; }
```

Критерий: за каждое неверное слагаемое : -3

10. Приведите фрагмент программы с примером ***ошибочного*** обращения к видимой, но недоступной функции.

Ответ:
```
class A {void f (){}; }; class B: public A { public: void g(){ f(); } };
        // метод f () в приватной части A  недоступен для наследника
```

Критерий: пример не удовлетворяет условию или отсутствует : -10

Вариант 4_2015

1. Что будет напечатано данной программой? Можно ли удалить хотя бы одну операцию разрешения области видимости так, чтобы печать не изменилась?

```
namespace L {
    int f(){return 2;}
    int g=-2;
}
namespace K {
    int f(){return 3;}
    int g=-3;
    int h=0;
}
```

```
int f() {return -7;}
int main() { int g=-1;
    cout<<f()<<K::f()<<L::f()<<K::h<<endl;
  using namespace K;
    cout<<K::f()<<g<<h<<endl;
  using namespace L;
    cout<<L::f()<<L::g<<h<<endl;
    return 0;
}
```

Ответ :
```
-7320
 3-10
 2-20
```

Удалить :: нельзя.

Критерий: за каждую неверно напечатанную строчку: -3
за неправильный ответ на второй вопрос: -4

2. Опишите прототипы двух перегруженных функций `p` из некоторой области видимости, для которых будут верны следующие обращения к ним:

```
p();
p(-6,"qxq");
p(-0.5);
p('*', 9);
```

Ответ: `p(double d=5, int j=9); p(int i, const char * s);` //Возможны другие решения

Критерий: -3 за каждый неподходящий вызов из четырех, но не меньше 0.
Если больше двух функций описано: 0

3. Перепишите данный фрагмент, не используя средства из C++11:

```
double  r;
decltype(r) x = 70.000;
list<double> l;
…
auto p=l.rbegin();
while (p!=l.rend()) {
    x-= *p; p++;
}
```

Ответ:

```
double  r,   x = 70.000;
list  <double> l;
…
list <double>::reverse_iterator p = l.rbegin();
while  (p != l.rend())   {x-= *p;  p++;}
```

Критерий: неправильно заменен decltype: -5
неправильно заменен auto: -5

4. Объясните, какой смысл имеет выражение `typeid`(*имя_типа*).

Ответ : в этом случае оператор typeid возвращает ссылку на объект типа **type_info**, представляющий тип именно с этим именем.

Критерий: - 10 за отсутствие или неправильный ответ.

5. Что будет напечатано в результате выполнения программы? Компилятор ***не*** оптимизирующий.

```
class B {
    static int c;
    int n;
public:
    B():n(++c){
       cout<<" B()="<< n;
    }

    B(const B&a):n(++c){
       cout<<" B(B&)="<< n;
    }
    ~B(){
       cout<<" ~B()="<< n;
    }
};
```

```
void s(){ try {throw B(); throw 1;}
          catch (int) {cout << "int";}
          catch (B &) {throw;}
          cout<< " s() ";
        }
int B::c=0;
int main(){
    try {
        try { s(); cout<< " after_s ";}
        catch(B  a) {throw a;}
        catch(...) { cout << "inner ..."; }
    }
    catch (...) {cout << " ... ";}
    return 0;
}
```

Ответ: `B()=1  B(B&)=2  ~B()=1  B(B&)=3 B(B&)=4  ~B()=3  ~B()=2 ...  ~B()=4`

Критерий: за каждую ошибку -3

6. Можно ли продемонстрировать параметрический полиморфизм с помощью оператора-выражения `(b*a)*b;` ? Обоснуйте ответ. Перечислите другие виды полиморфизма.
Ответ: другие виды полиморфизма – статический, динамический.
Параметрический продемонстрировать можно: нужно привести такое описание шаблонной операции, чтобы разные вхождения * в выражение `(b*a)*b;` настраивались на разные типы операндов (получаем, что одна и та же операция означает разное в зависимости от контекста – это и есть проявление полиморфизма). Однако, если приведен пример с настройкой на один и тот же тип, то такой ответ тоже засчитываем без снятия баллов. Обоснование может быть схематичным, например, без реализации тел функций.

```
#include <iostream>
using namespace std;
template <typename T1, typename T2>
T2 operator* (T1 a, T2 b) {cout<<a<<"*"<<b<<" | "; return b;
}
class A{
        char c;
        public:
        A():c('A'){}
        friend ostream & operator<<(ostream & out, const A & a);
};
ostream & operator<<(ostream & out, const A & a ){
        out<<a.c;
        return out;
}

class B{
        char c;
        public:
        B():c('B'){}
        friend ostream & operator<<(ostream & out, const B & b);
};
ostream & operator<<(ostream & out, const B & b ){
        out<<b.c;
        return out;
}

int main() {
A a; B b;
(b*a)*b; // operator*( operator*(b,a),b);
         // Напечатается : B*A | A*B |
        return 0;
}
```

Критерий:
за каждый не названный (или не существующий) вид полиморфизма: -2
за неверный ответ или отсутствующее обоснование про параметрический полиморфизм: -6

7. Данная программа выводит на печать несколько строк. Вычеркните ***фрагменты*** из функции ***main*** так, чтобы напечаталось верное утверждение об ООП и С++.

```
class Y {
public:
    Y() { cout<<"абстрактный "; }
    virtual ~Y() {cout<<  endl;}
    void g(int n){cout<<"класс ";}
    virtual void g(){cout<<"метод ";}
};

class Z : public Y {
public: Z(){}
    Z(int x) {cout<<"деструктор ";}
    void g() { cout<<"конструктор ";}
    void g(int n) { cout<<"интерфейс
";}
    ~Z() {cout<< "данных";}
};
```

```
int main() {
    cout<<" класс, содержащий чистую \
            виртуальную функцию, -- это ";
    Y * qy= new Z;
    qy->Y::g();
    delete qy;
    qy= new Z;
    { Y & ry= *qy;
      ry.g();
    }
    delete qy;
    Z * qz= new Z;
    qz->g(7);
    delete qz;
    qy= new Y;
    qy->g(7);
    delete qy;
    Z z(7);
}
```

Ответ: напечатаются строки

```
класс, содержащий чистую     виртуальную функцию, -- это абстрактный метод данных
абстрактный конструктор данных
абстрактный интерфейс данных
абстрактный класс
абстрактный деструктор данных
```

Единственный правильный вариант: *класс, содержащий чистую виртуальную функцию, -- это абстрактный класс*. Поэтому в теле main нужно оставить только:

```
cout<<  "класс, содержащий чистую виртуальную функцию, - это " ;
Y * qy;
qy=new Y;
qy-> g(7);
delete qy;
```

Критерий: за неверный ответ или вычеркивание не в main: -10

8. Может ли функция одновременно участвовать в скрытии (*hiding*) и замещении (*overriding*) и при этом не участвовать в перегрузке (*overloading*)? Обоснуйте ответ.

Примечание: замещение проявляется при работе механизма виртуальных функций.

Ответ : да, если в задаче 7 в классе Z вычеркнуть g(int); то оставшаяся функция void g(); из класса Z будет удовлетворять условию задачи.

Критерий: -10 за неправильный или не обоснованный ответ.

9. Добавьте реализацию метода `prod` (из класса `C`), который вычисляет произведение всех числовых полей объекта класса `C` (каждое поле учитывается в произведении один раз). Операцию `::` можно использовать не более одного раза в каждом первичном выражении (т.е. `Y::x` возможно, `Z::Y::x` – нет).

Примечание: у операции `->` приоритет выше, чем у операции приведения типа `(тип) x`.

```
class Base
{ public: int b; };

class B1 :
  virtual public Base
{ public:  int b; };
```

```
class A1 : public Base
{ public: int b; };

class B2 :
   virtual public Base
{ public:  int b; };
```

```
class A2 : public Base
{ public: int b;};

class C : public A1, public A2,
          public B1, public B2
{ public: int prod();  int b;
};
```

Ответ:

*int C::prod( ) { return b * A1::b* A2::b* B1::b * B2::b **
((Base) (A1*) this)->b * // другой вариант: ((A1*) this)->Base:: b*
((Base) (A2*) this)->b **
((Base) (B1*) this)->b ; }*

Критерий: за каждый неверный сомножитель : -3

10. Приведите пример ***ошибочного*** обращения к доступному, но не видимому (без операции `::`) методу класса.

Ответ: `class A {public: void f (){}; }; class B: public A { public: void g(int f) {f();};`
// параметр f скрыл метод f() из A

Или `class A {public: void f (){}; } ; class B: public A { int f; public: void g(){f();}`
// поле f скрыло метод f() из A

Критерий: пример не удовлетворяет условию или отсутствует: -10

2015 год
Экзамен

Вариант Э1_2015 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. При компиляции функции main() следующей программы будет зафиксирована ошибка: "неразрешимый вызов функции f (a, 0)". Объяснить причину такой ошибки, пользуясь алгоритмом выбора best-matching функции.

```
struct A {
    operator int () { return 1; }
};
void f (double d, char c)  { cout << "f (double, char)\n";}
void f (double d, int j)   { cout << "f (double, int)\n";}
void f (A a, const char * p)  { cout << "f ( A, const char *)\n";}
void f (int i, const char * p)  { cout << "f (int, const char *)\n";}
```

```
int main () {
        A a ;
        f (a, 0);
return 0;
```

2. Привести два примера классов, для которых нельзя создать объект. Причины, по которым нельзя создать объект, должны быть для этих классов **разными.**

3. При компиляции приведенной программы на C++ фиксируются ошибки. Объяснить, в чем они заключаются. Исправить все ошибочные конструкции **только** путем **вставки** или **замены** служебных слов C++. Что будет напечатано в результате работы получившейся правильной программы?

```
struct B {
    virtual char f () {
        cout << "f_from_B\n";  return 0; }
    virtual void g () {
        cout << "g_from_B\n "; }
};
struct T : B {
    int f () {
        cout << "f_from_T\n";  return 0; }
    void g () {
        cout << "g_from_T\n "; }
};
```

```
int main () {
    int n;
    const T ct;
    T t;
    B * pb = &t;
    n = pb -> f ();
    pb -> g ();
    ct.g();
    return 0;
}
```

4. Что будет напечатано в результате выполнения программы? Компилятор оптимизирующий.

```
class A {
public:
     static A sa;
    A(){
          cout << "A()\n"; }
    A(const A & a){
          cout << "A(const A&)\n"; }
   ~A(){
          cout << "~A()\n"; }
};
```

```
A A::sa;
int main() {
    try { A a1 = A::sa;
        try { throw A(); }
        catch(int){ cout << 111 << endl; }
    }
    catch (...) { cout << " ... \n"; }
    cout << 222 << endl;
    return 0;
}
```

5. Написать на C++ **алгоритм в стиле STL**, который применим к контейнерам, имеющим итератор произвольного доступа (Random_accessIterator). Алгоритм должен распечатывать элементы с 10-го по 5-й из заданного диапазона контейнера (элементы внутри диапазона нумеруются с единицы). Если количество заданных элементов меньше 10, напечатать последний элемент диапазона или 0 в случае пустого диапазона.

6. **а)** Построить приведенную грамматику, эквивалентную заданной грамматике G:

$S \to aABb \mid C$
$B \to bB \mid \varepsilon$
$A \to aA \mid D$
$C \to cCc \mid B$
$D \to dD$

б) Каков тип получившейся грамматики?
в) Какой язык порождает грамматика (формула)?
г) Каков тип этого языка?

Для типов грамматики и языка указать максимально возможный номер по Хомскому.

7. Дана автоматная праволинейная грамматика G:

$S \to 1B$
$B \to 1B \mid 1A$
$A \to 1C$
$C \to 0A \mid \perp$

а) Детерминирован ли разбор по этой грамматике? **Обосновать ответ**.
б) Построить по грамматике конечный автомат и преобразовать его к детерминированному виду по алгоритму НКА → ДКА
в) Построить по получившемуся ДКА леволинейную автоматную грамматику.
г) Эквивалентны ли исходная и получившаяся автоматные грамматики? Обосновать ответ.

8. Дана КС-грамматика G:

$S \to aABc \mid C$
$A \to aB \mid bA$
$B \to bBC \mid \varepsilon$
$C \to cC \mid \varepsilon$

а) Преобразовать грамматику G к неукорачивающей КС-грамматике с помощью алгоритма устранения пустых правых частей в КС грамматике (КС → НКС).
б) Применим ли метод рекурсивного спуска к исходной грамматике? Ответ обосновать.

9. Построить грамматику выражений, содержащих
- левоассоциативные целочисленные бинарные операции ***** и **/** ;
- односимвольные целые числа **0, 1, 2, 3, 4, 5**;
- скобок нет.

Добавить в грамматику действия (только вида **cout <<"символ"**;) по переводу заданных выражений в ПОЛИЗ во время анализа РС-методом.

Приоритет операций ***** и **/** должен быть **задан построенной грамматикой** таким образом, чтобы ПОЛИЗ выражения:

5 * 3 / 2 / 1 * 2 * 1

совпадал с ПОЛИЗом выражения со скобками:

((5 * ((3 / 2) / 1) * 2) * 1).

10. а) Перечислить этапы **создания** программного продукта.
б) Что называют программным продуктом?

Ответы_Вариант 1_2015

1. На этапе 2
- по первому параметру наилучшим образом подходит 3-я функция - на шаге а) – точное отождествление,
- по второму – 2-я - на шаге а) – точное отождествление.

Пересечение пусто, следовательно, вызов неразрешим.

Критерии: Неверное объяснение – **0**.
Достаточно указать выбранную функцию для каждого параметра.

2. 1) – абстрактный класс
2) – класс с единственным явно описанным конструктором в приватной области

Критерии: За каждую неверную\отсутствующую причину или соотв. пример - **– 5**.

3. Ошибки: 1) Разные типы возвращаемого результата функции f()
2) Вызов неконстантного метода для константного объекта ct.

Исправления: 1) Сделать типы результата f() одинаковыми
2) Сделать константным метод g() класса T.

Будет напечатано: f_from_T
g_from_B
g_from_T

Критерии: За каждую неверно найденную или неверно исправленную ошибку - **-5**.
За каждую неверную печать - **-3**

4.
```
A()
A(const A&)
A()
~A()
...
~A()
222
~A()
```

Критерии: За одну ошибку - **7**.
За две ошибки – **3**, за большее количество ошибок – **0.**

5.
```
template <class Random_accessIterator >
void  p_10_5 (Random_accessIterator first, Random_accessIterator last ) {
    if (first == last)
        cout << 0 << endl;
    else
        if (last - first < 10)
            cout << *(last-1) << endl;
        else
            for ( int i = 10; i > 4; )
                cout << first [--i] << endl;
    }
```

Критерии: Отсутствие указания типа допустимого итератора в заголовке шаблона - **-5**.
Ошибки в описании шаблонной функции - **-5** каждая.
Мелкие ошибки - **-2** каждая.

6.

$G_{привед.}$: $S \to C$ | $B \to bB \mid \varepsilon$ | $C \to cCc \mid B$

б) грамматика типа 2 (КС)
в) язык - $L = \{ c^n b^m c^n \mid n >= 0, m >= 0\}$
г) тип языка 2 (КС)

Критерии: За неправильную приведенную грамматику - **– 4**.
За неверный язык, неверный тип грамматики, неверный тип языка - **– 3** за каждый.
Если для неверной грамматики (языка) указан верный тип, считать ответ по пункту б) (г)) верным.

7. а) Нет, т.к. есть альтернативы вывода из B, начинающиеся одинаковыми символами

б) $\delta (S, 1) = B$
$\delta (B, 1) = \underline{AB}$

в) $F = S,\ S = H\ \ S \to C \perp \mid \underline{ABC} \perp$
$C \to A\ 1$

δ (AB, 1) = ABC
δ (ABC, 1) = ABC
δ (ABC, 0) = A
δ (ABC, ⊥) = F
δ (A, 1) = C
δ (C, 0) = A
δ (C, ⊥) = F

A → C 0 | ABC 0
ABC → ABC 1 | AB 1
AB → B 1
B → 1

г) Да, по построению или т.к. порождают один и тот же язык $L=\{1^n(01)^m\perp \mid n >= 3, m >= 0\}$

Критерии: неверный \ без обоснования ответ на пункт а) или б) **- – 4** за каждый,
неверный \ без обоснования ответ на пункт в) или г) **- – 3** за каждый.

8. а) S1 → S | ε
S → aABc | aAc | C
A → aB | bA | a
B → bBC | bB | bC | b
C → cC | c

б) Нет, т.к., например, *first* (C)∩ *follow*(C)={c}≠∅

Критерии: За любую ошибку в пункте а) - **-5**.
За неверный ответ\обоснование в пункте б) - **-5**.

9. E → T {* T < cout << '*' >}
T → F { / F < cout << '/' >}
F → 1 < cout << '1'>| 2 < cout << '2'> | 3 < cout << '3'> | 4 < cout << '4'>| 5 < cout << '5'>

Критерий: Неверная грамматика - **-5**.
Неверно вставленные действия - **-5**.

10. 1) Постановка задачи. Анализ (определение) требований.
2) Проектирование.
3) Написание текста программ (программирование, "кодирование")
4) Компоновка или интеграция программного комплекса.
5) Верификация, тестирование и отладка
6) Документирование

ПП — программа, оформленная, документированная и специфицированная таким образом, что ее можно использовать независимо, отчужденно от автора программы.

Критерий: За каждый пропущенный этап - **-2**.
Неверное понимание ПП - **-3**

Вариант Э2_2015 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std.`**

1	2	3	4	5	6	7	8	9	10

2. При компиляции функции main() следующей программы будет зафиксирована ошибка: "неразрешимый вызов функции f ('a', 0)". Объяснить причину такой ошибки, пользуясь алгоритмом выбора best-matching функции.

```
struct A {
    operator int () { return 1; }
};
void f (double d, char c)   { cout <<  "f (double, char)\n";}
void f (double d,  int  j)   { cout <<  "f (double, int)\n";}
void f (A a,  const char * p)   { cout <<  "f( A, const char *)\n";}
void f (int  i,  const char * p)   { cout <<  "f (int, const char *)\n";}
```

```
int main () {
        A a ;
        f ('a', 0);
return 0;
```

2. Привести пример программы, в которой в результате вызова из функции *main()* метода *f()* некоторого класса через указатель *p* (***p -> f()***) проработает метод ***f()***, описанный в **private** – области другого класса.

3. При компиляции приведенной программы на C++ фиксируются ошибки. Объяснить, в чем они заключаются. Исправить все ошибочные конструкции **только** путем **вставки** или **замены** служебных слов C++. Что будет напечатано в результате работы получившейся правильной программы?

```
struct A {
    virtual void h (int & a) {
            a++;
            cout << "h_from_A\n";  }
    virtual int t () {
            cout << "t_from_A\n ";  return 0;}
};
struct S : A {
     int h (long & a) {
            a++;
            cout << "h_from_S\n";  return 0; }
     int t () {
            cout << "t_from_S\n "; return 0; }
};
```

```
int main() {
      int n = 5;
      const A a1;
      S s1;
      A * p = &s1;
      a1.t ();
      p -> h (n);
      p -> t ();
      cout << "n = " << n << endl;
      return 0;
}
```

4. Что будет напечатано в результате выполнения программы? Компилятор оптимизирующий.

```
struct B {
     B(){  cout << "B()\n"; }
     ~B(){  cout << "~B()\n"; }
};
class A {
       static B sa;
public:
     A(){  cout << "A()\n"; }
     A(const A & a){  cout << "A(const A&)\n"; }
    ~A(){  cout << "~A()\n"; }
};
```

```
B A::sa;
int main() {
   try {A a1;
          try {  throw B();  }
          catch (A &) { cout << 333 << endl; }
   }
   catch (...) { cout << " ... \n"; }
   cout << " End\n ";
   return 0;
}
```

5. Написать **алгоритм в стиле STL**, который применим к контейнерам, имеющим двунаправленый итератор (BidirectionalIterator). Алгоритм должен распечатывать 3-й с конца элемент заданного диапазона контейнера. Если количество заданных элементов меньше 3, напечатать первый заданный элемент или строку `"Empty\n"` в случае пустого диапазона.

6. **а)** Построить приведенную грамматику, эквивалентную заданной грамматике G:

$$S \to aAb \mid CB$$
$$B \to bB$$
$$A \to aA \mid D$$
$$C \to cCc \mid B \mid c$$
$$D \to d$$

б) Каков тип получившейся грамматики?
в) Какой язык порождает грамматика (формула)?
г) Каков тип этого языка?
Для типов грамматики и языка указать максимально возможный номер по Хомскому.

7. Дана автоматная леволинейная грамматика G:

$$S \to C\perp$$
$$A \to Aa \mid a$$
$$B \to Aa \mid b$$
$$C \to Ba$$

а) Детерминирован ли разбор по этой грамматике? **Обосновать ответ**.
б) Построить по грамматике конечный автомат и преобразовать его к детерминированному виду по алгоритму НКА → ДКА
в) Построить по получившемуся ДКА праволинейную автоматную грамматику.
г) Эквивалентны ли исходная и получившаяся автоматные грамматики? **Обосновать ответ**.

8. Дана КС-грамматика G:

$S \to A \mid B$
$A \to aA \mid \varepsilon$
$B \to bB \mid cC \mid \varepsilon$
$C \to aABC \mid c$

а) Преобразовать грамматику G к неукорачивающей КС-грамматике с помощью алгоритма устранения пустых правых частей в КС грамматике (КС → НКС).
б) Применим ли метод рекурсивного спуска к исходной грамматике? **Ответ обосновать**.

9. Построить грамматику выражений, содержащих
- левоассоциативные целочисленные бинарные операции + и – ;
- односимвольные целые числа **0, 2, 4, 6** ;
- скобок нет.

Добавить в грамматику действия (только вида **cout <<"символ";**) по переводу заданных выражений в ПОЛИЗ во время анализа РС-методом.

Приоритет операций + и – должен быть **задан построенной грамматикой** таким образом, чтобы ПОЛИЗ выражения:
2 + 4 – 6 – 0 + 4 + 2
совпадал с ПОЛИЗом выражения со скобками:
((2+((4–6)–0)+4)+2).

10. **а)** Назвать основные задачи редактора связей.
б) Привести пример ситуации, в которой редактор связей выдаст сообщение об ошибке.

Ответы_Вариант 2_2015

1. На этапе 2
- по первому параметру наилучшим образом подходит последняя функция - на шаге б) – целочисленное расширение,
- по второму – вторая - на шаге а) – точное отождествление.
Пересечение пусто, следовательно, вызов неразрешим.

Критерии: Неверное объяснение – **0**.
Достаточно указать выбранную функцию для каждого параметра.

2. Надо в базовом классе описать виртуальную функцию в **public** – области, а в производном классе – функцию с таким же прототипом в **private** – области. Затем в main() описать объект производного класса и указатель на базовый класс, инициализированный адресом объекта производного класса. Далее через указатель вызвать соответствующий метод.

Критерии:

3. Ошибки: 1). неконстантный метод t () вызывается для константного объекта a1.
Исправления: 1). сделать метод t () класса A константным (добавить const)
Будет напечатано:
t_from_A
h_from_A
t_from_A
n = 6

Критерии: За каждую неверно найденную или неверно исправленную ошибку - **-5**.
За каждую неверную печать: **-3**

4.
```
B()
A()
B()
 ~A()
 ...
~B()
End
~B()
```

Критерии: За одну ошибку - **7**.
За две ошибки – **3**, за большее количество ошибок – **0.**

5.
```
template <class BidirectionalIterator>
  void p3 (BidirectionalIterator first, BidirectionalIterator last) {
    BidirectionalIterator tt = last;
    i = 0;
    while (tt != first) {  i++;  tt--;  }
    if (i == 0)  cout << " Empty\n";
    else
            if (i <3) cout << *first << endl;
            else {
                    last--; last--; last--;
                    cout << *last << endl;
            }
  }
```

Критерии: Отсутствие указания типа допустимого итератора в заголовке шаблона - **-5**.
Ошибки в описании шаблонной функции - **-5** каждая.
Мелкие ошибки - **-2** каждая.

6. $G_{привед.}$: $S \to aAb$
$A \to aA \mid D$
$D \to d$

б) грамматика типа 2 (КС)
в) язык - $L = \{ a^n d\, b \mid n >= 1, m >= 0\}$
г) тип языка 3 (Рег.)

Критерии: За неправильную приведенную грамматику - **–4**.
За неверный язык, неверный тип грамматики , неверный тип языка - **–3** за каждый.
Если для неверной грамматики (языка) указан ее верный тип, считать ответ по пункту б) (г)) верным.

7. а) Нет, т.к. есть одинаковые правые части в правилах вывода грамматики

б)
δ (H, a) = A
δ (H, b) = B
δ (A, a) = $\underline{AB}$
δ (B, a) = C
δ ($\underline{AB}$, a) = $\underline{ABC}$
δ ($\underline{ABC}$, a) = $\underline{ABC}$
δ ($\underline{ABC}$, $\perp$) = S
δ (C, $\perp$) = S

в) H = S, S = F
$S \to aA \mid bB$
$B \to aC$
$A \to a\,\underline{AB}$
$AB \to a\,\underline{ABC}$
$\underline{ABC} \to a\,\underline{ABC} \mid \perp$
$C \to \perp$

г) Да, по построению или т.к. порождают один и тот же язык $L=\{ba\} \cup \{a^n\perp \mid n >= 3\}$

Критерии: неверный\без обоснования ответ на пункт а) или б) - **– 4** за каждый,
неверный\без обоснования ответ на пункт в) или г) - **– 3** за каждый.

8. а)
$S1 \to S \mid \varepsilon$
$S \to A \mid B$
$A \to aA \mid a$

B → bB | b | cC
C → aABc | aAC | aBC | aC | c

б) Нет, т.к. из двух альтернатив нетерминала S выводится ε (или т.к. $first(A) \cap fillow(A) = \{a\} \neq \varnothing$)

Критерии: За любую ошибку в пункте а) - **-5**.
За неверный ответ\обоснование в пункте б) - **-5**.

9. E → T {+ T < cout << '+' >}
T → F { - F < cout << '-' >}
F → 0 < cout << '0'>| 2 < cout << '2'> | 4 < cout << '4'>| 6 < cout << '6'>

Критерий: Неверная грамматика - **-5**.
Неверно вставленные действия - **-5**.

10. 1. связывает между собой по внешним данным объектные модули, порождаемые компилятором и составляющие единую программу,
1. связывает файлы статически подключаемых библиотек с целью получения единого исполняемого модуля,
3. готовит таблицу трансляции относительных адресов для загрузчика,
4. готовит таблицу точек вызова функций динамически подключаемых библиотек.

Пример: отсутствие описания объявленного имени функции при его использовании.

Критерии: За отсутствующий или неверный пункт 1-3 - **-3**,
За отсутствующий или неверный пункт 4 - **-1**,
Нет\неверный пример - **-3**.

2016 год
Коллоквиум

Вариант 1_2016 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. Вычеркните только те строки функции *main ()*, которые приводят к синтаксической ошибке (если таковые имеются). Обязательно **объясните** причины ошибок. Что будет напечатано в результате работы получившейся программы.

```
struct A {
      int n;
      A(){ n = 9;}
      A operator+(const A & a){
            A t;
            t.n = n + a.n;
            return t;     }
};
struct B:A {
      B operator+(const B & a){
            B t;
            t.n = n + a.n + 10;
            return t;   }
};
```

```
int main () {
      A a;
      B b;
      a = a + b;
      cout << a.n << '\n';
      b = b + a;
      cout << b.n << '\n';
      return 0;
}
```

2. Исправьте **только описание класса A** так, чтобы в приведенной ниже программе не было ошибок, а на экран напечаталось **f (int, int)**.

```
struct A {  int n;
            A (int  m) { n = m; }
            operator int () { return 1; }
};
void  f ( int i,  int  j )  { cout << "f(int,  int)\n";}
void  f ( A b,  A a )  { cout << "f(A,  A)\n";}

int main () {          A a (1);
                       f (a, 1);
                       return 0;   }
```

3. Опишите необходимые классы так, чтобы в заданной функции *main ()* не было ошибок, а на экран печаталось **310.**

```
int main () {
        T t (3);
        P <T> p1(&t), p2(0);
        cout  <<  p1 -> n  <<  p2 -> n  <<  endl;
        return 0 ; }
```

4. Добавьте в описание класса A **из задачи 1** метод

virtual A& **operator** - (){ n = - (n+1)**;** cout << n <<**"** A::**operator** - ()\n**"; return *this;** },

а в описание (производного от класса A) класса B из задачи 1 -

B& **operator** - () { cout << n << **"** B::**operator** - () \n**"; return *this;** }

Что напечатает программа с нижеприведенной функцией *main()* и получившимися классами A и B?

```
int main () {

        B b;
        A  a,  & ra = b;
        a = -a;
        ra = -ra;
        ra = a;
        ra = -ra;     return 0;
}
```

5. Что такое абстрактный класс в C++?
Обязательно приведите **пример** описания и использования абстрактного класса.

6. Укажите все прототипы конструкторов и деструкторов в порядке их выполнения в следующей программе:

```
class A {};
```

```
class B: public A{};
struct  D { A a; };

int main () {
       D d;
       { d.a = B();
         B b; }
       A a1, a2 = a1;
       return 0;
}
```

7. Добавьте в следующую программу необходимые описания так, чтобы в ней не было ошибок.

```
    int main () {
            const A a;
            a.x = 3;
            cout << a.y << a.x << a.f(1) << endl;
            return 0; }
```

8. Модифицируйте функцию *main()*, **ничего в ней не удаляя, не используя комментарии, goto и прочие переходы** так, чтобы программа завершалась нормально (не аварийно).

```
struct B { virtual void f () { cout << "B::f()\n"; } };
struct D : B { void f () { cout << "D::f()\n"; } };

int main () {
        D d, &rd  = d;
        B b, & rb = b, &rbd = d;
        rd = dynamic_cast <D&> (rbd);  rd.f();
        rd = dynamic_cast <D&> (rb);   rd.f();
        return 0;
}
```

9. C++11. Вычеркните неверные конструкции в следующей программе на C++11.
Обязательно **объясните** причины ошибок в вычеркнутых конструкциях.

```
struct A {
        int  i = 10;
};

void f (A && x) { }

int main () {
        A a;
        int && n, m;
        auto b = A();
        decltype (a) c;
        f (b);
        f ( A() );
        m = nullptr;
        return 0;
}
```

10. **STL.** Напишите шаблонную функцию, подсчитывающую сумму пяти последних элементов заданного контейнера (списка или вектора, константного или неконстантного). Тип элементов контейнера является числовым типом. Если в заданном контейнере меньше 5 элементов, функция должна вернуть 0 соответствующего типа.

Ответы_Вариант 1_2016

1. **18**
9

Вычеркнуть строку b = b+a; поскольку фактическим параметром при вызове метода **operator**+ (**const** B& b) из класса B является объект класса A.

Приведение типа A к произв. от A типу B приводит к синт. ошибке.

Критерии: Ошибка не найдена или неправильно объяснена – **0**.
Лишнее вычеркивание, неверный ответ **– 5 за каждую ненаведенную ошибку**

4 .

Надо добавить **explicit** в конструктор преобразования.

Критерии: Правильный ответ – **10**, иначе - **0**.

3.

```
template <class TT>      class P {
public:
        TT * p, * p_null;
        P (TT * t) {  p = t; }
        TT* operator -> ( ) {
                if (p)   return p;
                else     return p_null = new TT;
        }
        ~P() { if (!p) delete p_null; }
};
```

```
struct T {
        int n;
        T (int k = 10){n = k;}
        };
```

Критерии: За неверную\отсутствующую операцию **->** **–7**.
За ошибки в шаблоне и др. - **-2** за каждую погрешность.
За отсутствие деструктора и освобождение дин. памяти не наказывать.

4.

```
-10  A::operator - ()
  9  B::operator - ()
-10  B::operator - ()
```

Критерии: Ошибки в ответе – **0**, иначе – **10**.
Ошибки, не связанные с виртуальностью - **-2**.

5.

Абстрактный класс – класс, содержащий хотя бы одну чистую виртуальную функцию.

Пример:

. . .

Критерии: За неправильное определение - 0.
За отсутствие\неверный пример - **–5**.

6.

```
A(), D(), A(), B(), ~B(), ~A(), A(), B(), ~B(), ~A(), A(), A(const A&),
~A(), ~A(), ~D(), ~A()
```

Критерии: За каждый лишний/пропущенный/переставленный конструктор/деструктор – **-4**

7.

Например,

```
struct A {
        int y;
        static int x;
        static int  f (int k) { return x + k; } // или  int  f (int k) const{ return x + k; }
```

```
        A(){ y = 5; }
};
int A::x = 0;      !!!
```

Критерии: Нет **static** при описании поля x - **-5**.
Нет **static** или **const** при описании метода f(int) - **-5**.
Не описано статическое поле x вне класса - **-5.**
Если поле y нестатическое и нет явного конструктора - -1.
Другие ошибки - **-2** за каждую.

8.
Модификация:
Заключить в **try-catch** блок использование операций (и) **dynamic_cast**, используя **catch** (bad_cast){....}

Критерии: Правильный ответ – **10**, иначе – **0**.

9.
Неверные конструкции:

int && n;	- r-value ссылка должна быть инициализирована;
f (b);	- нельзя передавать l-value выражение по r-value ссылке;
n = **nullptr**;	- тип nullptr_t не приводится к целому типу.

Критерии: За каждое неверное\отсутствующее\без_объяснения вычеркивание **– -4**.

10.

```
template <class T>
typename T::value_type sum5(const T& t){
      typename T::value_type s = 0;
      typename T::const_iterator p = t.end();
      for (int i = 0; i < 5; i++) {
            if (p == t.begin())
                  return 0;
            s += *--p;
      }
      return s;
}
```

Критерии: За пропуск ***typename*** - **-5 (за все вместе)**.
За использование неконстантного итератора - **-5**.
За ошибки в алгоритме - **-5**.
За другие ошибки - **-2** за каждую.

Вариант 2_2016 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std.`**

2. Вычеркните только те строки функции *main ()*, которые приводят к синтаксической ошибке (если таковые имеются). Обязательно **объясните** причины ошибок. Что будет напечатано в результате работы получившейся программы.

```
struct T {
        int i;
        T(){i = 3;}
        T(int t){ i = t;}
        T operator*(const T & a){
                T t;
                t.i = i * a.i;
                return t;     }
};
struct P:T {
        P operator*(const P & a){
                P t;
                t.i = i * a.i * 2;
                return t;   }
};
```

```
int main () {

        T a(5);
        P b;
        a = a * b;
        cout << a.i << '\n';
        b = b * a;
        cout << b.i << '\n';
        return 0;
}
```

3. Исправьте **только описание класса A** так, чтобы в приведенной ниже программе не было ошибок, а на экран напечаталось **f (A, A)**.

```
struct A {  int n;
            A (int  m) { n = m; }
            operator int () { return 1; }
};
void  f ( int i,  int  j )  { cout << "f(int,  int)\n";}
void  f ( A b,  A a )  { cout << "f(A,  A)\n";}

int main () {          A a (1);
                       f (a, 1);
                       return 0;   }
```

3. Опишите необходимые классы так, чтобы в заданной функции *main ()* не было ошибок, а на экран печаталось **520.**

```
int main () {
        C c (5);
        B <C> b1(&c), b2(0);
        cout  <<  (*b1). n  <<  (*b2). n  <<  endl;
        return 0 ; }
```

4. Добавьте в описание класса T **из задачи 1** метод

virtual T& **operator --** (){ i = i - 1; cout << i <<**"** T::**operator** -- () \n**"; return *this;**} ,

а в описание (производного от класса T) класса P из задачи 1 -

P& **operator --** () { cout << i << **"** P::**operator** -- () \n**"; return *this;** }

Что напечатает программа с нижеприведенной функцией *main()* и получившимися классами T и P?

```
int main () {
        P b;
        T  a(5),  & rt = b;
        --a;
        --rt;
        rt = a;
        --rt;     return 0;
}
```

5. Что такое функция-друг в языке Си++? Приведите **пример** описания и использования функции-друга.

6. Укажите все прототипы конструкторов и деструкторов в порядке их выполнения в следующей программе:

```
class A {};
class B: public A{};
struct T { B * pb; };

int main () {
       A a;
       { a = B();
         T t; }
       B b1, b2 = b1;
       return 0; }
```

7. Добавьте в следующую программу необходимые описания так, чтобы в ней не было ошибок.

```
    int main () {
            B::y = 3 ;
            const B b;
            cout << b.x-B::h(2)<< b.h(0)<< endl;
            return 0; }
```

8. Модифицируйте функцию *main()*, **ничего в ней не удаляя, не используя комментарии, goto и прочие переходы** так, чтобы программа завершалась нормально (не аварийно).

```
struct B { virtual void  g() { cout << "B::g () \n"; } };
struct D : B { };

int main () {
        D d, * pd1, *pd2;
        B b, * pb = &b, * pbd = &d;
        pd1 = dynamic_cast < D* > (pbd);
        pd2 = dynamic_cast < D* > (pb);
        if ( typeid (*pd1) == typeid (*pd2) )    pb -> g () ;
        return 0; }
```

10.C++11. Вычеркните неверные конструкции в следующей программе на C++11.
Обязательно **объясните** причины ошибок в вычеркнутых конструкциях.

```
struct S {
        int  x = 1;
};

void  g (S && s) { cout << s.x << endl;}

int main () {
        int && i;
        auto k = nullptr;
        decltype ( S ( ) ) a;
        char * s = k;
        k = s;
        g (a);
        g ( S() );
        return 0;
}
```

10. **STL.** Напишите шаблонную функцию, подсчитывающую сумму первых семи элементов заданного контейнера (списка или вектора, константного или неконстантного). Тип элементов контейнера является числовым типом. Если в заданном контейнере меньше 7 элементов, функция должна вернуть 0 соответствующего типа.

Ответы_Вариант 2_2016

2. **15**
3

Вычеркнуть строку b = b*a; поскольку фактическим параметром при вызове метода **operator*** (**const** P& b) из класса P является объект класса T.

Приведение типа T к произв. от T типу P приводит к синт. ошибке.

Критерии: Ошибка не найдена или неправильно объяснена – **0**.
Лишнее вычеркивание, неверный ответ **– 5 за каждую ненаведенную ошибку**

5.

Надо удалить функцию преобразования **operator int**().

Критерии: Правильный ответ – **10**, иначе – **0**.

3.

```
template <class T>        class B {
public:
        T * p, * p_null;
        B (T * t) {  p = t; }
        T& operator *( ) {
                if (p)   return *p;
                else     return *(p_null = new T);
        }
        ~B() { if (!p) delete p_null; }
};
```

```
struct C {
        int n;
        C (int k = 20){ n = k; }
        };
```

Критерии: За неверную\отсутствующую операцию * **–7**.
За ошибки в шаблоне и др. - **-2** за каждую погрешность.
За отсутствие деструктора и освобождение дин. памяти не наказывать.

4.

```
4  T::operator -- ()
3  P::operator -- ()
4  P::operator -- ()
```

Критерии: Ошибки в ответе – **0**, иначе – **10**.
Случайные неточности, не связанные с виртуальностью - **-2**.

5.

Функция-друг класса – внешняя по отношению к классу функция, которой разрешен доступ к закрытым и защищенным членам класса.

Пример: ....

Критерии: Неверное определение - **0**.
Неверный\отсутствует пример **- -5**.

6.

```
A(), A(), B(), ~B(), ~A(), T(), ~T(), A(), B(), A(const A&), B(const B&);
   ~B(), ~A(), ~B(), ~A(), ~A()
```

Критерии: За каждый лишний/пропущенный/переставленный конструктор/деструктор – **-4.**

7. Например,

```
struct  B {
        int x;
        static int y;
        static int  h (int k) {return y*k;}
```

```
        B ( ) { x  = 10;}
};
int B::y = 1;        !!!
```

Критерии: Нет **static** при описании поля y - **-5**.
Нет **static** при описании метода h(int) - **-5**.
Не описано статическое поле y вне класса - **-5.**
Если поле x нестатическое и нет явного конструктора - **-1.**
Другие ошибки - **-2** за каждую.

8.
Модификация: Заключить в **try-catch** блок условный оператор в функции main () ,
используя **catch** (bad_typeid){....}

Критерии: Правильный ответ – **10**, иначе – **0**.

10. Неверные конструкции:

int && i; - r-value ссылка должна быть инициализирована;
f (a); - нельзя передавать l-value выражение по r-value ссылке;
k = s; - тип char * не приводится к nullptr_t .

Критерии: За каждое неверное\отсутствующее\без_объяснения вычеркивание – **-4**.

10.

```
template <class T>
        typename T::value_type sum7(const T& t){
              typename T::value_type s = 0;
              typename T::const_iterator p = t.begin();
              for (int i = 0; i < 7; i++) {
                    if (p == t.end())
                          return 0;
                    s += *p++;
              }
              return s;
        }
```

Критерии: За пропуск ***typename*** - **-5 (за все вместе)**.
За использование неконстантного итератора - **-5**.
За ошибки в алгоритме - **-5**.
За другие ошибки - **-2** за каждую.

2016 год
Экзамен

Вариант 1_2016 Ф.И.О._____№ группы_____

1. Ответьте на следующие вопросы. Обоснуйте ответ на каждый вопрос.

1	2	3	4	5	6	7	8	9	10

(1) Верно ли, что каждая регулярная грамматика является
неукорачивающей грамматикой?
Ответ:

(2) Верно ли, что каждый регулярный язык является контекстно-зависимым языком?
Ответ:

2. Дана регулярная грамматика G :

$S \to Bc \mid Ac$

$B \to Ba \mid Bb \mid Aa \mid Ab$

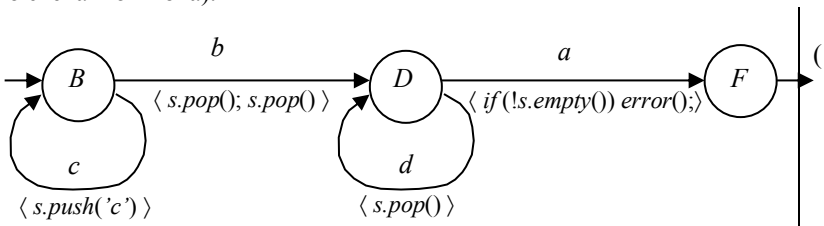
$A \to b \mid Ba$

(1) Сколько состояний будет иметь ДКА A_1, построенный по регулярной грамматике G (с проведением детерминизации, если нужно) ? *Ответ*:_____

(2) Построить ДКА A_2 с тремя состояниями, такой что $L(A_1)=L(A_2)$.
Ответ:

3. Дан перечень понятий: *отладчик, редактор связей, транслятор, драйвер, макрогенератор, редактор текстов, библиотеки стандартных подпрограмм*. Вычеркнуть из перечня одно лишнее понятие и назвать термин (его нет в перечне), в который входят все оставшиеся понятия.
Ответ:

4. Дан автомат A с действиями над стеком s типа *stack<char>* (полагаем, что стек потенциально бесконечен, *pop*() из пустого стека – ошибка):



(1) Привести формулу языка $L(A)$
Ответ:

(2) С помощью КС-грамматики, анализируемой методом рекурсивного спуска, с действиями только вида ⟨ *cout* <<'СИМВОЛ'; ⟩ реализовать формальный перевод $\tau = \{ (x, y) \mid x \in L(A), y = x^R \}$
Ответ:

5. Дана польская запись фрагмента программы на Си, в которой пропущены некоторые метки и операции перехода ('!','!F'). Исходный фрагмент содержал одну логическую операцию (&& или ||) и не содержал ни одного оператора перехода (*goto*, *break* и т.п.). Вставьте пропущенные элементы в ПОЛИЗе и восстановите фрагмент на Си.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&x	x	1	+	=	;	a	b	==	17	!F	b	x	<	18	!	0

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
		&y	#+	;	y	p	q	p	–	+	<		!F		!	

Ответ:

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std.`**

6. Объясните все роли ключевого слова **throw** языка C++ в различных смысловых контекстах. Например, знак «–» в зависимости от контекста играет роли унарного или бинарного минуса.

Ответ:

7. Вычеркните, если есть, ошибочные вызовы функций. Что напечатает получившаяся программа?

```
void g (const int & i ) { cout << " g(const int & ) \n"; }
void g ( int i ) { cout << " g( int ) \n"; }

int main () {   int a=48;   const int b=10;
   f((char) a);
   g((char) a);
```

```
void  f ( int  x )     { cout <<  " f (  int   )\n";}
void  f ( short  x )   { cout <<  " f ( short  )\n";}
```

```
    f(short(b));
    g(b);
}
```

Ответ:

8. Можно ли продемонстрировать динамический полиморфизм с помощью оператора-выражения вида $(b+a)+b$; ? Если да – приведите соответствующий пример, если нет – объясните почему. Перечислите другие виды полиморфизма в C++.

Ответ:

9. Опишите один класс A с не более чем тремя явно описанными методами (включая специальные) так, чтобы программа с данной функцией main (и **без** каких-либо директив) успешно выполнилась.

```
int main () {
  A a;
  A(a, a);
  A((a, a));
}
```

Ответ:

10. Объясните, что делает данная программа:

```
vector<bool> v(10);

int main() { int k;
   for(int i=0; i< v.size(); i++) {cin>>k; v[i]=k%2;}
   for(vector<bool>::const_iterator p = v.begin(); p!=v.end(); ++p)
       cout << *p;
}
```

Ответ:

Вариант 1_2016 **Ответы**

За каждую задачу максимум 10 баллов. При вычитании баллов полагаем 0-x=0 для любого x.

1. Ответьте на следующие вопросы. Обоснуйте ответ на каждый вопрос.

(1) Верно ли, что каждая регулярная грамматика является неукорачивающей грамматикой?

Ответ: Не верно. Регулярная грамматика $S \to aS \mid \varepsilon$ не является неукорачивающей (в соответствии с определением некуорачивающей гр-ки)

(2) Верно ли, что каждый регулярный язык является контекстно-зависимым языком?

Ответ: Верно. Если язык регулярный, то он задается регулярной грамматикой. Алгоритм устранения пустых правых частей дает эквивалентную регулярную грамматику, которая удовлетворяет определению КЗ-грамматики. Следовательно, регулярный язык КЗ.

Критерии: неверный ответ или не обоснованный или ошибка в обосновании: -5 за каждый пункт

2. Дана регулярная грамматика G :

$S \to Bc \mid Ac$

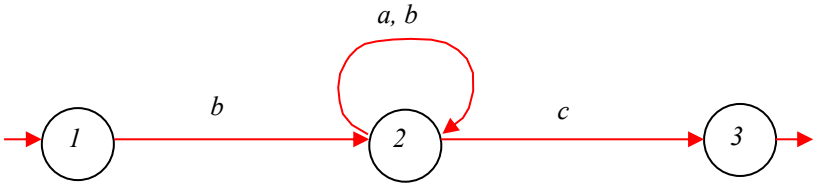
$B \to Ba \mid Bb \mid Aa \mid Ab$

$A \to b \mid Ba$

(3) Сколько состояний будет иметь ДКА A_1, построенный по регулярной грамматике G (с проведением детерминизации, если нужно) ? *Ответ*:_____5___

(4) Построить ДКА A_2 с тремя состояниями, такой что $L(A_1)=L(A_2)$.

Ответ:



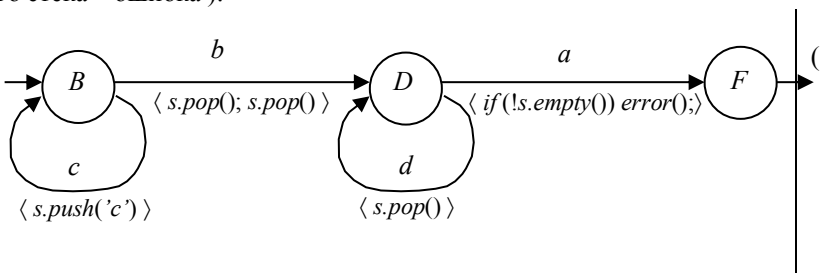
Критерии: -5 за каждый неверно отвеченный пункт

3. Дан перечень понятий: *отладчик, редактор связей, транслятор, ~~драйвер,~~ макрогенератор, редактор текстов, библиотеки стандатрных подпрограмм.* Вычеркнуть из перечня одно лишнее понятие и назвать термин (его нет в перечне), в который входят только все оставшиеся понятия.

Ответ: Система программирования

Критерии: -5 за неправильное вычеркивание, -8 за неправильный термин

4. Дан автомат A с действиями над стеком s типа *stack<char>* (полагаем, что стек потенциально бесконечен, *pop*() из пустого стека – ошибка):



(1) Привести формулу языка $L(A)$

Ответ:

$\{c^{n+2}bd^na \mid n \geq 0\}$

(2) С помощью КС-грамматики, анализируемой методом рекурсивного спуска, с действиями только вида ⟨ *cout* <<'СИМВОЛ'; ⟩ реализовать формальный перевод $\tau = \{ (x, y) \mid x \in L(A), y = x^R \}$

Ответ: $S \to cc \langle a \rangle Ba \langle cc \rangle$

$B \to c \langle d \rangle Bd \langle c \rangle \mid b \langle b \rangle$

Критерии: -5 за каждый неверно отвеченный пункт

5. Дана польская запись фрагмента программы на Си, в которой пропущены некоторые метки и операции перехода ('!','!F'). Исходный фрагмент содержал одну логическую операцию (&& или ||) и не содержал ни одного оператора перехода (*goto*, *break* и т.п.). Вставьте пропущенные элементы в ПОЛИЗе и восстановите фрагмент на Си.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ПОЛИЗ	&x	x	1	+	=	;	a	b	==	17	!F	b	x	<	18	!	0

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
23	!F	&y	#+	;	y	p	q	p	–	+	<	34	!F	1	!	

Ответ: **do { x=x+1; if (a==b && b<x) y++; } while** (y < p +(q – p));

Критерии: -8 за неверно заполненные элементы

-5 за неверно восстановленный фрагмент

-2 за отсутствие скобок в p +(q – p)

6. Объясните все роли ключевого слова **throw** языка C++ в различных смысловых контекстах. Например, знак «–» в зависимости от контекста играет роли унарного или бинарного минуса.

Ответ: 1) задает список исключений в заголовке функции
2) в операторе с параметром – генерирует исключение
3) в операторе без параметра – «пробрасывает» текущее обрабатываемое исключение в объемлющий уровень обработки; если текущего исключения нет, то приводит к ненорамальному завершению программы.

7. Вычеркните, если есть, ошибочные вызовы функций. Что напечатает получившаяся программа?

```
void g (const int & i ) { cout << " g(const int & ) \n"; }
void g ( int i ) { cout << " g( int ) \n"; }
void f ( int x ) { cout << " f ( int )\n";}
void f ( short x ) { cout << " f ( short )\n";}
```

```
int main () {   int a=48;   const int b=10;
   f((char) a);
   g((char) a);
   f(short(b));
   g(b);
}
```

Ответ: f (int)
f (short)

8. Можно ли продемонстрировать динамический полиморфизм с помощью оператора-выражения вида (*a*+*b*)+*a*; ? Если да – приведите соответствующий пример, если нет – объясните почему. Перечислите другие виды полиморфизма в C++.

Ответ: Другие виды полиморфизма – статический, параметрический.

Динамический продемонстрировать можно. Компилятор трактует выражение (*a*+*b*)+*a* как (*a.operator* +(*b*)).*operator*+(*a*). Полиморфизм для операции означает, что в зависимости от типа операндов вызываются разные методы с одинаковым обозначением операции +. Чтобы сработал именно динамический полиморфизм, методы должны вызываться через ТВМ, в данном выражении это возможно, если *a* и *b* являются ссылками на базовый класс для разных полиморфных объектов из иерархии. Приведем возможный пример:

```
class A{
   public:
   virtual A& operator+(A& a){cout << "+ from A | "; return a;}
};
class B:public A {
   public:
   A& operator+(A& a){ cout << "+ from B | "; return *this;}
};
int main() {
 A a1;
 B b1;
 A & a=a1, & b=b1;

(a+b)+a ; // (a.operator+(b)).operator+(a);
          // Вызов виртуального метода через ссылку на базовый класс,  используется ТВМ
          // Напечатается:  + from A | + from B |
return 0;
}
```

9. Опишите один класс A с не более чем тремя явно описанными методами (включая специальные) так, чтобы программа с данной функцией main (и **без** каких-либо директив) успешно выполнилась.

```
int main () {
   A a;
   A(a, a);
   A((a, a));
 }
```

Ответ:

```
struct A {
  A (int a=0, int b=0) {}
  operator int() {return 1;}
  A operator ,(A a){ return a; }
};
```

10. Описать **функцию-шаблон (shift_R)**, которая по первому аргуметну-контейнеру строит второй аргумент-контейнер путем циклического сдвига всех элементов первого на один вправо. Содержимое первого аргумента-конейнера не изменяется. Старое содержимое второго аргумента-контейнера стирается. Использование **shift_R должно быть корректно** в следующей функции:

```
void  sR (const vector <int> & V, list <int> & L){
            shift_R (V, L);
          }
```

Ответ:

2017 год
Коллоквиум

Вариант 1_2017 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Вычеркните из функции main блоки, содержащие ошибки компиляции (предупреждения warning не считаем ошибкой). Что будет напечатано получившейся после вычеркивания программой? **Примечание**: считаем, что указатель можно без потерь преобразовать в long, а форма с многоточием поддерживает передачу любых объектов.

```
#include <iostream>
using namespace std;
struct A {
  A(int
 a=0){cout<<1;}
  A(const A&a){
      cout<<2;}
 ~A(){cout<<endl;}
};
```

```
struct B:  A {
  B(B& b){cout<<3;}
  B(const B& b){cout<<4;}
};

struct C:  B {
  C(...):B(*this){cout<<5;}
};
```

```
int main() {
  {A
 a((long)&a);}
  {A d=d=d;}
  {B b;}
  {const B b=b;}
  {C c(1); }
  {return 0;}
}
```

Ответ:

2. Измените данную программу, добавляя только операции разрешения области видимости :: так, чтобы было напечатано число 39.

```
#include <iostream>

using namespace std;
int x=0;
namespace X {  // int x=39;
    namespace Y { int x=17; }
    const int x=9;
}
```

```
struct A { int x;
    struct B{ int x; B(){ x=39;}} x1;
    A(){x=22;}
public:
    int h(){ { int x=39;}
        return     x +     x +     x; }
};
int main(){ A a; cout<< a.h(); return 0;}
```

3. Не изменяя оператор return и не используя символы /, \, ″, %, #, добавьте всё необходимое в описание класса A так, чтобы компиляция прошла без ошибок.

```
class A { public:  int f(){      return (i,j)[i,j](i,j);}
};
```

Ответ:

4. Перечислите имена классов, описания которых содержат ошибки.

```
struct A {
  int register f(){
      return 1;}
  int const f(int x){
      return 2;}
};
```

```
struct B {
  static int  f(){
    return 1;}
  int  f() const{
    return 2;}
};
```

```
struct C {
  static int
  f(const int& x ){
    return 1;}
  int  f(int& x) {
    return 2;}
};
```

```
struct D {

  static virtual
  int f() {
    return 0 ;}
};
```

Ответ:

5. Что напечатает данная программа?

```
#include <iostream>
#include <typeinfo>
using namespace std;
class L { public: virtual void f (char x='L') {cout <<x<< "0"<<endl;}};
class P : public L { public: void f (char x='P') {cout <<x<< "1"<<endl;}};
int main() { P p; L * pl=&p; pl->f(); dynamic_cast<P*>(pl)->f(); return 0; }
```

Ответ:

6. Что будет напечатано в результате выполнения следующей программы?

```
#include <iostream>
using namespace std;
int f ( int n){
 try {  if (n<=0) throw n; f(--n); }
    catch(int&  k){cout<<k++;  throw;  throw
 "b";}
    catch(const char *s){cout<<s;}
 throw "d";
}
```

```
int main() {
  try { cout<<'a'; f(2); cout<<'c';}
    catch (int k){cout<<k;}
    catch(const char *s){cout<<s;}
  cout<<'e';
  return 0;
}
```

Ответ:

7. Объясните ошибки компиляции для данной программы. Не исправляя имеющиеся строки, вставьте новую строку кода, в которой нет символов /, \, ″, %, #, (,), так, чтобы программа скомпилировалась. **Примечание**: угловые скобки использовать можно.

```
typedef int T;

T p( T a){ return a;}

float p( float a){ return a;}
```

```
int main() {
 return int(p('a')+p(3.14));
}
```

8. Может ли абстрактный класс в языке C++ являться абстрактным типом данных? Если да, то приведите пример, если нет – объясните почему. *Ответ:*

9. Добавьте в main() один оператор так, чтобы при выполнении программы произошла ошибка обращения по ссылке к уже не существующему объекту. Объясните причину ошибки. **Примечание:** считаем, что повторный вызов деструктора и обращение к членам объекта после его работы сами по себе не являются ошибками.

```
#include <iostream>
```

```
using namespace std;
class List { char elem; List & next;
public:
    List(): elem('\n'), next(*this){}
    List(const char*s): elem(*s? *s:'\n'), next(*s? * new List(s+1): *this){}
    List(const List& l):  elem(l.elem), next(&l!=&l.next? * new List(l.next): *this){}
    void print()const {cout<<elem; if(&next!=this) next.print();}
   ~List(){ if(&next!=this) {delete &next;}  }
};
int main() { List b("abc");           b.print(); }
```

10. Приведите функцию, эквивалентную данной, не содержащую новых конструкций из C++11.

```
#include <list>
int f(int x, const std::list<int>& l){
  auto p=l.begin();
  decltype(x) s=0;
  while(p!=l.end()) s+=*p++==x;
  return s;
}
```

Ответ:

Вариант 2_2017 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Вычеркните из функции main блоки, содержащие ошибки компиляции (предупреждения warning не считаем ошибкой). Что будет напечатано получившейся после вычеркивания программой? **Примечание**: считаем, что указатель можно без потерь преобразовать в long, а форма с многоточием поддерживает передачу любых объектов.

```
#include <iostream>
using namespace std;
struct X {
  X(int
 x=0){cout<<1;}
  X(const X&x){
      cout<<2;}
 ~X(){cout<<endl;}
};
```

```
struct Y:  X {
  Y(){cout<<3;}
  Y(Y& y):X(*this){
     cout<<4;}
};
struct Z:  Y {
  Z(...){cout<<5;}
};
```

```
int main() {
  {X x((long)&x);}
  {Y c(c=c);}
  {const X b=b;}
  {X p; Z q=p;}
  {Z z; }
  {return 0;}
}
```

Ответ:

2. Измените данную программу, добавляя только операции разрешения области видимости :: так, чтобы было напечатано число 27.

```
#include <iostream>
using namespace std;
int t=0;
namespace A {  // int t=27;
    namespace B { int t=14; }
    const int t=11;
}
```

```
struct X { int t;
    struct Y{ int t; Y(){ t=27;}} y1;
    X(){t=13;}
public:
    int h(){ { int t=27;}
        return     t +     t +     t; }
};
int main(){ X b; cout<< b.h(); return 0;}
```

3. Не изменяя оператор return и не используя символов /, \, ", %, #, добавьте всё необходимое в описание класса B так, чтобы компиляция прошла без ошибок.

```
class B { public:  int f(){      return b--->a<---b;} // пробелов в выражении нет
};
```

Ответ:

4. Перечислите имена классов, описания которых содержат ошибки.

```
struct W {
  int static
  g(const int& x ){
    return 1;}
  int const g(int&x){
    return 2;}
};
```

```
struct X {

  static int
  g()const{
    return 0;}
};
```

```
srtuct Y {
  static int g() {
    return 1;}
  virtual int
  g() const {
    return 2 ;}
};
```

```
struct Z {
  int register&g(){
      return 1; }
  int const
  g(int x){
    return 2;}
};
```

Ответ:

5. Что напечатает данная программа?

```
#include <iostream>
#include <typeinfo>
using namespace std;
class C { public: virtual void f (char x='C') {cout <<x<< "1"<<endl;}};
class D : public C { public: void f (char x='D') {cout <<x<< "2"<<endl;}};
int main(){ D d; C & rc=d; rc.f(); dynamic_cast<D&>(rc).f(); return 0; }
```

Ответ:

6. Что будет напечатано в результате выполнения следующей программы?

```
#include <iostream>
using namespace std;
int f ( int n){
 try{  if (n>0) throw n; throw "b"; }
   catch(int k){cout<<k--; f(k); }
   catch(const char *s){cout<<s; throw; }
}
```

```
int main() {
 try { cout<<'a'; f(2); cout<<'c';}
      catch (int k){cout<<k;}
      catch(const char *s){cout<<s;}
    cout<<'e';
 return 0;
}
```

Ответ:

7. Объясните ошибки компиляции для данной программы. Не исправляя имеющиеся строки, вставьте новую строку кода, в которой нет символов /, \, ″,%,#,(,), так, чтобы программа скомпилировалась. **Примечание**: угловые скобки использовать можно.

```
typedef int T;

short g( short  b){ return b;}

T g( T b){ return b;}
```

```
int main() {
 return int(g('b')+g(9.8));
}
```

8. Может ли некоторый класс в языке C++ являться абстрактным классом и при этом не быть абстрактным типом данных? Если да, то приведите пример, если нет – объясните почему. *Ответ:*

9. В процессе выполнения данной программы происходит ошибка обращения по «висячему» указателю, т.е. к уже не существующему объекту. Объясните причину ошибки. Не изменяя main() и представление данных, исправьте

класс List так, чтобы этой ошибки не возникало. **Примечание:** считаем, что повторный вызов деструктора и обращение к членам объекта после его работы сами по себе не являются ошибками.

```
#include <iostream>
using namespace std;
class List { char elem; List *next;
public:
    List(): elem('\n'), next(this){}
    List(const char*s): elem(*s? *s:'\n'), next(*s? new List(s+1): this){}
    List(const List& l): elem(l.elem), next(&l!=l.next? new List(*l.next): this){}
    void print()const {cout<<elem; if(next!=this) next->print();}
   ~List(){ if(next!=this)  delete next; }
};
int main() { List c("cde");  c.~List();   c.print(); }
```

Ответ:

10. Приведите функцию, эквивалентную данной, не содержащую новых конструкций из C++11.

```
#include <vector>
int f(int y, const std::vector<int>&
  v){
  auto q=v.rbegin();
  decltype(y) s=0;
  while(q!=v.rend()) s+=*q++==y;
  return s;
}
```

Ответ:

Коллоквиум по C ++ 2017 (задачи, ответы и критерии)

За каждую задачу максимум 10 баллов. При вычитании баллов по критериям задачи полагаем y–x=0 при y < x.

Вариант 1

1. Вычеркните из функции main блоки, содержащие ошибки компиляции (предупреждения warning не считаем ошибкой). Что будет напечатано получившейся после вычеркивания программой? **Примечание**: считаем, что указатель можно без потерь преобразовать в long, а форма с многоточием поддерживает передачу любых объектов.

```
#include <iostream>
using namespace std;
struct A {
  A(int
 a=0){cout<<1;}
  A(const A&a){
      cout<<2;}
 ~A(){cout<<endl;}
};
```

```
struct B:  A {
  B(B& b){cout<<3;}
  B(const B& b){cout<<4;}
};

struct C:  B {
  C(...):B(*this){cout<<5;}
};
```

```
int main() {
  {A
 a((long)&a);}
  {A d=d=d;}
  {B b;}
  {const B b=b;}
  {C c(1); }
  {return 0;}
}
```

Ответ:

~~{B b;}~~
1
2
14
135

Область действия описываемого имени объекта начинается сразу после вхождения его описателя (описатели бывают вида p, *p, p( ) - функция, p[ ] - массив) в описание, перед инициализатором, если он есть (пункт 3.3.2 Стандарта 2011). В языках Си и C++ (в отличие от стандартного Паскаля) не считается ошибкой использование неинициализированного объекта (обращение к нему). То есть в выражении инициализации можно использовать описываемое имя объекта, например, для взятия адреса объекта.

[Обращение к полям (или методам) объекта или к его базовой части до начала выполнения конструктора или после окончания выполнения деструктора приводит (согласно пункту 12.7 Стандарта) к неопределенному поведению (undefined behavior) программы: например, в одной реализации такая программа может корректно выполниться (пункт 1.4.2 и сноска 2 в нем), в другой – приводить к ошибке компиляции или выполнения (см. 1.3.25)] [Передача потенциально-вычисляемого аргумента классового типа (9), имеющего нетривиальный конструктор копирования, нетривиальный конструктор перемещения или нетривиальный деструктор, в отсутствие соответствующего параметра, поддерживается условно; семантика такой передачи определяется реализацией (см. 5.2.2.7)]. [Компилятор Visual Studio не поддерживает форму инициализации B b (b);]

В первом блоке действует конструктор с целым параметром, напечатается 1.

При выходе из каждого блока прорабатывает деструктор базового класса, осуществляющий перевод строки.

Во втором блоке описание A d=d=d ; трактуется как A d(d=d), то есть сначала выполняется операция присваивания, сгенерированная автоматически, которая по сути ничего не делает, затем конструктор копирования, который тоже по сути ничего не копирует, но напечатает 2.

В третьем блоке ошибка, так как в классе B нет конструктора умолчания, это вычеркивается.

В четвертом блоке описание const B b=b; трактуется как const B b(b), то есть срабатывает константный конструктор копирования, который по сути ничего не копирует. Для базового класса вызывается конструктор умолчания, напечатается 1, затем конструктор производного класса напечатает число 4.

В пятом блоке для объекта c вызывается единственный конструктор «на все случаи жизни», который явно вызывает конструктор копирования (неконстантный) класса B, для которого вызывается конструктор умолчания класса A. В результате напечатается 135. (Конструктор может быть с многоточием, поскольку он является хотя и специальной, но функцией, а функция в C++ может быть описана с многоточием).

В шестом блоке лишь законный return, ошибки нет.

Критерии: -3 за каждое неверное вычеркивание (или невычеркивание) для всех блоков, кроме второго и четвертого; за каждую ошибочную напечатанную строку (в соответствии с невычеркнутыми блоками); за неверное разбиение на строки; -1 (всего) за вычеркивание второго и/или четвертого блока.

2. Измените данную программу, добавляя только операции разрешения области видимости :: так, чтобы было напечатано число 39.

```
#include <iostream>

using namespace std;
int x=0;
namespace X {  // int x=39;
    namespace Y { const int x=17; }
    const int x=9;
}
```

```
struct A { int x;
    struct B{ int x; B(){ x=39;}} x1;
    A(){x=22;}
public:
    int h(){ { int x=39;}
        return  x +   x +   x; }
};
int main(){ A a; cout<< a.h(); return 0;}
```

Ответ: нужно изменить выражение в операторе **return** `x + ::x + X::Y::x;`

Критерии: -4 за каждое неверное слагаемое (порядок слагаемых может быть любым, вместо x может быть A::x)

3. Не изменяя оператор return и не используя символов /, \, ″, %, #, добавьте всё необходимое в описание класса A так, чтобы компиляция прошла без ошибок.

```
class A { public:  int f(){      return (i,j)[i,j](i,j);}
};
```

Ответ:

```
class A { int a;
public:
   A(int a=0){this->a=a;}
   A operator[](int i){ return A(i);}
   A  operator ,(A a){return a;}
   int operator ()(A a1, A a2){ return a2.a+a1.a;}
     operator int(){return a;}
   int f(){A i,j; return (i,j)[i,j](i,j);}

};
```

Возможны вариации решения. Запрет использования / и ″ не позволяет закомментировать фрагмент или сделать его строкой, запрет # не позволяет макросы, запрет \ – эскейп-последовательности и переносы в макросах, запрет % не позволяет использовать дограф %: как альтернативу для #. [Триграфы исключены из Стандарта 2017].

Критерии: -5 за неверную трактовку выражения, -2 за каждую ошибку при описании класса

4. Перечислите имена классов, описания которых содержат ошибки.

```
struct A {
  int register f(){
      return 1;}
  int const f(int x){
      return 2;}
};
```

```
struct B {
  static int  f(){
    return 1;}
  int  f() const{
    return 2;}
};
```

```
struct C { public:
  static int
  f(const int& x ){
    return 1;}
  int  f(int& x) {
    return 2;}
};
```

```
struct D {

  static virtual
  int f() {
    return 0 ;}
};
```

Ответ: A, B, D.

В классе A ошибочно использован спецификатор register – он применяется только к данным. В классе B статическая функция не может быть перегружена с нестатической функцией с такими же параметрами. В классе C ошибок нет, нужная функция выбирается в зависимости от константности фактического параметра. В классе D ошибка в том, что статическая функция не получает скрытый параметр this на объект, от имени которого она вызвана, и поэтому не может непосредственно обращаться к нестатическим полям своего объекта (в т.ч. к скрытому полю pvtbl, содержащему адрес ТВМ) и, следовательно, не может быть виртуальной.

Критерии: -3 за каждый пропущенный или лишний класс.

5. Что напечатает данная программа?

```
#include <iostream>
#include <typeinfo>
using namespace std;
class L { public: virtual void f (char x='L') {cout <<x<< "0"<<endl;}};
class P : public L { public: void f (char x='P') {cout <<x<< "1"<<endl;}};
int main() { P p; L * pl=&p; pl->f(); dynamic_cast<P*>(pl)->f(); return 0; }
```

Ответ: L1
P1

Параметр по умолчанию для функции, вызванной через указатель или ссылку, определяется статически (в момент компиляции) по типу указателя или ссылки.

Критерии: -5 за каждую неверную строку.

6. Что будет напечатано в результате выполнения следующей программы?

```
#include <iostream>
using namespace std;
int f ( int n){
 try {  if (n<=0) throw n; f(--n); }
    catch(int&  k){cout<<k++;  throw;  throw
 "b";}
    catch(const char *s){cout<<s;}
 throw "d";
}
```

```
int main() {
  try { cout<<'a'; f(2); cout<<'c';}
     catch (int k){cout<<k;}
     catch(const char *s){cout<<s;}
  cout<<'e';
  return 0;
}
```

Ответ: a0123e (исключение перебрасывается в предыдущий рекурсивный вызов с увеличением на 1).

Критерии: -3 за каждый неверно напечатанный (или пропущенный) символ.

7. Объясните ошибки компиляции для данной программы. Не исправляя имеющиеся строки, вставьте новую строку кода, в которой нет символов /, \, ", %, #, (,), так, чтобы программа скомпилировалась.

```
typedef int T;

T p( T a){ return a;}

float p( float a){ return a;}
```

```
int main() {
 return int(p('a')+p(3.14));
}
```

Ответ: ошибка в неоднозначности для вызова p(3.14): на шаге (в) алгоритма best-matching подходят обе функции p(). Нужно вставить строку **template**<**class** T> непосредственно перед `T p( T a){ return a;}` Запрет круглых скобок не позволяет добавить перегруженную без шаблона функцию.

Критерии: -5, если неверно объяснена ошибка (не найдена, найдены лишние)
-5, если неверно вставлена строка (или не вставлена)

8. Может ли абстрактный класс в языке C++ являться абстрактным типом данных? Если да, то приведите пример, если нет – объясните почему.

Ответ: В C++ АТД реализуется с помощью классов (структур), в которых нет открытых членов-данных. Абстрактный класс содержит чистую виртуальную функцию. Следовательно, ответ положительный, вот пример абстрактного класса, являющегося АТД:

```
class A { public: virtual void f()=0;}
```

Критерии: -10 за неверный (отсутствующий) ответ, ошибочные примеры или объяснения.

9. Добавьте в main() один оператор так, чтобы в процессе выполнения программы произошла ошибка обращения по ссылке к уже не существующему объекту. Объясните причину ошибки. **Примечание:** считаем, что повторный вызов деструктора и обращение к членам объекта после его работы сами по себе не являются ошибками.

```
#include <iostream>
using namespace std;
class List { char elem; List & next;
public:
    List(): elem('\n'), next(*this){}
    List(const char*s): elem(*s? *s:'\n'), next(*s? * new List(s+1): *this){}
    List(const List& l): elem(l.elem), next(&l!=&l.next? * new List(l.next): *this){}
    void print()const {cout<<elem; if(&next!=this) next.print();}
   ~List(){ if(&next!=this) {delete &next;}  }
};
int main() { List b("abc");           b.print(); }
```

Ответ:

```
int main() { List b("abc");  b.~List(); b.print();  }
```

При явном вызове деструктора освобождается память, занимаемая звеньями списка, причем значение поля next объекта b становится неопределенным, но при следующем вызове `b.print()` делается попытка его использования. [Повторный вызов деструктора относится к неопределенному поведению (12.4.15). После окончания работы деструктора, обращение к членам объекта также относится к неопределённому поведению (12.7.1). Это означает широкий диапазон поведения в разных реализациях: в одной реализации такая программа считается корректной (что предполагалось в условии задачи), в другой может фиксироваться ошибка, так как объект «перестает существовать» с момента первого вызова деструктора.]

Критерии: -5 если не предложено верное добавление
-5 если нет правильного объяснения

10. Приведите функцию, эквивалентную данной, не содержащую новых конструкций из C++11.

```
#include <list>
int f(int x, const std::list<int>& l){
   auto p=l.begin();
   decltype(x) s=0;
   while(p!=l.end()) s+=*p++==x;
   return s;
}
```

Ответ:

```
#include <list>
int f(int x, const std::list<int>& l){
   std::list<int>::const_iterator p=l.begin();
   int s=0;
   while(p!=l.end()) s+=*p++==x;
   return s;
}
```

Критерии: - 5 за каждую неверно замененную конструкцию (auto, decltype).

Вариант 2

1. Вычеркните из функции main блоки, содержащие ошибки компиляции (предупреждения warnig не считаем ошибкой). Что будет напечатано получившейся после вычеркивания программой? **Примечание**: считаем, что указатель можно без потерь преобразовать в long, а форма с многоточием поддерживает передачу любых объектов.

```
#include <iostream>
using namespace std;
struct X {
  X(int
 x=0){cout<<1;}
  X(const X&x){
      cout<<2;}
 ~X(){cout<<endl;}
};
```

```
struct Y:  X {
  Y(){cout<<3;}
  Y(Y& y):X(*this){
     cout<<4;}
};
struct Z:  Y {
  Z(...){cout<<5;}
};
```

```
int main() {
  {X
 x((long)&x);}
  {Y c(c=c);}
  {const X b=b;}
  {X p; Z q=p;}
  {Z z; }
  {return 0;}
}
```

Ответ:

~~{X p; Z q=p;}~~
1
24
2
135

Область действия описываемого имени объекта начинается сразу после вхождения его описателя (описатели бывают вида p, *p, p( ) - функция, p[ ] - массив) в описание, перед инициализатором, если он есть (пункт 3.3.2 Стандарта 2011). В языках Си и С++ (в отличие от стандартного Паскаля) не считается ошибкой использование неинициализированного объекта (обращение к нему). То есть в выражении инициализации можно использовать описываемое имя объекта, например, для взятия адреса объекта. [Обращение к полям (или методам) объекта или к его базовой части до начала выполнения конструктора или после окончания выполнения деструктора приводит (согласно пункту 12.7 Стандарта) к неопределенному поведению (undefined behavior) программы: например, в одной реализации такая программа может корректно выполниться (пункт 1.4.2 и сноска 2 в нем), в другой – приводить к ошибке компиляции или выполнения (см. 1.3.25)].[Передача потенциально-вычисляемого аргумента классового типа (9), имеющего нетривиальный конструктор копирования, нетривиальный конструктор перемещения или нетривиальный деструктор, в отсутствие соответствующего параметра, поддерживается условно; семантика такой передачи определяется реализацией (см. 5.2.2.7)]. [Компилятор Visual Studio не поддерживает форму инициализации X b (b);]

В первом блоке действует конструктор с целым параметром, напечатается 1. Согласно пункту 5.2.10.4 Стандарта 2011, указатель может быть преобразован к целому типу, достаточному для его представления.

При выходе из каждого блока прорабатывает деструктор базового класса, осуществляющий перевод строки.

Во втором блоке сначала выполняется операция присваивания (для вычисления выражения в скобках), сгенерированная автоматически, которая по сути ничего не делает, затем конструктор копирования, который тоже по сути ничего не копирует, сначала он явно вызывает конструктор базового класса, который напечатает 2, после чего конструктор производного класса напечатает 4.

В третьем блоке описание const X b=b; трактуется как const X b(b), то есть срабатывает конструктор копирования, который по сути ничего не копирует. Напечатается 2.

В четвертом блоке ошибка, так как делается попытка инициализировать объект производного класса объектом базового класса, это вычеркивается.

В пятом блоке для объекта z вызывается единственный конструктор «на все случаи жизни», вызывающий конструктор умолчания класса B, для которого вызывается конструктор умолчания класса A. В результате напечатается 135. (Конструктор может быть с многоточием, поскольку он является хотя и специальной, но функцией, а функция в C++ может быть описана с многоточием).

В шестом блоке лишь законный return, ошибки нет.

Критерии: -3 за каждое неверное вычеркивание (или невычеркивание) для всех блоков, кроме второго и третьего; за каждую ошибочную напечатанную строку (в соответствии с невычеркнутыми блоками); за неверное разбиение на строки; -1 (всего) за вычеркивание второго и/или третьего блока.

2. Измените данную программу, добавляя только операции разрешения области видимости :: так, чтобы было напечатано число 27.

```
#include <iostream>

using namespace std;
int t=0;
namespace A {  // int t=27;
    namespace B { const int t=14; }
    const int t=11;
}
```

```
struct X { int t;
    struct Y{ int t; Y(){ t=27;}} y1;
    X(){t=13;}
public:
    int h(){ { int t=27;}
        return  t +   t +    t; }
};
int main(){ X b; cout<< b.h(); return 0;}
```

Ответ: нужно изменить выражение в операторе `return  t + ::t + A::B::x;`

Критерии: -4 за каждое неверное слагаемое (порядок слагаемых может быть любым, вместо t может быть X::t)

3. Не изменяя оператор return и не используя символов /, \, ″, %, #, добавьте всё необходимое в описание класса B так, чтобы компиляция прошла без ошибок.

```
class B { public:  int f(){      return b--->a<---b;} // пробелов в выражении нет
};
```

Ответ: Если *b* описать как объект класса *B*, содержащего поле *a,* то выражение можно трактовать следующим образом: `b.operator--(int).operator->() < b.operator-(). operator--().operator int()`. Остается определить в классе необходимые операции и, возможно, конструкторы.

```
class B { int a;
public:
   B(int a=0){this->a=a;}
   B & operator--(){a--; return *this;}
   B operator--(int){B temp(*this);a--; return temp;}
   B * operator->(){return this;}
   B operator -(){B temp(-a); return temp;}
     operator int(){return a;}
   int f(){B b;  return b--->a<---b;}
};
```

Возможны вариации решения. Запрет использования / и ″ не позволяет закомментировать фрагмент или сделать его строкой, запрет # не позволяет макросы, запрет \ – эскейп-последовательности и переносы в макросах, запрет % не позволяет использовать дограф %: как альтернативу для #. [Триграфы исключены из Стандарта 2017].

Критерии: -5 за неверную трактовку выражения, -2 за каждую ошибку при описании класса.

4. Перечислите имена классов, описания которых содержат ошибки.

```
struct W {
  int static
  g(const int& x ){
    return 1;}
  int const g(int&x){
    return 2;}
};
```

```
struct X {

  static int
  g()const{
    return 0;}
};
```

```
struct Y {
  static int g() {
    return 1;}
  virtual int
  g() const {
    return 2 ;}
};
```

```
struct Z {
  int register&g(){
      return 1; }
  int const
  g(int x){
    return 2;}
};
```

Ответ: X, Y, Z.

В классе W ошибок нет, нужная функция выбирается в зависимости от константности фактического параметра. В классе X ошибка в том, что статическая функция по определению не пользуется нестатическими полями (в т.ч. воображаемым полем this) и, следовательно, не может быть константной. В классе Y статическая функция не может быть перегружена с нестатической функцией с такими же параметрами. В классе Z ошибочно использован спецификатор register – он применяется только к данным в блоке или параметрам функции (7.1.1.2). [В Стандарте 2017 http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf слово register всё ещё остаётся зарезервированным, но уже не означает ничего]

Критерии: -3 за каждый пропущенный или лишний класс.

5. Что напечатает данная программа?

```
#include <iostream>
#include <typeinfo>
using namespace std;
class C { public: virtual void f (char x='C') {cout <<x<< "1"<<endl;}};
class D : public C { public: void f (char x='D') {cout <<x<< "2"<<endl;}};
int main(){ D d; C & rc=d; rc.f(); dynamic_cast<D&>(rc).f(); return 0; }
```

Ответ: C2
D2

Параметр по умолчанию для функции, вызванной через указатель или ссылку, определяется статически (в момент компиляции) по типу указателя или ссылки.

Критерии: -5 за каждую неверную строку.

6. Что будет напечатано в результате выполнения следующей программы?

```
#include <iostream>
using namespace std;
int f ( int n){
 try{  if (n>0) throw n; throw "b"; }
   catch(int k){cout<<k--; f(k); }
   catch(const char *s){cout<<s; throw; }
}
```

```
int main() {
 try { cout<<'a'; f(2); cout<<'c';}
      catch (int k){cout<<k;}
      catch(const char *s){cout<<s;}
    cout<<'e';
 return 0;
}
```

Ответ: a21bbe (функция рекурсивно вызывается из обработчика).

Критерии: -3 за каждый неверно напечатанный (или пропущенный) символ.

7. Объясните ошибки компиляции для данной программы. Не исправляя имеющиеся строки, вставьте новую строку кода, в которой нет символов /, \, ″, %, #, (,), так, чтобы программа скомпилировалась.

```
typedef int T;

short g( short  b){ return b;}

T g( T b){ return b;}
```

```
int main() {
 return int(g('b')+g(9.8));
}
```

Ответ: ошибка в неоднозначности для вызова g(9.8): на шаге (в) алгоритма best-matching подходят обе функции g(). Нужно вставить строку **template**<**class** T> непосредственно перед `T g( T b){ return b;}`. Запрет круглых скобок не позволяет добавить перегруженную без шаблона функцию.

Критерии: -5, если неверно объяснена ошибка (не найдена, найдены лишние).
-5, если неверно вставлена строка (или не вставлена).

8. Может ли некоторый класс в языке C++ являться абстрактным классом и при этом **не** быть абстрактным типом данных? Если да, то приведите пример, если нет – объясните почему.

Ответ: В C++ АТД реализуется с помощью классов (структур), в которых нет открытых членов-данных. Абстрактный класс содержит чистую виртуальную функцию. Следовательно, ответ положительный, вот пример абстрактного класса, не являющегося АТД:

```
class A { public: int a; virtual void f()=0;}
```

Критерии: -10 за неверный (отсутствующий) ответ, ошибочные примеры или объяснения.

9. В процессе выполнения данной программы происходит ошибка обращения по «висячему» указателю, т.е. к уже не существующему объекту. Объясните причину ошибки. Не изменяя main() и представление данных, исправьте класс List так, чтобы этой ошибки не возникало. **Примечание:** считаем, что повторный вызов деструктора и обращение к членам объекта после его работы сами по себе не являются ошибками.

```
#include <iostream>
using namespace std;
class List { char elem; List *next;
public:
    List(): elem('\n'), next(this){}
    List(const char*s): elem(*s? *s:'\n'), next(*s? new List(s+1): this){}
    List(const List& l): elem(l.elem), next(&l!=l.next? new List(*l.next): this){}
    void print()const {cout<<elem; if(next!=this) next->print();}
   ~List(){ if(next!=this)  delete next; }
};
int main() { List c("cde");  c.~List();   c.print(); }
```

Ответ: При явном вызове деструктора освобождается память, занимаемая звеньями списка, причем значение поля next объекта c становится неопределенным, но при следующем вызове `c.print()` делается попытка его использования.

Достаточно исправить деструктор следующим образом:

```
~List(){ if(next!=this) { delete next; next=this; elem='\n';}    }
```

[Повторный вызов деструктора относится к неопределенному поведению (12.4.15). После окончания работы деструктора, обращение к членам объекта также относится к неопределённому поведению (12.7.1). Это означает широкий диапазон поведения в разных реализациях: в одной реализации такая программа считается корректной (что предполагалось в условии задачи), в другой может фиксироваться ошибка, так как объект «перестает существовать» с момента первого вызова деструктора.]

Критерии: - 5 если ошибка не найдена или не объяснена правильно,
- 5 если не предложено верное исправление.

10. Приведите функцию, эквивалентную данной, не содержащую новых конструкций из C++11.

```
#include <vector>
int f(int y, const std::vector<int>& v){
  auto q=v.rbegin();
  decltype(y) s=0;
  while(q!=v.rend()) s+=*q++==y;
  return s;
}
```

Ответ:

```
#include <vector>
int f(int y, const std::vector<int>& v){
   std::vector<int>::const_reverse_iterator q=v.rbegin();
   int s=0;
   while(q!=v.rend()) s+=*q++==y;
   return s;
}
```

Критерии: - 5 за каждую неверно замененную конструкцию (auto, decltype).

2017 год
Экзамен

Вариант 1_2017 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1. Описать **класс A** и **одну** функцию *sum (x,y),* вычисляющую сумму своих двух аргументов, так, чтобы в приведенной функции main () не было ошибок.

```
int main (){
        A a, b(5);
        return   sum (a,b) + sum <int>(17,18);
}
```

2. Что напечатает данная программа?
Укажите прототипы конструкторов и деструкторов классов A и B в порядке их выполнения.

```
struct A {
        virtual void f () { cout << "A_f\n"; }
        virtual void g () { cout << "A_g\n"; }
};
struct B : A {
        void f () { cout << "B_f\n"; }
};
```

```
int main () {
try {   A a;  B b;  throw b;
        }
        catch ( A & a) { a.f();  a.g(); }
        catch(...){cout << "catch(...)\n";}
        return 0;
}
```

3. Ничего **не удаляя и не исправляя структуру D и функцию *main()***, **добавить** к нижеприведенному фрагменту **необходимые описания** так, чтобы в получившейся программе не было ошибок, а на экран выдалось **6**.

```
struct D : B {
        int func1 (int a) { return a; }
};
int main () {
        D d;
        cout << d.func1 (B::x) + d.func2 (2) + d.B::func1 (1) << endl;
        return 0;
}
```

4. Если при компиляции следующей программы будут обнаружены ошибки, найдите их и **объясните причины**, иначе укажите, **что** будет **выдано на печать**.

```
char fff (char c) { return c+1; }
double fff (double d) { return d+2; }
namespace N {
        int fff (int i) { return i+3; }
        float fff (float f) { return f+4; }
}
int main () {
        cout <<  fff (3) << fff (5.0f) << endl;
        return 0;
}
```

5. STL. Что такое итераторы? Какие категории итераторов реализованы в STL? Чем разные категории итераторов отличаются друг от друга?

6. **a)** В чем **различие** статических и динамических библиотек?

b) Какие компоненты вычислительной системы и **на каком этапе** работы занимаются подключением к программам каждого из этих видов библиотек?

7. **a)** Эквивалентны ли следующие грамматики? **Ответ обосновать**.

b) Каков **тип** каждой из заданных **грамматик?**

c) Каков **тип языка**, порождаемого каждой грамматикой?

G1: $S \to S1 \mid S2 \mid 1 \mid 2$

G2: $S \to 1A \mid 2A$
$A \to 1A \mid 2A \mid \varepsilon$

G3: $S \to ABS \mid 1 \mid 2$
$AB \to BA$
$A \to 1A \mid \varepsilon$
$B \to 2B \mid \varepsilon$

8. **a) Вставить** в одну из грамматик предыдущего задания **№ 7** **действия** вида

⟨ *cout << "символы";* ⟩

так, чтобы во время анализа **методом рекурсивного спуска** цепочек ω исходного языка L (порождаемого грамматикой) был реализован перевод

$$\tau = \{(\omega, 1^k 2^m) \mid \omega \in L,\ k = |\omega|_1,\ m = |\omega|_2\}$$

b) Возможно ли вставить подобные действия в другую грамматику из задания № 7? Ответ **обосновать**.

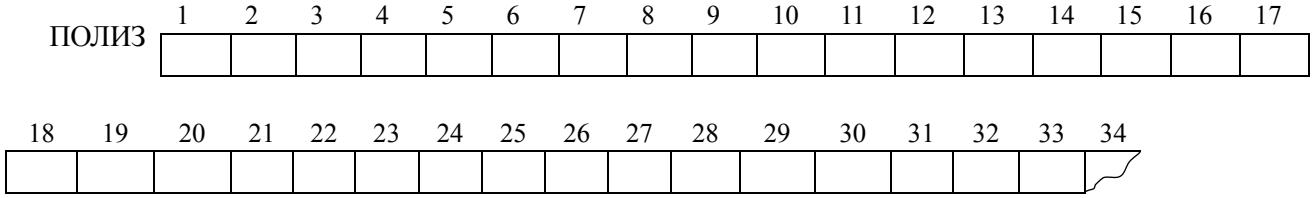
9. **a)** Для заданной грамматики построить эквивалентную неукорачивающую грамматику, пользуясь **алгоритмом удаления пустых правых частей в КС-грамматике**.

b) Построить **конечный автомат** по получившейся грамматике. Если получившийся автомат недетерминирован, преобразовать его **к ДКА**, иначе построить по нему **анализатор**.

$S \to Aa \mid Bb$
$A \to Bb \mid b \mid \varepsilon$
$B \to Aa \mid \varepsilon$

10. Записать на ПОЛИЗе фрагмент программы на Си:

```
a  =  x < --y + 7  ?  -x  * (a = b = 5)  :  b - a - 1;
```



Вариант 2_2017 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std.`**

1	2	3	4	5	6	7	8	9	10

1. Описать **класс B** и **одну** функцию *neg (x),* меняющую знак своего аргумента на противоположный, так, чтобы в приведенной функции main () не было ошибок.

```
int main (){
        B a, b(7);
        return   neg (a) * neg<B>(b) * neg<int>(-5);
}
```

2. Что напечатает данная программа? **Укажите прототипы** конструкторов и деструкторов классов B и D в порядке их выполнения.

```
struct B {
        virtual void f () { cout << "B_f\n"; }
        virtual void g () { cout << "B_g\n"; }
};
struct D : B {
        void f () { cout << "D_f\n"; }
};
```

```
int main () {
try { D d;  throw d;
        }
        catch ( B  a) { a.f();  a.g(); }
        catch(...){cout << "catch(...)\n";}
return 0;
}
```

3. Ничего **не удаляя и не исправляя структуру B и функцию *main()***, **добавить** к нижеприведенному фрагменту **необходимые описания** так, чтобы в получившейся программе не было ошибок, а на экран выдалось **10**.

```
struct B : A {
        int g2 (int a) { return a; }
};
int main () {
        B b;
        cout << b.g1 (6) + b.g2 (4) + b.A::g1 (2) + A::x << endl;
        return 0;
}
```

4. Если при компиляции следующей программы будут обнаружены ошибки, найдите их и **объясните причины**, иначе укажите, **что** будет **выдано на печать**.

```
int f2 (int i) { return i+1; }
long double f2 (long double ld) { return ld+2; }
namespace M {
        char f2 (char c) { return c+3; }
```

```
        double f2 (double d) { return d+4; }
    }
    int main () {
        cout <<  f2 ('3') << f2 (5.0) << endl;
        return 0;
}
```

5. Какие типы стандартных исключений автоматически генерируются компилятором при динамической идентификации типа (механизм **RTTI**)? **Объясните**, в каких ситуациях они генерируются.

6. **a) Какие** стратегии трансляции могут выбираться в различных системах программирования?

b) В чем **отличия** этих подходов к реализации процесса трансляции?

c) Приведите по **одному примеру** формального языка, к которому применяется каждая из названных вами стратегий.

7. **a)** Эквивалентны ли следующие грамматики? **Ответ обосновать**.

b) Каков **тип** каждой из заданных **грамматик?**

c) Каков **тип языка**, порождаемого каждой грамматикой?

G1: $S \to bA$, $A \to aC$, $B \to aBb \mid \varepsilon$, $C \to Bba \mid \varepsilon$

G2: $S \to bAa$, $A \to aAb \mid \varepsilon$

G3: $S \to bAa$, $bA \to bC$, $C \to aCb \mid \varepsilon$

8. **a) Вставить** в одну из грамматик предыдущего задания **№ 7** **действия** вида

⟨ *cout << "символы";* ⟩

так, чтобы во время анализа **методом рекурсивного спуска** цепочек ω исходного языка L (порождаемого грамматикой) был реализован перевод

$$\tau = \{(\omega, b^k a^m) \mid \omega \in L,\ k = |\omega|_b,\ m = |\omega|_a\}$$

b) Возможно ли вставить подобные действия в другую грамматику из задания № 7? Ответ **обосновать**.

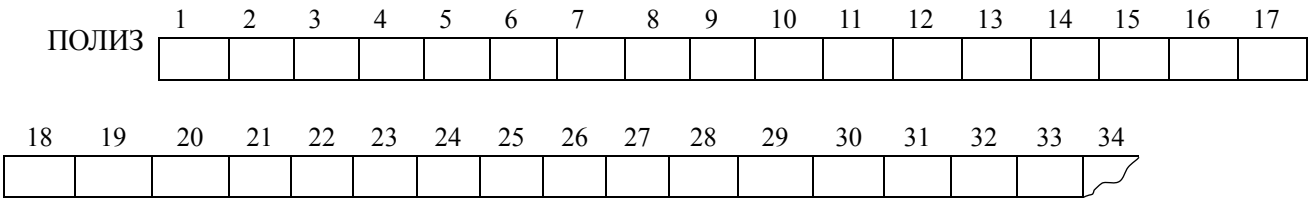
9. **a)** Для заданной грамматики построить эквивалентную неукорачивающую грамматику, пользуясь **алгоритмом удаления пустых правых частей в КС-грамматике**.

b) Построить **конечный автомат** по получившейся грамматике. Если получившийся автомат недетерминирован, преобразовать его **к ДКА**, иначе построить по нему **анализатор**.

$S \to 0A \mid 1B$
$A \to 0A \mid 0B$
$B \to 1A \mid 1B \mid \varepsilon$

10. Записать на ПОЛИЗе фрагмент программы на Си:

```
do  {x = z = (x + y + 5) * b - 3/f; b++; } while (-x + y >0)
```



Ответы - 2 вариант 2017
Каждая задача оценивается максимум в 10 баллов

1.
```
class B {                    Возможны варианты решения !!!
        int x;
public:
        B (int a = 0) { x = a; }
        int operator - () { return  -x; }
};
template < class T >
int neg (const T & t) {return  -t; }
```

Критерии: нет\неверное описание шаблона функции - **-5**
неверное описание класса (нет нужных конструкторов, не перегружен ун. -) - **-5**
если параметры в neg передаются по ссылке, то за неконстантную ссылку - **-2**
если решение не требует использования операций (например, в классе реализована в функция преобразования объекта в int), то за отсутствие перегрузки операций не снижаем.

2.
B_f
B_g
B(), D(), B(**const** B&), D(**const** D&), ~D(), ~B(), B(**const** B&), ~B(), ~D(), ~B()

Критерии: неверная печать - **-5**
каждое нарушение порядка, лишний\отсутствующий конструктор\деструктор
(за парный к отсутствующему конструктору деструктор не наказывать) - **-3**

3.
```
struct A {                          Возможны варианты решения !!!
        static int  x;
        int g1 (int a) { return a; }
        int g2 (int a) { return a; }
};
int A::x = -2;
```

Критерии: нет **static int** x; - **-5**
отсутствие описания x вне класса, нет необходимых методов в A,
другие ошибки - **-3** за каждую.

4. Ошибка одна - неоднозначный выбор перегруженной функции в вызове f2(5.0) - обнаруживается на шаге 2- в) работы соответствующего алгоритма.

Критерии: **нет объяснения** ошибки, ошибка не найдена - **0**
наряду с правильным ответом найдены другие ошибки - **-5** за каждую.

5. ***bad_cast*** - при невозможности приведения типов полиморфных классов с помощью операции *dynamic_cast*.

bad_typeid - при попытке раскрытия нулевого указателя в аргументе операции *typeid*.

Критерии: нет\неверное объяснение ***bad_cast*** - **-5**
нет\неверное объяснение ***bad_typeid*** - **-5**

Ответы - Грамматики - В-4 (2) - 2017

6. Стратегии трансляции:
- **компиляция** (вся программа с ЯП перводится в машинные коды - Паскаль, Си, ...),
- **интерпретация** (в машинные коды переводится конструкция за конструкцией ЯП и сразу выполняется - любой скриптовый язык,...),
- **смешанная стратегия** (транслятор переводит программу на ЯП в программу на промежуточном языке, которая затем интерпретируется, JavaScript ... в принципе почти любой интерпретируемый язык переводится в некоторое внутреннее представление).

Критерии: пропуск каждой стратегии - **-4 ,**
неверное объяснение сути каждой стратегии **- -3 ,**
нет\неверный пример - **-2** за каждый.

7. Да, эквивалентны, поскольку порождают один и тот же язык: $L = \{b\ a^n\ b^n\ a \mid n >= 0\}$
G1 - тип 2, КС; G2 - тип 2, КС; G3 - тип 0.
Тип языка - тип 2, КС.

Критерии: неверно ответ на 1 вопрос - **-5 ,**
неверное указание типа грамматики - **-3** за каждый ,
неверное указание типа языка - **-3** за каждый.

8. G1: $S \rightarrow b<cout << 'b';>A$
$A \rightarrow aC <cout << 'a';>$
$B \rightarrow a<cout << 'b';> B <cout << 'a';>b \mid \varepsilon$

$C \rightarrow <cout << 'b';> B <cout << 'a';>\ ba \mid \varepsilon$

Нет, нельзя: G3 - не КС, РС-метод неприменим,
G2 - РС-метод неприменим: $first(A) \cap follow(A) = \{a\}$

Критерии: неверно выбрана грамматика и\или вставлены действия - **-5 ,**
неверное обоснование выбора **- -5**.

9. Неукорачивающая грамматика:
$S \rightarrow 0A \mid 1B \mid 1$
$A \rightarrow 0A \mid 0B \mid 0$
$B \rightarrow 1B \mid 1A \mid 1$

ДКА:
$q(S, 0) = A$
$q(S, 1) = \underline{BF}$
$q(A, 0) = \underline{ABF}$
$q(\underline{BF}, 1) = \underline{ABF}$
$q(\underline{ABF}, 0) = \underline{ABF}$
$q(\underline{ABF}, 1) = \underline{ABF}$

Критерии: неверно построена неукорачивающая грамматика - **-5 ,**
нет\неверно построен ДКА **- -5**.

10. do {x = z = (x + y + 5) * b - 3/f; b++; } while (-x + y >0)

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
&x	&z	x	y	+	5	+	b	*	3	f	/	-	=	=	;	&b

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
#+	;	x	@	y	+	0	>	30	!F	1	!					

Критерии: пропуск & (адреса), каждой операции - **-2 ,** пропуск ';' - **-1** ;
каждая неверная позиция операции - **-4**.

2018 год
Коллоквиум

Коллок_1_2018 Ф.И.О_____№ группы_____

1	2	3

1. (50 баллов) **Найти и обязательно объяснить** все ошибки, обраруженные при компиляции приведенной программы на C++ (с элементами новых стандартов). Если в какой-то конструкции программы несколько ошибок, укажите и объясните их все.

Исправить найденные ошибки, не меняя текст ниже 19-й строки и внося только необходимые для компиляции программы исправления. **Нельзя** закрытые члены-данные заданных классов делать открытыми. Правильные конструкции заменять, комментировать или удалять **нельзя**.

Что будет выдано на печать при работе получившейся правильной программы?

```
1. #include <iostream>
2. #include <typeinfo>
3. using namespace std;
4.  class A {
5.     int x, y;
6.     A(const A& b) {}
7.  public:
8.     A (int a = 0, int b = 0) { x = a; y = b;}
9.     int g() noexcept { return 555;}
10. };
11.
12. class B : public A {
13.    int z;
14.    B (const B& b) {}
15. public:
16.    int w;
17.    B (int b) { z = b; }
18. };
19.
20. B& operator- (B& b) {
21.    b.z = -b.z;
22.    return b;
23. }
24.
25. int x = 1;
26.
27. void f(float y) {
28.    cout << "::f \n";
29. }
```

```
30. template <class T> void f(T t) noexcept {
31.    cout << "f<> \n";
32. }
33.
34. namespace nnn {
35.    int x = 2;
36.    void f(int c) {
37.           cout << "nnn::f \n";
38.    }
39. }
40. using namespace nnn;
41.
42. int main () {
43.    try {
44.        B b1, b2(6), *pb;
45.        const B b;
46.        A a1, a2(3), *pa = &b2, *pa1 = &a1;
47.        pb = dynamic_cast <B*>(pa1);
48.        if (pb) cout << typeid (*pb).name() << endl;
49.        f(1);
50.        cout << ::x << endl;
51.        cout << b.w++ << endl << b.g() << endl;
52.        throw *pa;
53.    }
54.    catch( B&) { cout << "error BBB " << endl; }
55.    catch( A&) { cout << "error AAA " << endl; }
56.    catch( ... ) { cout << "error ... " << endl; }
57.    return 0;
58. }
```

2. (40 баллов) **Добавить необходимые описания** в заданную программу, никоим образом **не изменяя и не удаляя приведенный текст,** так, чтобы в получившейся программе не было ошибок, а на экран выдалось:

f()_B
5
3 3
0 0

```
#include <iostream>
using namespace std;

struct A {
       int a;
       A (int n):a(n){}
       virtual void f() {cout << "f()_A\n";}
       virtual int g()=0;
};

struct T {
       int a,b;
       T(int x = 0, int y = 0) {
              a = x; b = y;
       }
};

int main () {
       B b;
       A* pa = &b;
       pa -> f();
       cout << pa->a << endl;
       T t;
       P p(&t);
       p->a = p->b = 3;
       cout << t.a << ' ' << t.b << endl;
```

```
        P p1(NULL);
        cout << p1->a << ' ' << p1->b << endl;
        return 0;
}
```

3. (10 баллов) **STL**. а) Что такое алгоритм в STL?
б) Каким образом алгоритмы STL получают доступ к тому или иному контейнеру?
в) Какое принято соглашение с целью определения того, к каким контейнерам применим тот или иной алгоритм STL?

Ответы_Коллок 1_2018

1. Максимум 50 баллов.
Ошибки:
1) с. 21 - обращение к закрытому члену класса (сделать эту функцию другом B)

2) с. 44 - при создании объекта b1 - нет конструктора умолчания (например, внести в конструктор класса B значение параметра по умолчанию).

3) с. 47 - операция dynamic_cast применена к указателю неполиморфного класса (любым способом сделать класс B полиморфным).

4) с. 51 **-** а) попытка изменить поле константного объекта (сделать его статическим: добавить static перед int w; и вне класса описание статического поля: int B::w = 0; с некоторой инициализацией для дальнейшей печати)

5) с. 51 **-** б) применение неконстантного метода к константному объекту (сделать метод g константным: const перед noexcept, если метод g() виртуальный, статическим он быть не может)

6) с. 52 - при генерации объекта исключения вызывается конструктор копирования класса A, насположенный в **private**-области (перенести его в **public**-область или перегрузить в **public** без **const**, конструктор крпирования класса B никак не задействован, его трогать не нужно).

На печать будет выдано:
nnn::f
1
0 (или другое инициализирующее значение)
555
error AAA

Критерии: Ошибка не найдена или неправильно объяснена, лишняя ошибка – **-5 за каждую**.
Неверное исправление - **-5 за каждое;**
Нет описания статического поля вне класса **- -2;**
Неверная печать **– 5 за каждый неверный элемент печати.**

2. Максимум 40 баллов.//Возможны варианты решения!!!

```
struct B : A {
      B(): A(5) {}
      virtual void f() { cout << "f()_B\n";}
      virtual int g() { return 555;}
};

class  P {
      T* pt;
      bool del;
public:
      P(T* t){
            del = t ? false : true;
            pt = t ? t: new T();
      }
```

```
T* operator -> () { return pt; }
~P() { if (del) delete pt; }
      };
```

Критерии: Нет отношения наследрвания между A и B **- -20.**
Неверная\отсутствующая операция **-> –10.**
Нет списка инициализации в конструкторе B(), нет конструктора - **-10.**
Не переопределена функция g() - **-10**.
Не освобождается используемая динамическая память - **-2**.
Неверная печать **– 5 за каждый неверный элемент печати.**

3. Максимум 10 баллов.

а) Алгоритм **STL** - шаблонная функция, выполняющая некоторое действие над некоторым контейнером.
б) Доступ к контейнеру осуществляется посредством итераторов контейнера, передаваемых в качестве параметров функции-шаблона.
в) Алгоритм применим к контейнерам, тип итераторов которых "не ниже" обозначенного именем формального параметра шаблона.

Критерии: Неверный ответ на пункт а**) – -4**.
Неверный ответ на пункт б) **– -3.**
Неверный ответ на пункт в) **– -3.**

Коллок_2_2018 Ф.И.О._____№ группы_____

1	2	3

1. (50 баллов) **Найти и обязательно объяснить** все ошибки, обраруженные при компиляции приведенной программы на C++ (с элементами новых стандартов). Если в какой-то конструкции программы несколько ошибок укажите и объясните их все.

Исправить найденные ошибки, не меняя текст ниже 18-й строки и внося только необходимые для компиляции программы исправления. **Нельзя** закрытые члены-данные заданных классов делать открытыми. Правильные конструкции заменять, комментировать или удалять **нельзя**.

Что будет выдано на печать при работе получившейся правильной программы?

```
1.  #include <iostream>
2.  #include <typeinfo>
3.  using namespace std;
4.  class B {
5.     int x;
6.  public:
7.     B (int a = 0) { x = a;}
8.     B (B & b) {x = b.x;}
9.     int g() { return 222;}
10. };
11.
12. class D : public B {
13.    int z;
14. public:
15.    int y;
16.    D (int b) { z = b; }
17. };
18.
19. B operator+ (B& b1, B& b2) {
20.    return b1.x + b2.x;
21. }
22.
23. int x = 1;
24.
25. void f (float y) {
26.    cout << "::f(float) \n";
27. }
28. void f (char y) {
```

```
29.    cout << "::f(char) \n";
30. }
31. template <class T> void f (T t){
32.    cout << "f<> \n";
33. }
34.
35. namespace kkk {
36.    int x = 2;
37.    void f (int c) {
38.        cout << c << "kkk::f(int) \n";
39.    }
40. }
41. using namespace kkk;
42.
43. int main () {
44.    try {
45.        const D d;
46.        D && rrr = D();
47.        const B & rb = d;
48.        const D & rd = dynamic_cast <const D &>(rb);
49.        f(::x);
50.        f(10.5);
51.        cout << --d.y << endl << d.g() << endl;
52.        throw rrr;
53.    }
54.    catch( B&) { cout << "error BBB " << endl; }
55.    catch( ... ) { cout << "error ... " << endl; }
56.    return 0;
57. }
```

2. (40 баллов) **Добавить необходимые описания** в заданную программу, никоим образом **не изменяя и не удаляя приведенный текст,** так, чтобы в получившейся программе не было ошибок, а на экран выдалось:

f()_D
f()_B
28
1

```
#include <iostream>
using namespace std;

struct B {
       int b;
       B (int x){ b = x;}
       virtual void func () {cout << "f()_B\n";}
       virtual void h(double)=0;
};

struct T {
       int a;
       T(int x = 1) {a = x;}
};

int main () {
       D d;
       C  c;
       B *bp = &d, *bp1 = &c;
       bp -> func();
       bp1-> func();
       T t;
       P p(&t);
       (*p).a = 28;
       cout << t.a << endl;
```

```
        P p1(NULL);
        cout << (*p1).a << endl;
        return 0;
}
```

3. (10 баллов) **STL**. а) Что такое итератор STL?
б) Перечислите категории итераторов STL .

Ответы_Коллок 2_2018

1. Максимум 50 баллов. Ошибки:

1) с. 20 - обращение к закрытому члену класса (сделать эту функцию другом B)

2) с. 20 - при формировании возвращаемого по значению объекта-результата вызывается конструктор копирования. Ему по неконстантной ссылке передаётся временный объект, что запрещено (сделать параметр конструктора копирования класса B константным).

3) с. 45 - при создании объекта d - нет конструктора умолчания (например, внести в конструктор класса D значение параметра по умолчанию).

4) с. 48 - операция dynamic_cast применена к ссылке неполиморфного класса (любым способом сделать класс B полиморфным).

5) с. 51 - а) попытка изменить поле константного объекта (сделать его статическим: добавить **static** перед **int** y; и вне класса описание статического поля: **int** D::y = 0 (какая-то инициализация для дальнейшей печати)

6) с. 51 - б) применение неконстантного метода к константному объекту (сделать метод g константным: если метод g() сделан виртуальным, статическим он быть не может)

На печать будет выдано: 1 kkk::f (**int**)
f <>
-1 (если инициализирующее значение 0, или, соответственно, что-то другое)
222
error BBB

Критерии: Ошибка не найдена или неправильно объяснена, лишняя ошибка – **-5 за каждую**.
Неверное исправление - **-5 за каждое;**
Нет описания статического поля вне класса **- -2;**
Неверная печать **– 5 за каждый неверный элемент печати.**

4. //Возможны варианты решения!!! Максимум 40 баллов.

```
struct C : B {
      C(): B(12) { }
      void h (double d) { cout << d << endl;}
};
struct D : C {
      void func () { cout << "f()_D\n";}
};
class  P {
      T* pt;
      bool bb;
public:
      P(T* t){
```

```
            bb = t ? true: false;
            pt = bb ? t: new T();
        }
        T& operator* () { return *pt; }
        ~P(){ if (!bb) delete pt; }
    };
```

Критерии: Нет отношения наследования между B, C и\или D - **-20**
Неверная\отсутствующая операция * - **–10.**
Нет списка инициализации в конструкторе C(), нет конструктора C() - **-10.**
Не переопределена функция h(double) - **-10**.
В методе **operator*** результат возвращается НЕ по ссылке - **-5**.
Не освобождается используемая динамическая память - **-2**.
Неверная печать **– 5 за каждый неверный элемент печати.**

3.

а) Итератор **STL** - шаблонный класс, объекты которого выполняют роль указателя по отношению к некоторым контейнерам.

б) Выделяют 5 категорий итераторов:

1. **Вывода** (output) (запись в контейнер - *p = , ++)
2. **Ввода** (input) (считывание из контейнера - = *p, →, ++, ==, !=)
3. **Однонаправленный** (forward) (считывание и запись в одном направлении - *p =, →, =*p, ++ , ==, !=)
4. **Двунаправленный** (bidirectional) (*p=, =*p, →, ++,--, ==, !=) - list, map, set
5. **С произвольным доступом** (random_access) (все 4. плюс [], +, -, +=, -=, <, >, <=, >=) - vector, deque

в) Типы итераторы различаются набором применимых к ним операций.

Критерии: Неверный ответ на пункт а**)** – **-4**.
Неверный ответ на пункт б) – **-1 за каждый пропущенный тип итераторов.**
Неверный ответ на пункт в) – **-3.**

2018 год
Экзамен

Вариант 1_2019 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Ответьте на следующие вопросы. Обоснуйте ответ на каждый вопрос.

(а) Может ли грамматика одновременно быть регулярной и не быть контекстно-зависимой?

(б) Может ли конечный язык быть неоднозначным?

(в) Существует ли регулярный язык, который не порождается неукорачивающей грамматикой?

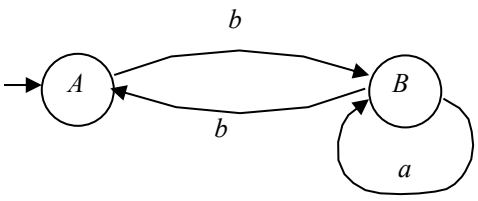
2. Постройте грамматику выражений над переменными a и b с операциями сложения и умножения и унарным минусом (наивысший приоритет), к которой применим метод рекурсивного спуска. Добавьте в грамматику действия только вида ⟨ *cout* <<'СИМВОЛ'; ⟩, реализующие формальный перевод $\tau = \{ (x, y) \mid x$ – арифметическое выражение, y – эквивалентное арифметическое выражение, не содержащее двух рядом стоящих унарных минусов $\}$. Пример: $-(a + - - b * - - - (a))$ ➔ $- (a + b * - (a))$.

3. Даны названия программных систем: *CVS, SVN, GIT, Make*. Вычеркнуть среди них одно лишнее и дать определение понятию, примерами которого являются все оставшиеся системы.

Ответ:

4. Даны: 1) часть множества P_L леволинейной грамматики $G_L = \langle \{A, B, C\}, \{a, b\}, P_L, C \rangle$; 2) часть множества P_R праволинейной грамматики $G_R = \langle \{A, B, C\}, \{a, b\}, P_R, A \rangle$; 3) часть множества дуг δ конечного автомата $A = \langle \{A, B, C\}, \{a, b\}, \delta, \{A\}, \{C\} \rangle$.

δ:



P_L: $A \to Aa \mid \varepsilon$

...

P_R: $C \to aC \mid \varepsilon$

...

Пополнить δ, P_L, P_R, (дописать недостающие правила, дорисовать дуги) так, чтобы одновременно выполнялись условия: (а) грамматики G_L и G_R приведённые и эквивалентные; (б) A получается из G_L алгоритмом построения конечного автомата по леволинейной грамматике; (в) A получается из G_R алгоритмом построения конечного автомата по праволинейной грамматике; (г) A не является детерминированным; (д) количество дуг в A не превосходит 6.

5. Дана польская запись фрагмента программы на Си. Восстановите этот фрагмент на языке Си без операторов перехода. Приведите еще один (отличающийся, но также без операторов перехода) текстовый фрагмент, имеющий такую же польскую запись. **Примечание:** пустой оператор не занимает место в польской записи.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ПОЛИЗ	&b	+#	7	!F	7	!	&a	–#	;	a	b	+	17	!F	1	!	

Ответ:

6. Если в функции main есть блоки, содержащие ошибки компиляции, вычеркните их. Что будет напечатано получившейся программой? Считать, что компилятор ничего не оптимизирует.

```
#include <iostream>
using namespace std;
struct A {
  A(int a){cout<<1;}
  A(const    A&a    =
 A(1)){
      cout<<2;}
  A(A& a){cout<<3;}
 ~A(){cout<<'a';}
};
```

```
struct B:  A {

 B(const      B&      b){
 cout<<4;}

 B(int i=0):A(*this){

 cout<<5;}
 ~B(){cout<<'b';}
};
```

```
int main(){
 {cout<< "start";}
 {cout<<endl; A a(1);}
 {cout<<endl; A a;}
 {cout<<endl; B b;}
  return 0;
}
```

Ответ:

7. (а) Привести описания не более двух функций, чтобы были верны следующие обращения:

```
g(50L);  g((const double) 1); g ();   g ("str"); g (0, 0);    g (3, "str");
```

(б) Может ли константный член класса быть статическим ? Обоснуйте ответ.

8. Может ли в выражении вида $a+b$ проявиться одновременно и статический, и динамический полиморфизм? Обоснуйте ответ. Какой еще вид полиморфизма есть в C++?

9. Что будет напечатано в результате работы данной программы? В курсе рассматривались стандартные исключения bad_alloc, bad_cast, bad_typeid.

```
#include <iostream>
#include <exception>
#include <typeinfo>
using namespace std;
class List { char elem; List * next;
             int ballast[30000000];
public:
List(const char*s): elem(*s? *s:'\n'),
          next(*s?     new   List(s+1):
 NULL){}

List & operator*= (List & l) {
    List *q=&l, *p=this;
    while (p->next!=NULL) p=p->next;
    while (q->next!=NULL){
        p->elem = q->elem;
        p->next = new List("");
        q=q->next; p=p->next;
    }
    return *this;
}
~List(){ if(next!=NULL) {delete next;}
}
};

int main() {
    List w("xyz");
    try{
        try { throw '5'; throw 5;
        }
        catch (int) {cout<< "int ";}
        catch (char){cout<< "char ";
            w*=w; throw;
        }
     }
     catch   (char)   {cout<<   "char"<<
endl;}
     catch(exception & e){
         cout<<typeid(e).name() <<endl;
     }
     return 0;
}
```

Ответ:

10. Опишите шаблонную функцию *Reverse* (*a*), которая по контейнеру *a* (вектор, список, очередь и т.п.) строит в качестве значения новый контейнер, в котором элементы *a* расположены в обратном порядке. Например, для списка $\langle 1, 2, 3\rangle$ функция построит список $\langle 3, 2, 1\rangle$.

Вариант 1_3_2018 **Задачи. Ответы и решения, критерии**

За каждую задачу максимум 10 баллов. При вычитании баллов по критериям задачи полагаем 0–x=0.

1. Ответьте на следующие вопросы. Обоснуйте ответ на каждый вопрос.

(а) Может ли грамматика одновременно быть регулярной и не быть контекстно-зависимой?

Ответ: Да, может. Пример: $S\to Sa \mid \varepsilon$.

(б) Может ли конечный язык быть неоднозначным?

Ответ: Нет, не может. Для конечного языка с n цепочками можно построить грамматику с n альтернативами для начального символа, где в правых частях – цепочки языка. Каждая цепочка в такой грамматике выводится за один шаг единственным образом.

(в) Существует ли регулярный язык, который не порождается неукорачивающей грамматикой?

Ответ: Нет, не существует. Любую регулярную грамматику можно преобразовать в эквивалентную регулярную неукорачивающую с помощью алгоритма устранения правил с пустой правой частью.

Критерии: за каждый пункт: если неверный ответ: -4,
если ответ верный, но не обоснован: -3.

2. Постройте грамматику выражений над переменными a и b с операциями сложения и умножения и унарным минусом (наивысший приоритет), к которой применим метод рекурсивного спуска. Добавьте в грамматику действия только вида ⟨ *cout* <<'символ'; ⟩, реализующие формальный перевод $\tau = \{ (x, y) \mid x$ – арифметическое выражение, y – эквивалентное арифметическое выражение, не содержащее двух рядом стоящих унарных минусов $\}$. Пример: $-(a + - - b * - - - (a))$ ➔ $-(a + b * -(a))$.

Ответ: запись действия ⟨ *cout* <<'символ'; ⟩ будем сокращать до ⟨символ ⟩. Фигурные скобки означают итерацию, как в БНФ. Грамматика может быть и без них, с использованием только рекурсии.

$S \to A \ \{+ \langle + \rangle \ A\}$

$A \to B \ \{* \ \langle * \rangle \ B\}$

$B \to N F$

$N \to - \ M \mid \varepsilon$

$M \to - \ N \mid \varepsilon \ \langle - \rangle$

$F \to a \langle a \rangle \mid \ b \langle b \rangle \mid (\langle (\rangle \ S \ ) \langle) \rangle$

Критерии: построенная грамматика описывает не все выражения или есть лишние цепочки: -5
метод рекурсивного спуска не применим к построенной грамматике: -3
за каждую неверную или отсутствующую печать в грамматике с действиями: -2

Замечание. За формально правильную схему перевода даже при наличии дефектов в исходной грамматике не снижаем, так как за дефекты грамматики (например, рекурсивный спуск неприменим) мы уже снизили.

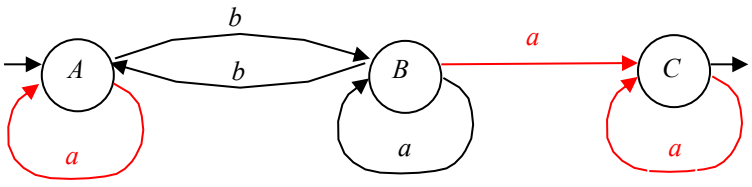
3. Даны названия программных систем: *CVS, SVN, GIT, Make*. Вычеркнуть среди них одно лишнее и дать определение понятию, примерами которого являются все оставшиеся системы.

Ответ: Лишнее – *Make*. *Система контроля версий* – в самом общем понимании осуществляет отслеживание версий (ревизий) некоторого набора объектов (например, для файлов можно при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение и т.п.).

Критерии: за неправильное вычеркивание: -5 . За ошибочное или отсутствующее определение: -5
Определение может быть дано другими словами.

4. Даны: 1) часть множества P_L леволинейной грамматики $G_L = \langle \{X, Y, Z\}, \{a, b\}, P_L, Z \rangle$; 2) часть множества P_R праволинейной грамматики $G_R = \langle \{X, Y, Z\}, \{a, b\}, P_R, X \rangle$; 3) часть множества дуг δ конечного автомата $A = \langle \{X, Y, Z\}, \{a, b\}, \delta, \{X\}, \{Z\} \rangle$.

δ:



P_L: $C \to Ca \mid Ba$
$B \to Ba \mid Ab$
$A \to Aa \mid \varepsilon \mid Bb$

P_R: $A \to aA \mid bB$
$B \to aB \mid aC \mid bA$

$C \to aC \mid \varepsilon$

Пополнить δ, P_L, P_R, (дописать недостающие правила, дорисовать дуги) так, чтобы одновременно выполнялись условия: (а) грамматики G_L и G_R приведённые и эквивалентные; (б) A получается из G_L алгоритмом построения конечного автомата по леволинейной грамматике; (в) A получается из G_R алгоритмом построения конечного автомата по праволинейной грамматике; (г) A не является детерминированным; (д) количество дуг в A не превосходит 6.

Ответ: Элементы, добавленные в δ, P_L, P_R, на рисунке показано красным. Есть другой вариант ответа, где из B в C идет дуга, помеченная символом b – такой автомат тоже будет недетерминированным.

Критерии: есть ошибка (лишняя дуга или нет нужной) в пополнении δ : -4
есть ошибка в пополнении P_L,: -3
есть ошибка в пополнении P_R: -3

5. Дана польская запись фрагмента программы на Си. Восстановите этот фрагмент на языке Си без операторов перехода. Приведите еще один (отличающийся, но также без операторов перехода) текстовый фрагмент, имеющий такую же польскую запись. **Примечание:** пустой оператор не занимает место в польской записи.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&b	+#	7	!F	7	!	&a	–#	;	a	b	+	17	!F	1	!	

Ответ: ***do*** *{if(++b);* ***else****; --a;}* ***while****(a+b);*
Чтобы получить еще один пример, можно добавить круглые скобки вокруг арифметического подвыражения или пустой оператор изобразить как {}, или непустой оператор заключить в блок.

Критерии: неверно восстановлен фрагмент: -5
не приведен еще один эквивалентный фрагмент: -5

6. Если в функции main есть блоки, содержащие ошибки компиляции, вычеркните их. Что будет напечатано получившейся программой? Считать, что компилятор ничего не оптимизирует.

```
#include <iostream>
using namespace std;
struct A {
  A(int a){cout<<1;}
  A(const    A&a    =
 A(1)){
      cout<<2;}
  A(A& a){cout<<3;}
 ~A(){cout<<'a';}
};
```

```
struct B:  A {

 B(const      B&      b){
 cout<<4;}

 B(int i=0):A(*this){

 cout<<5;}
 ~B(){cout<<'b';}
};
```

```
int main(){
 {cout<< "start";}
 {cout<<endl; A a(1);}
 {cout<<endl; A a;}
 {cout<<endl; B b;}
  return 0;
}
```

Ответ:
start
1a
12aa
35ba

Критерии: ошибок в программе нет, за каждую ошибочную строку (из четырех) -3.

7. (а) Привести описания не более двух функций, чтобы были верны следующие обращения:

```
g(50L);  g((const double) 1); g ();   g ("str"); g (0, 0);    g (3, "str");
```

(б) Может ли константный член класса быть статическим ? Обоснуйте ответ.

Ответ: (а)
```
void g (double n = 0, const char * t = 0){}
void g (const char * t){}
```
Возможны другие решения. Вместо `const char *` допускается `char *` .

(б) Может. Пример: `struct A{ static const int zero;}; const int A::zero=0;`

Замечание. Функция-член класса не может быть одновременно константной и статической, так как const относится к “скрытому параметру” this, а статическая функция ”не получает” параметр this.

Критерии: (а) за каждую ошибочную, недостающую или лишнюю функцию -4; (б) за неверный или не обоснованный ответ -3.

8. Может ли в выражении вида *a+b* проявиться одновременно и статический, и динамический полиморфизм? Обоснуйте ответ. Какой еще вид полиморфизма есть в C++?

Ответ: еще есть параметрический полиморфизм.

Одновременно проявиться может. Опишем базовый класс с двумя перегруженными виртуальными операциями + (например, одна с параметром типа *int*, другая – типа *char*. Имя *a* опишем как ссылку на базовый класс, инициализировав ее объектом производного класса, *b* опишем как *int*. Тогда в выражении *a+b* статически выберется операция + с параметром *int*, и динамически через ТВМ вызовется метод + для производного класса.

В качестве обоснования может быть просто описание классов и имен *a* и *b*, без подробных словесных пояснений.

Критерии: за не названный, или не существующий вид полиморфизма : -2
за правильный, но не обоснованный ответ на первый вопрос: -7
за неправильный ответ на первый вопрос: -8

9. Что будет напечатано в результате работы данной программы? В курсе рассматривались стандартные исключения bad_alloc, bad_cast, bad_typeid.

```
#include <iostream>
#include <exception>
#include <typeinfo>
using namespace std;
class List { char elem; List * next;
             int ballast[30000000];
public:
List(const char*s): elem(*s? *s:'\n'),
          next(*s?      new   List(s+1):
 NULL){}

List & operator*= (List & l) {
    List *q=&l, *p=this;
    while (p->next!=NULL) p=p->next;
    while (q->next!=NULL){
        p->elem = q->elem;
        p->next = new List("");
        q=q->next; p=p->next;
    }
    return *this;
}
~List(){ if(next!=NULL) {delete next;}
}
};

int main() {
    List w("xyz");
    try{
        try { throw '5'; throw 5;
        }
        catch (int) {cout<< "int ";}
        catch (char){cout<< "char ";
            w*=w; throw;
        }
     }
     catch   (char)   {cout<<   "char"<<
endl;}
     catch(exception & e){
         cout<<typeid(e).name() <<endl;
     }
return 0;
}
```

Ответ: char bad_alloc
Критерии: за каждую неверную печать -5

10. Опишите шаблонную функцию *Reverse* (*a*), которая по контейнеру *a* (вектор, список, очередь и т.п.) строит в качестве значения новый контейнер, в котором элементы *a* расположены в обратном порядке. Например, для списка ⟨1, 2, 3⟩ функция построит список ⟨3, 2, 1⟩.
Ответ:

```
template <class T>
T Reverse (const T& a){
    T res;
    typename T::const_reverse_iterator it=a.rbegin();
    while (it!=a.rend()) res.push_back(*it++);
```

```
    return res;
}
```

Замечание. Метод push_front( ) для векторов не определен.

Критерии: за неконстантный итератор (если передача параметра по ссылке) : -2
за другие ошибки: -3 за каждую

Вариант 4_2018 Ф.И.О._____№ группы_____

1	2	3	4	5	6	7	8	9	10

1. Ответьте на следующие вопросы. Обоснуйте ответ на каждый вопрос.

(а) Может ли грамматика одновременно быть контекстно-свободной и не быть контекстно-зависимой?

(б) Может ли регулярный язык быть неоднозначным?

(в) Существует ли контекстно-свободный язык, который не порождается неукорачивающей грамматикой?

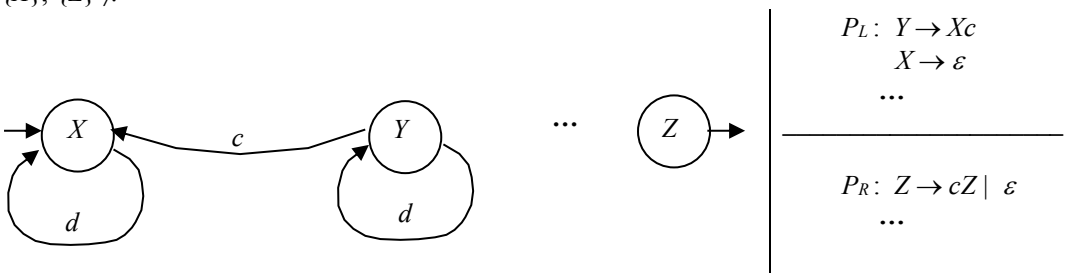
2. Постройте грамматику логических выражений над переменными *a* и *b*, к которой применим метод рекурсивного спуска. Добавьте в грамматику действия только вида ⟨ *cout* <<'СИМВОЛ'; ⟩, реализующие формальный перевод $\tau = \{ (x, y) \mid x$ – логическое выражение, y – эквивалентное логическое выражение, не содержащее двух рядом стоящих операций *not* $\}$.

Пример: ***not*** (*a* ***or not not*** *b* ***and not not not*** (*a*)) ➔ ***not*** (*a* ***or*** *b* ***and not*** (*a*)).

3. Даны названия программных средств: *CORBA, COM, Java Beans*, *Valgrind*. Вычеркнуть среди них одно лишнее и дать определение понятию, примерами которого являются все оставшиеся средства.
Ответ:

4. Даны: 1) часть множества P_L леволинейной грамматики $G_L = \langle \{X, Y, Z\}, \{c, d\}, P_L, Z \rangle$; 2) часть множества P_R праволинейной грамматики $G_R = \langle \{X, Y, Z\}, \{c, d\}, P_R, X \rangle$; 3) часть множества дуг δ конечного автомата $A = \langle \{X, Y, Z\}, \{c, d\}, \delta, \{X\}, \{Z\} \rangle$.

δ:



Пополнить δ, P_L, P_R, (дописать недостающие правила, дорисовать дуги) так, чтобы одновременно выполнялись условия: (а) грамматики G_L и G_R приведённые и эквивалентные; (б) A получается из G_L алгоритмом построения конечного автомата по леволинейной грамматике; (в) A получается из G_R алгоритмом построения конечного автомата по праволинейной грамматике; (г) A не является детерминированным; (д) количество дуг в A не превосходит

5. Дана польская запись фрагмента программы на Си. Восстановите этот фрагмент на языке Си (без операторов перехода). Приведите еще один (отличающийся, но также без операторов перехода) текстовый фрагмент, имеющий такую же польскую запись. **Примечание:** пустой оператор не занимает место в польской записи.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ПОЛИЗ	a	b	+	17	!F	&b	#–	12	!F	12	!	&a	#+	;	1	!	

Ответ:

6. Если в функции main есть блоки, содержащие ошибки компиляции, вычеркните их. Что будет напечатано получившейся программой? Считать, что компилятор ничего не оптимизирует.

```
#include <iostream>
using namespace std;
struct C {
  C(int c){cout<<1;}
  C(const    C&c    =
 C(1)){
      cout<<2;}
  C(C& c){cout<<3;}
~C(){cout<<'c';}
};
```

```
struct D:  C {

  D(const              D&
 d){cout<<4;}
  D(int i=0){
             cout<<5;}
  ~D(){cout<<'d';}
};
```

```
int main(){
 {cout<< "start";}
 {cout<<endl;         C
 c=C(1);}
 {cout<<endl; C c(1);}
 {cout<<endl; D d;}
  return 0;
}
```

Ответ:

7. (а) Привести описания не более двух функций, чтобы были верны следующие обращения:

```
h("str");   h (); h ((const int)2.0); h(0, 0);h ("b", 10);  h (10, "0.0");
```

(б) Может ли статический член класса быть константным ? Обоснуйте ответ.

8. Может ли в выражении вида *a*[*b*] проявиться одновременно и статический, и динамический полиморфизм? Обоснуйте ответ. Какой еще вид полиморфизма есть в C++?

9. Что будет напечатано в результате работы данной программы? В курсе рассматривались стандартные исключения bad_alloc, bad_cast, bad_typeid.

Ответ:

```
#include <iostream>
#include <exception>
#include <typeinfo>
using namespace std;
class List { char elem; List * next;
             int ballast[20000000];
public:
List(const char*s): elem(*s? *s:'\n'),
          next(*s?      new   List(s+1):
 NULL){}

List & operator+= (List & l) {
    List *p=&l, *q=this;
    while (q->next!=NULL) q=q->next;
    while (p->next!=NULL){
        q->elem=p->elem;
        q->next = new List("");
        p=p->next; q=q->next;
    }
    return *this;
}
```

```
~List(){  if(next!=NULL)  {delete  next;}
}
};

int main() {
    List b("abc");
    try{
        try { throw 5; throw '5';
        }
        catch (char) {cout<< "char ";}
        catch (int){cout<< "int ";
            b+=b; throw;
        }
     }
     catch (int) {cout<< "int"<< endl;}
     catch(exception & e){
         cout<<typeid(e).name() <<endl;
     }
     return 0;
}
```

10. Опишите шаблонную функцию *Concat* (*a*, *b*), которая по двум однотипным контейнерам *a* и *b* (векторы, списки, очереди и т.п.) строит в качестве значения новый контейнер, содержащий конкатенацию *a* и *b*. Например, для списков ⟨1, 2, 3⟩ и ⟨4, 5⟩ конкатенацией будет ⟨1, 2, 3, 4, 5⟩.

Вариант 2_4_2018 **Задачи. Ответы и решения, критерии**

За каждую задачу максимум 10 баллов. При вычитании баллов по критериям задачи полагаем 0–x=0.

1. Ответьте на следующие вопросы. Обоснуйте ответ на каждый вопрос.

(а) Может ли грамматика одновременно быть контекстно-свободной и не быть контекстно-зависимой?
Ответ: Да, может. Пример: $S \to bSa \mid \varepsilon$.

(б) Может ли регулярный язык быть неоднозначным?
Ответ: Нет, не может. Регулярный язык всегда можно задать детерминированным конечным автоматом, и построить по нему праволинейную грамматику, например. Она будет однозначной вследствие детерминированности автомата.

(в) Существует ли контекстно-свободный язык, который не порождается неукорачивающей грамматикой?
Ответ: Нет, не существует. Любую КС-грамматику можно преобразовать в эквивалентную неукорачивающую КС-грамматику с помощью алгоритма устранения правил с пустой правой частью.

Критерии: за каждый пункт: если неверный ответ: -4,
если ответ верный, но не обоснован: -3.

2. Постройте грамматику логических выражений над переменными *a* и *b*, к которой применим метод рекурсивного спуска. Добавьте в грамматику действия только вида ⟨ *cout* <<"символы"; ⟩, реализующие формальный перевод τ = { (*x*, *y*) | *x* – логическое выражение, *y* – эквивалентное логическое выражение, не содержащее двух рядом стоящих операций *not* }.
Пример: ***not*** (*a* ***or not not*** *b* ***and not not not*** (*a*)) ➔ ***not*** (*a* ***or*** *b* ***and not*** (*a*)).

Ответ: запись действия ⟨ *cout*<< "символы"; ⟩ будем сокращать до ⟨символы⟩. Фигурные скобки означают итерацию, как в БНФ. Грамматика может быть и без них, с использованием только рекурсии.

$S \to A \ \{ \ \mathbf{or} \ \langle \ \mathbf{or} \ \rangle \ A\}$
$A \to B \ \{ \ \mathbf{and} \ \langle \ \mathbf{and} \ \rangle \ B\}$
$B \to N \ F$
$N \to \mathbf{not} \ \ M \mid \ \varepsilon \ \langle \ \mathbf{not} \ \rangle$
$M \to \mathbf{not} \ \ N \mid \ \varepsilon \ \langle \ \mathbf{not} \ \rangle$
$F \to a \ \langle a \rangle \mid \ b \ \langle b \rangle \mid \mathbf{(} \ \langle (\ \rangle \ S \ \mathbf{)} \ \langle \ ) \rangle$

Критерии: построенная грамматика описывает не все выражения или есть лишние цепочки: -5
метод рекурсивного спуска не применим к построенной грамматике: -3
за каждую неверную или отсутствующую печать в грамматике с действиями: -2

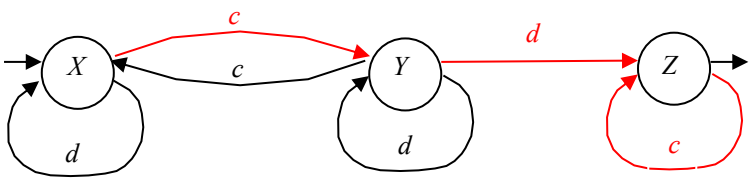
Замечание. За формально правильную схему перевода даже при наличии дефектов в исходной грамматике не снижаем, так как за дефекты грамматики (например, рекурсивный спуск неприменим) мы уже снизили.

3. Даны названия программных средств: *CORBA, COM, Java Beans*, *Valgrind*. Вычеркнуть среди них одно лишнее и дать определение понятию, примерами которого являются все оставшиеся средства.
Ответ: Лишнее – *Valgrind*. *Библиотека компонентов* – это библиотека готовых откомпилированных программных модулей, предназначенных для использования в качестве составных частей программ, и которыми можно манипулировать во время разработки программ.

Критерии: за неправильное вычеркивание: -5 . За ошибочное или отсутствующее определение: -5
Определение может быть дано другими словами.

4. Даны: 1) часть множества P_L леволинейной грамматики $G_L = \langle \{X, Y, Z\}, \{c, d\}, P_L, Z \rangle$; 2) часть множества P_R праволинейной грамматики $G_R = \langle \{X, Y, Z\}, \{c, d\}, P_R, X \rangle$; 3) часть множества дуг δ конечного автомата $A = \langle \{X, Y, Z\}, \{c, d\}, \delta, \{X\}, \{Z\} \rangle$.

δ:



P_L: $Z \to Zc \mid Yd$
$Y \to Xc \mid Yd$
$X \to \varepsilon \mid Xd \mid Yc$

P_R: $X \to dX \mid cY$
$Y \to dZ \mid dY \mid cX$
$Z \to cZ \mid \varepsilon$

Пополнить δ, P_L, P_R, (дописать недостающие правила, дорисовать дуги) так, чтобы одновременно выполнялись условия: (а) грамматики G_L и G_R приведённые и эквивалентные; (б) A получается из G_L алгоритмом построения конечного автомата по леволинейной грамматике; (в) A получается из G_R алгоритмом построения конечного автомата по праволинейной грамматике; (г) A не является детерминированным; (д) количество дуг в A не превосходит 6.

Ответ: Элементы, добавленные в δ, P_L, P_R, на рисунке показаны красным. Есть другой вариант ответа, где из Y в Z идет дуга, помеченная символом c – такой автомат тоже будет недетерминированным.

Критерии: есть ошибка (лишняя дуга или нет нужной) в пополнении δ : -4
есть ошибка в пополнении P_L,: -3
есть ошибка в пополнении P_R: -3

5. Дана польская запись фрагмента программы на Си. Восстановите этот фрагмент на языке Си (без операторов перехода). Приведите еще один (отличающийся, но также без операторов перехода) текстовый фрагмент, имеющий такую же польскую запись. **Примечание:** пустой оператор не занимает место в польской записи.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ПОЛИЗ	a	b	+	17	!F	b	#–	12	!F	12	!	&a	#+	;	1	!	

Ответ: ***while*** *(a+b){* ***if*** *(b--);* ***else****; a++;}*
Чтобы получить еще один пример, можно добавить круглые скобки вокруг арифметического подвыражения или пустой оператор изобразить как {}, или непустой оператор заключить в блок.

Критерии: неверно восстановлен фрагмент: -5
не приведен еще один эквивалентный фрагмент: -5

6. Если в функции main есть блоки, содержащие ошибки компиляции, вычеркните их. Что будет напечатано получившейся программой? Считать, что компилятор ничего не оптимизирует.

```
#include <iostream>
using namespace std;
struct C {
  C(int c){cout<<1;}
  C(const     C&c     =
 C(1)){
      cout<<2;}
  C(C& c){cout<<3;}
 ~C(){cout<<'c';}
};
```

```
struct D:   C {

  D(const                D&
 d){cout<<4;}
  D(int i=0){
              cout<<5;}
  ~D(){cout<<'d';}
};
```

```
int main(){
 {cout<< "start";}
 {cout<<endl;          C
 c=C(1);}
 {cout<<endl; C c(1);}
 {cout<<endl; D d;}
  return 0;
}
```

Ответ:
start
12cc
1c
12c5dc

Критерии: ошибок в программе нет, за каждую ошибочную строку (из четырех) -3.

7. (а) Привести описания не более двух функций, чтобы были верны следующие обращения:

```
h("str");   h (); h ((const int)2.0); h(0, 0);h ("b", 10);   h (10, "0.0");
```

(б) Может ли статический член класса быть константным ? Обоснуйте ответ.

Ответ: (а)
```
void h (double r = 0, const char * s = 0) {}
void h (const char *s, int n = 0) {}
```

Возможны другие решения. Вместо `const char *` допускается `char *` .

(б) Может. Пример: `struct A{ static const int zero;}; const int A::zero=0;`

Замечание. Функция-член класса не может быть одновременно константной и статической, так как const относится к “скрытому параметру” this, а статическая функция ”не получает” параметр this.

Критерии: (а) за каждую ошибочную, недостающую или лишнюю функцию -4; (б) за неверный или не обоснованный ответ -3.

8. Может ли в выражении вида *a*[*b*] проявиться одновременно и статический, и динамический полиморфизм? Обоснуйте ответ. Какой еще вид полиморфизма есть в C++?

Ответ: еще есть параметрический полиморфизм.

Одновременно проявиться может. Рассмотрим описания классов:

```
class A{
    public:
        virtual void operator[](int){cout<<"A[int]";}
        virtual void operator[](char){cout<<"A[char]";}
};

class B: public A{
        public:
        virtual void operator[](int){cout<<"B[int]";}
        virtual void operator[](char){cout<<"B[char]";}
};
```

Имя *a* опишем как ссылку на базовый класс *A*, инициализировав ее объектом производного класса, *b* опишем как *int*. Тогда в выражении *a*[*b*] статически выберется операция [] с параметром *int*, и динамически через ТВМ вызовется метод [] для производного класса. Напечатается *B*[*int*].

Обоснование может быть и словесным, без конкретного примера класса.

Критерии: за не названный, или не существующий вид полиморфизма : -2
за правильный, но не обоснованный ответ на первый вопрос: -7
за неправильный ответ на первый вопрос: -8

9. Что будет напечатано в результате работы данной программы? В курсе рассматривались стандартные исключения bad_alloc, bad_cast, bad_typeid.

```
#include <iostream>
#include <exception>
#include <typeinfo>
using namespace std;
class List { char elem; List * next;
             int ballast[20000000];
public:
List(const char*s): elem(*s? *s:'\n'),
          next(*s?      new   List(s+1):
 NULL){}

List & operator+= (List & l) {
    List *p=&l, *q=this;
    while (q->next!=NULL) q=q->next;
    while (p->next!=NULL){
        q->elem=p->elem;
        q->next = new List("");
        p=p->next; q=q->next;
    }
    return *this;
}
~List(){ if(next!=NULL) {delete next;}
}
};

int main() {
    List b("abc");
    try{
        try { throw 5; throw 'c';
        }
        catch (char) {cout<< "char ";}
        catch (int){cout<< "int ";
            b+=b; throw;
        }
     }
     catch (int) {cout<< "int"<< endl;}
     catch(exception & e){
         cout<<typeid(e).name() <<endl;
     }
    return 0;
}
```

Ответ: int bad_alloc
Критерии: за каждую неверную печать -5

10. Опишите шаблонную функцию *Concat* (*a*, *b*), которая по двум однотипным контейнерам *a* и *b* (векторы, списки, очереди и т.п.) строит в качестве значения новый контейнер, содержащий конкатенацию *a* и *b*. Например, для списков ⟨1, 2, 3⟩ и ⟨4, 5⟩ конкатенацией будет ⟨1, 2, 3, 4, 5⟩.

Ответ:

```
template <class T>
T Concat (const T & a, const T & b){
    T res;
    typename T::const_iterator it=a.begin();
    while (it!=a.end()) res.push_back(*it++);
    it=b.begin();
    while (it!=b.end()) res.push_back(*it++);
    return res;
}
```

Возможны другие решения, например, без циклов через insert():

```
res.insert(res.end(), a.begin(), a.end());
res.insert(res.end(), b.begin(), b.end());
```

Критерии: за неконстантный итератор (если используется передача параметров по ссылке): -2
за другие ошибки: -3 за каждую

2019 год
Коллоквиум

Коллоквиум_2019_1 ФИО_____№ группы_____

1. Дана заготовка класса *Scanner*. Метод *analyze*() данного класса осуществляет разбор по некоторой детерминированной леволинейной грамматике G_{left} с нетерминальным алфавитом {*A, B*} и множеством правил {$A \to Aa \mid Ba \mid \varepsilon$; $B \to Bb \mid Ab$}. Полагаем, что задаваемая во входном потоке *cin* цепочка для анализа всегда завершается символом '#',

который не входит в алфавит терминальных символов грамматики G_{left}. Если анализатор попадает в состояние ошибки *ERR*, то он прекращает работу, оставаясь в этом состоянии.

```
#include<iostream>
#include<typeinfo>
using namespace std;

class Scanner {
  class State {

    State * operator ()(char c) const =
;



  };
  class A: public State { public:
    State * operator ()(char c)const{
      if (c=='b') return new B;
      else if (c=='a') return new A;
      else return new ERR;
    }
  };
  class  B:             {




  };
  class ERR: public State  {
    State * operator ()(char c)const{
      return new ERR;}
     // из состояния ошибки нет перехо-
дов
     // в другие состояния
  };
```

```
  char c; // текущий символ
  State * temp, *next; // текущее и
    //следующее состояния автомата (ДС)
public:
  void analyze(){ State::st_count=0;
    for (temp = new A, cin>>c;
         c!='#'&&
         !( dynamic_cast<ERR *> (temp));
         cin>>c
         ){
            next=(*temp)(c);
            delete temp;
            temp=next;
    }
    if ( c!='#' ||
         typeid(*temp)!=typeid(B)
        )
          cout<< "ERROR" <<endl;
    else
          cout<< "OK, длина пути в \
                  диаграмме состояний ="
              << State::st_count <<
endl;
    delete temp;
  }
};

int Scanner::State::st_count;
```

а) Определите начальный символ грамматики G_{left} по описанию класса *Scanner*. (Обратите внимание прежде всего на метод *analyze()*). *Ответ:*

б) Определите язык, порождаемый грамматикой G_{left} (формула или словесное описание).
Ответ:

в) Постройте диаграмму состояний для G_{left} – пусть это будет конечный автомат $\mathcal{Y}$.
Ответ:

г) Постройте эквивалентный автомат $\mathcal{Y}'$ путем добавления в $\mathcal{Y}$ только одной дуги так, чтобы $\mathcal{Y}'$ получился бы недетерминированным. (Новые состояния не добавлять.)
Ответ:

д) Добавьте недостающие фрагменты описаний в классы *State* и *B* (и только в них!) так, чтобы: - метод *analyze*() считал правильными только все цепочки языка *L* (G_{left}),
- никаким способом невозможно было бы создать отдельный объект класса *State,*
- в случае успеха *analyze*() печатал бы количество вершин на пути по диаграмме состояний (ДС для G_{left}) из начальной вершины в заключительную.

2. Дана заготовка класса *Parser.* Метод *analyze*() класса *Parser* осуществляет разбор по некоторой контекстно-свободной грамматике $G = \langle \{a, b\}, \{A, B, S\}, P, S \rangle$, используя принцип рекурсивного спуска. Полагаем, что задаваемая во входном потоке *cin* цепочка для анализа всегда завершается символом *'#'*.

```
#include<iostream>
#include<list>
using namespace std;

class Parser {
  static list<char> result; // результат
             //перевода исходной цепочки
  static char c;//текущий символ(лек-
сема)

  static void gc(){cin>>c;} // получить
              //следующий символ-лексему

  class S {
  public:
    S(){ if (c=='a') {gc ();  A();
            if (c=='b') {gc(); B();
            }
            else throw c;
          }
          else throw c;
    }
    ~S(){ result.push_front('S');}
  };
  class A {
  public:
    A(){if (c=='a') {gc ();  A();}
    }
    ~A(){ result.push_front('A');}
  };
  class B {
  public:
    B(){ if (c=='b') gc ();
         else throw c;
    }
    ~B(){ result.push_front('B');}
  };

public:
  void analyze(){
    try{ result.clear();
         gc();
         S();
         if (c!='#') throw c;
         cout<< "Результат перевода для
\
                 входной цепочки: ";




         cout<<endl;
    }
    catch(       ){

    }
  }
}; //end of Parser

char Parser::c;
list<char> Parser::result;
```

а) Восстановите по анализатору множество правил *P* грамматики *G*.
Ответ:

б) Докажите, что метод рекурсивного спуска применим для *G*.
Ответ:

в) Однозначна ли грамматика *G*? Обоснуйте ответ.
Ответ:

г) Постройте эквивалентную неукорачивающую грамматику алгоритмом устранения эпсилон-правил. *Ответ*:

д) Добавьте в метод *analyze*() (и только в него!) все необходимое так, чтобы в результате работы анализатора
1) для **корректных** входных цепочек на экран печатался результат перевода входной цепочки в цепочку в алфавите {*S, A, B*}, в которой нетерминальные символы грамматики располагаются в порядке их замены при построении **правостороннего** вывода,
2) в случае **ошибки** печаталось сообщение:
« Ошибочный символ : < *здесь печатать символ, ставший причиной ошибки*> ».

Коллоквиум_2019_2 ФИО_____№ группы_____

1. Дана заготовка класса *Parser*. Метод *analyze*() класса *Parser* осуществляет разбор по некоторой контекстно-свободной грамматике $G = \langle \{b, d\}, \{B, D, S\}, P, S \rangle$, используя принцип рекурсивного спуска. Полагаем, что задаваемая во входном потоке *cin* цепочка для анализа всегда завершается символом *'#'*.

```
#include<iostream>
#include<list>
using namespace std;
class Parser {
  static list<char> trans; // результат
            //перевода исходной цепочки
  static char c;//текущий символ(лек-
сема)

  static void gc(){cin>>c;} // получить
             //следующий символ-лексему

  struct S {
    S(){ if (c=='b') {gc ();  B(); D();
            if (c=='d') {gc();
            }
            else throw c;
         }
         else if (c=='d') gc();
              else throw c;
    }
    ~S(){ trans.push_front('S');}
  };
  struct D {
    D(){ if (c=='d') gc ();
         else throw c;
    }
    ~D(){ trans.push_front('D');}
  };
  struct B {
    B(){if (c=='b') {gc ();  B();}
    }
    ~B(){ trans.push_front('B');}
  };

public:
  void analyze(){
    try{ trans.clear();
         gc();
         S();
         if (c!='#') throw c;
         cout<< "Результат перевода для\
                 входной цепочки: ";



         cout<<endl;
    }
    catch(       ){

    }
  }
}; //end of Parser

char Parser::c;

list<char> Parser::trans;
```

а) Восстановите по анализатору множество правил P грамматики G.
Ответ:

б) Докажите, что метод рекурсивного спуска применим для G.
Ответ:

в) Однозначна ли грамматика G? Обоснуйте ответ.
Ответ:

г) Постройте эквивалентную неукорачивающую грамматику алгоритмом устранения эпсилон-правил. *Ответ*:

д) Добавьте в метод *analyze*() (и только в него!) все необходимое так, чтобы в результате работы анализатора
1) для **корректных** входных цепочек на экран печатался результат перевода входной цепочки в цепочку в алфавите $\{S, B, D\}$, в которой нетерминальные символы грамматики располагаются в порядке их замены при построении **правостороннего** вывода,

2) в случае **ошибки** печаталось сообщение:

« Ошибочный символ : < *здесь печатать символ, ставший причиной ошибки*> ».

2. Дана заготовка класса *Scanner*. Метод *analyze*() данного класса осуществляет разбор по некоторой детерминированной леволинейной грамматике G_{left} с нетерминальным алфавитом {*C, D*} и множеством правил { $D \rightarrow Da \mid Ca$; $C \rightarrow Cb \mid Db \mid \varepsilon$ }. Полагаем, что задаваемая во входном потоке *cin* цепочка для анализа всегда завершается символом '#', который не входит в алфавит терминальных символов грамматики G_{left}. Если анализатор попадает в состояние ошибки *ERR*, то он прекращает работу, оставаясь в этом состоянии.

```
#include<iostream>
#include<typeinfo>
using namespace std;

class Scanner {
  class State {

    State * operator ()(char c) const =
;



  };
  class C: public State { public:
    State * operator ()(char c)const{
      if (c=='b') return new C;
      else if (c=='a') return new D;
           else return new ERR;
    }
  };
  class  D:             {




  };
  class ERR: public State  {
    State * operator ()(char c)const{
      return new ERR;}
     // из состояния ошибки нет перехо-
дов
     // в другие состояния
  };
```

```
  char c; // текущий символ
  State * temp, *next; // текущее и
     //следующее состояния автомата
(ДС)
public:
  void analyze(){ State::st_number=0;
    for (temp=new C, cin>>c;
         c!='#'&&
         !( dynamic_cast<ERR *>
(temp));
         cin>>c
         ){
            next=(*temp)(c);
            delete temp;
            temp=next;
    }
    if ( c!='#' ||
         typeid(*temp)!=typeid(D)
        )
          cout<< "ERROR" <<endl;
    else
          cout<< "OK, количество \
          состояний в пройденном пути
="
          << State::st_number <<endl;
    delete temp;
  }
};

int Scanner::State::st_number;
```

а) Определите начальный символ грамматики G_{left} по описанию класса *Scanner*. (Обратите внимание прежде всего на метод *analyze()*). *Ответ:*

б) Определите язык, порождаемый грамматикой G_{left} (формула или словесное описание).
Ответ:

в) Постройте диаграмму состояний для G_{left} – пусть это будет конечный автомат $\mathcal{U}$.
Ответ:

г) Постройте эквивалентный автомат $\mathcal{U}'$ путем добавления в $\mathcal{U}$ только одной дуги так, чтобы $\mathcal{U}'$ получился бы недетерминированным. (Новые состояния не добавлять.)
Ответ:

д) Добавьте недостающие фрагменты описаний в классы *State*, *D* (и только в них!) так, чтобы:
- метод *analyze*() считал правильными только все цепочки языка *L* (G_{left}),
- никаким способом невозможно было бы создать отдельный объект класса *State,*

- в случае успеха *analyze*() печатал бы количество состояний на пути по диаграмме состояний (ДС для G_{left}) из начального состояния в заключительное.

Коллоквиум_2019 Ответы и критерии

За каждую задачу максимум 50 баллов.

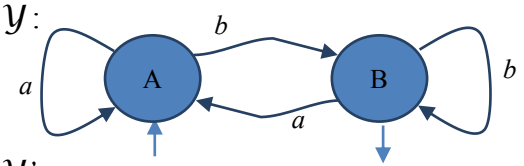
Вариант 1.

1. Ответы:

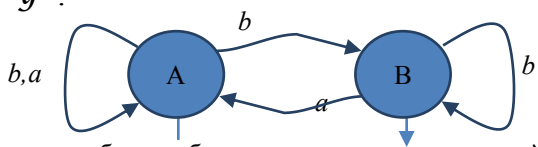
е) B – начальный символ в $G_{left.}$

ж) $L(G_{left}) = \{\omega b \mid \omega \in \{a, b\}^*\}$. Непустая цепочка в алфавите $\{a, b\}$, заканчивающаяся на b.

з) $\mathcal{Y}$:



и) $\mathcal{Y}'$:



Ответом может быть добавление через запятую другой буквы на любую из четырех дуг.

к)

```
class State { public: static int st_count;
          State(){st_count++;}
          virtual State * operator ()(char c) const=0;
          virtual ~State(){};
};
class B :public State { public:
   State * operator ()(char c)const{
            if (c=='b') return new B;
            else if (c=='a') return new A;
            else return new ERR;
   }
};
```

Критерии: Пункты (а, б, в): по -5 каждый нерешенный или неправильный пункт. За ошибку в пункте (г) : -10. Пункт (д): по -5 за каждую ошибку – нет `virtual`, нет `=0`, `public` и т.п., неправильно запрограммирован переход в новое состояние и т.д., но в целом снижаем не более 25 баллов за этот пункт. За отсутствие деструктора не снижаем.

2. Ответы:

е) $P=\{ S\to aAbB \; ; A\to aA \mid \varepsilon \; ; B\to b \}$

ж) $first(aA)\cap first(\varepsilon) = \emptyset$, $first(A) \cap follow(A) = \emptyset \Rightarrow$ выполняется критерий применимости.

з) Для неоднозначных грамматик нарушается хотя бы один пункт критерия применимости рекурсивного спуска. Из (б) следует, что грамматика G однозначна.

и) $S\to aAbB \mid abB \; ; \; A\to aA \mid a \; ; \; B\to b$

к)

```
void analyze(){
  try{ result.clear();
       gc();
       S();
       if (c!='#') throw c;
       cout<< "Результат перевода для входной цепочки: ";
       for (list<char>::iterator p=result.begin();p!=result.end();p++)
            cout<<*p;

       cout<<endl;
  }
  catch(char c){cout<< "Ошибочный символ "<< c<<endl;
}
```

Критерии: Пункты (а, б, в): по -5 каждый нерешенный или неправильный пункт. За ошибку в пункте (г) : -10. Пункт (д): по -7 за каждую ошибку – неправильно описан итератор (`auto` допускается), печать «задом наперед», нет печати требуемого сообщения и символа и т.д., но в целом снижаем не более 25 баллов за этот пункт.

Коллоквиум_2019 Ответы и критерии

За каждую задачу максимум 50 баллов.

Вариант 2.

1. Ответы:

а) $P=\{ S\to bBDd \mid d\ ;\ B\to bB \mid \varepsilon\ ;\ D\to d \}$

б) $first(bBDd)\cap first(d) = \varnothing,\ \ first(bB)\cap first(\varepsilon) = \varnothing,\ \ first(B) \cap follow(B) = \varnothing \ \ \Rightarrow$ выполняется критерий применимости.

в) Для неоднозначных грамматик нарушается хотя бы один пункт критерия применимости рекурсивного спуска. Из (б) следует, что грамматика G однозначна.

г) $S\to bBDd \mid bDd \mid d\ \ ;\ \ B\to bB \mid b\ ;\ \ D\to d$

д)

```
void analyze(){
  try{ trans.clear();
       gc();
       S();
       if (c!='#') throw c;
       cout<< "Результат перевода для входной цепочки: ";
       for (list<char>::iterator p=trans.begin();p!=trans.end();p++)
            cout<<*p;

       cout<<endl;
  }
  catch(char c){cout<< "Ошибочный символ "<< c<<endl;
}
```

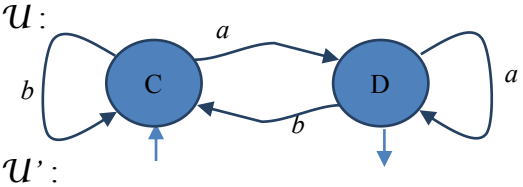
Критерии: Пункты (а, б, в): по -5 каждый нерешенный или неправильный пункт. За ошибку в пункте (г) : -10. Пункт (д): по -7 за каждую ошибку – неправильно описан итератор (`auto` допускается), печать «задом наперед», нет печати требуемого сообщения и символа и т.д., но в целом снижаем не более 25 баллов за этот пункт.

2. Ответы:

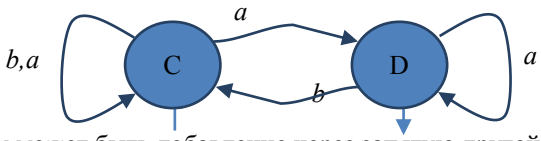
а) D – начальный символ в G_{left}.

б) $L(G_{left}) = \{\omega a \mid \omega \in \{a, b\}^*\}$. Непустая цепочка в алфавите $\{a, b\}$, заканчивающаяся на a.

в) $\mathcal{U}$:



г) $\mathcal{U}'$:



Ответом может быть добавление через запятую другой буквы на любую из четырех дуг.

```
class State { public: static int st_number;
          State(){st_number++;}
          virtual State * operator ()(char c) const=0;
          virtual ~State(){};
};
class D    :public State   {public:
   State * operator ()(char c)const{
             if (c=='a') return new D;
             else if (c=='b') return new C;
             else return new ERR;
   }
```

```
};
```

Критерии: Пункты (а, б, в): по -5 каждый нерешенный или неправильный пункт. За ошибку в пункте (г) : -10. Пункт (д): по -5 за каждую ошибку – нет `virtual`, нет `=0`, `public` и т.п., неправильно запрограммирован переход в новое состояние и т.д., но в целом снижаем не более 25 баллов за этот пункт. За отсутствие деструктора не снижаем.

2019 год
Экзамен

Вариант 1_2019 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. Добавить в приведенную программу необходимые конструкции C++ (**не заменяя и не удаляя** имеющиеся) так, чтобы при ее компиляции не было ошибок, а на печать выдалось:

```
Constr_B
Constr_A

Destr_A

Destr_B
```

```
struct B {
        virtual void f() = 0;
        B(){ cout << "Constr_B\n";}
        ~B(){ cout << "Destr_B\n"; }
};
int main () {
        NN::A a;

        B *pb = &a;

        return 0; }
```

2. Добавить **только в класс A** необходимые элементы, **не вводя никаких дополнительных членов класса A**, ничего **не удаляя и не заменяя**, так, чтобы в получившейся программе не было ошибок, а на печать выдалось**: 7 3**

```
class A {
    int n;
public:
    int f(int n = 0) { return n;}
    A(int a) { n = a;}
};
```

```
int main () {
    A a, b (3);
    cout << a.f() << " " << b.f(1) << endl;
    return 0;
}
```

3. Найти ошибки в приведенной программе и **объяснить** их. Исправить найденные ошибки. **Что напечатает** получившаяся программа?

```
template <class T>
T::value_type  f (const T& t) {
        if(t.size())
                return t[t.size()-1];
        else
                return (typename T::value_type)0;
}
int main () {
        vector <int> v;
        list <int> l(2,3);
        cout << f<vector<int>>(v) << f<list<int>>(l) << endl;
        return 0; }
```

4. Удалить некоторые элементы **в теле функции *main()***, **ничего не добавляя и не заменяя**, так, чтобы в получившейся программе не было ошибок, а на экран выдалось:

A() A() B() ~B() ~A() ~A() ~A() ~B()~A()

```
struct A {
   A(){ cout << "A()  "; }
  ~A(){ cout << "~A()  "; }
};
struct B:A {
   B(){ cout << "B()  "; }
   ~B(){ cout << "~B()  "; }
};
```

```
int main () {
      try { int n = -1;  A a;  B b;
            if(n > 0) throw a;
            else  throw b;
      }
      catch (B & x) {return 1;}
      catch (A & y) {return 2;}
}
```

5. Внести необходимые изменения в программу, **не вводя дополнительных членов** класса *C*, **не меняя функцию *main()*** и ничего **не удаляя**, так, чтобы в ней не было ошибок, а на экран выдалось: **6 18** .

```
class C {
      int i;
public:
      explicit C(int k = 0){ i = k; }
      void i_out(){ cout << i << " ";}
};
```

```
int main ()
   C a(3), b(2);
   (a*b).i_out();
   (a*b*a).i_out();
   return 0;
```

6. Даны грамматики G_1 и G_2.

G_1: $S \to ASB \mid \varepsilon$
$AB \to BA$
$A \to a$
$B \to b$

G_2: $S \to aSbS \mid bSaS \mid \varepsilon$

а) Эквивалентны ли G_1 и G_2? Ответ обосновать. *Ответ*:_____
_____
б) Определить тип грамматики G_1 : _____
в) Определить тип грамматики G_2 : _____
г) Определить тип языка $L(G_1)$: _____
д) Определить тип языка $L(G_2)$ _____

7. Дана грамматика многочленов от одной переменной x с односимвольными коэффициентами и степенями, к которой применим метод рекурсивного спуска. Фигурные скобки в грамматике означают итерацию. Добавьте в грамматику действия только вида ⟨ *cout* <<″символы″; ⟩, реализующие формальный перевод $\tau = \{ (p, r) \mid p$ – многочлен, r – первообразная многочлена $p \}$.

Пример: 2x^2 – 3x^0 + 5x^1 ➔ 2x^3 /3 – 3x^1 + 5x^2 /2 + C

Ответ:

$S \to A \{ Z A \}$
$Z \to + \mid -$
$A \to Kx\text{^}N$
$N \to 0 \mid 1 \mid 2$
$K \to 0 \mid 1 \mid 2$

8. Дана леволинейная грамматика G_{left} :

$S \to Sa \mid Aa$
$A \to Aa \mid b \mid a$

а) Применим ли метод рекурсивного спуска к данной грамматике? Обоснуйте ответ.
Ответ:

б) Постройте эквивалентную детерминированную праволинейную грамматику G_{right}

Ответ:

9. Дана заготовка КС-грамматики $G = \langle \{A, B, S\}, \{a, b\}, P, S \rangle$, где $P = \{ A \to AA \mid a \; ; \; B \to b \; ; \; \boldsymbol{X} \to \boldsymbol{\alpha} \}$.
Найти такие $X \in \{ A, B, S\}$ и $\alpha \in \{ A, B, S, a, b \}^*$ для G, чтобы одновременно выполнялись условия:

(а) G не является приведённой грамматикой;
(б) $L(G) \neq \emptyset$;
(в) грамматика G однозначна.

Ответ:

10. Что такое оптимизация программы? Ниже дан ПОЛИЗ фрагмента программы на языке Си. Оптимизируйте длину кода (постройте более короткий эквивалентный ПОЛИЗ), используя только те виды операций, которые есть в исходном ПОЛИЗе.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ПОЛИЗ	*a*	*b*	>	12	!F	*&a*	c	=	;	16	!	&a	*d*	=	;		
Ответ: ПОЛИЗ																	

Ответы и критерии: B1 - 2019

Каждая задача оценивается максимум в 10 баллов, минимум – 0.

№ 1. Следует добавить:

```
namespace NN {
        struct A:B {
                void f() {}
                 A(){  cout << "Consrt_A\n"; }
                ~A(){ cout << "Destr_A\n"; }
        };
}
```

Критерии: **1.** Нет **namespace**: **-5**.
2. Нет метода **f()** в классе A: **-5**.
3. Другие ошибки: по **-2** за каждую.

№ 2. Следует добавить: …

```
int f(int n = 0) { return this -> n;}
A(int a = 7) { n = a;}  …
```

Критерии: **1.** Нет **this**: **-5**.
2. Нет конструктора умолчания: **-5**.
3. Другие ошибки: по **-2 за** каждую.

№ 3. Ошибки: …

```
template <class T>
typename T::value_type f(const T& t){
        if(t.size())
         return t[t.size()-1]; …..
```

Напечатается: **03** .

Критерии: **1.** Не нашли **typename**: **-5;**
2. Не нашли, неверно исправили **[ ]** : **-5;**
3. Другие ошибки: **-2** каждая.

№ 4. Следует удалить первый **catch (B & b)** и ссылку во втором **catch (A a)**.

Критерии: **1.** За каждое неверное или невыполненное удаление по **-5**.

№ 5. Следует описать и объявить в классе C дружественную функцию (или, если без **friend**, добавить еще один **public:**):

```
…    friend C operator * (C, C); …

   C operator *(C a, C b){
```

```
        C r(a.i * b.i);
        return r;
    }
```

Критерии: **1.** Нет **`friend (`**или **`public:)`** : **-5**.
2. Нет/неверная функция **`operator*()`**: **-5**.
3. Другие ошибки: по **-2** за каждую.

№6. (а) да, обе грамматики порождают язык цепочек в алфавите $\{a, b\}$, в которых символов a и b поровну;
(б) тип 0, то есть грамматика без ограничений;
(в) тип 2, контекстно-свободная грамматика;
(г) тип 2, контекстно-свободный язык;
(д) тип 2, контекстно-свободный язык.

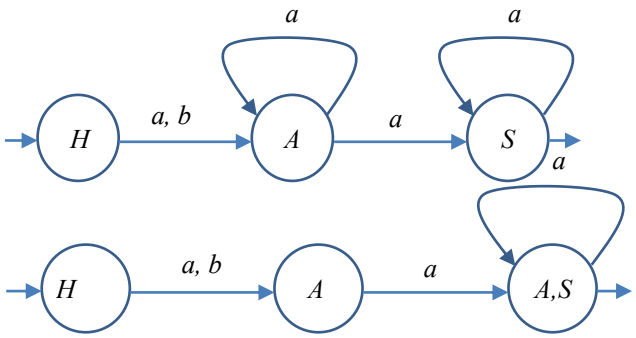
Критерии: (а) за неверный ответ **-4**, какое-то обоснование обязательно;
остальные пункты по **-2** за каждый неверный ответ.

№7. $S \to A\ \{\ Z\ A\}\ \langle$ cout << "+C "; $\rangle$
$Z \to +\ \langle$ cout << "+"; $\rangle\ |\ -\ \langle$ cout <<"–"; $\rangle$
$A \to K\ x\hat{}\ \langle$ cout << "x^"; $\rangle\ N$
$N \to\ 0\ \langle$ cout << "1 "; $\rangle\ |\ 1\ \langle$ cout << "2/2"; $\rangle\ |\ 2\ \langle$ cout << "3/3"; $\rangle$
$K \to\ 0\ \langle$ cout << "0"; $\rangle\ |\ 1\ \langle$ cout << "1"; $\rangle\ |\ 2\ \langle$ cout << "2"; $\rangle$

Критерии: за каждую неверную или отсутствующую печать в грамматике с действиями по **-2** . За альтернативный функционально эквивалентный ответ не снижаем.

№8 (а) нет, так как $first(Sa) \cap first(Aa) \supset \{a\} \neq \varnothing \Rightarrow$ нарушается критерий применимости;
(б)

$S \to Sa \mid Aa$
$A \to Aa \mid b \mid a \quad \Rightarrow$



$\Rightarrow$ {детерминизация} $\Rightarrow$

$\Rightarrow \quad H \to b\ A \mid a\ A$
$\quad\ A \to a\ X \quad$, где $X=\{A,S\}$
$\quad\ X \to \varepsilon \mid aX$

Критерии: за каждый неправильный или отсутствующий пункт: **-5**.
Возможна другая эквивалентная грамматика в качестве ответа в пункте (б).
Промежуточные преобразования необязательны. Допускается введение нового заключительного состояния и добавление перехода в него по маркеру конца цепочки $\perp$.

№9. Подойдет любое правило вида $S \to \alpha$, где α – любая цепочка (в т.ч. пустая) в алфавите $\{a, b, B\}$.

Критерии: за неправильный или отсутствующий ответ **-10**.

№10. Оптимизация программы – это изменение транслируемой программы (в основном переупорядочивание и замена операций) с целью получения более эффективного объектного кода.
Исходный ПОЛИЗ соответствует фрагменту на Си: ***if*** *(a > b) a = c;* ***else*** *a = d;*
Он эквивалентен фрагменту *a = a > b ? c : d;* , которому соответствует более короткий ПОЛИЗ:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ПОЛИЗ	&a	a	b	>	10	!F	c	11	!	d	=	;					

Критерии: неверно приведено понятие оптимизации программы: **-5** ;
не приведен эквивалентный более короткий ПОЛИЗ: **-5**.

Вариант 2_2019 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых включений и директивы `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. **Добавить** в приведенную программу необходимые конструкции C++ (**не заменяя и не удаляя** имеющиеся) так, чтобы при ее компиляции не было ошибок, а на печать выдалось:

```
Constr_B
Constr_D
Destr_D
Destr_B
```

```
namespace SS { struct B {
                    virtual void g() = 0;
                    B(){ cout << "Constr_B\n";}
                    ~B(){ cout << "Destr_B\n"; }
             };   }
int main () {
       D d;
       B *pb = &d;
       return 0; }
```

2. Добавить **только в класс C** необходимые элементы, **не вводя никаких дополнительных членов класса C**, ничего **не удаляя и не заменяя**, так, чтобы в получившейся программе не было ошибок, а на печать выдалось**: 5 3**

```
class C {
    int i;
public:
    C(int a) { i = a;}
    int f(int i = 0) { return  i; }
};
```

```
int main () {
    C c1(5), c2;
    cout << c1.f() << " " << c2.f(1) << endl;
    return 0;
}
```

3. Найти ошибки в приведенной программе и **объяснить** их. Исправить найденные ошибки. **Что напечатает** получившаяся программа?

```
template <class T>
typename T::value_type  f ( T& t) {
       if(t.size()) { t.push_front(t.back());
              return t.front(); }
       else
              return ( T::value_type ) 0;
}
int main () {
       list <int> l(2, 5);
       vector <int> v;
       cout << f<vector<int>>(l) << f<list<int>>(v) << endl;
       return 0; }
```

4. **Добавить** некоторые элементы **в тело функции *main()***, **ничего не удаляя и не заменяя**, так, чтобы в получившейся программе не было ошибок, а на экран выдалось: ***A() B() A(const A&) ~ B() ~ A() ~ A()***

```
struct A {
   A() { cout << "A()  "; }
   A(const A& a){ cout << "A(const A&)  "; }
  ~A(){ cout << "~A()  "; }
};
struct B:A {
   B(){ cout << "B()  "; }
   ~B(){ cout << "~B()  "; }
};
```

```
int main () {
       try { B b;  A *pa = &b;
              throw *pa;
       }
       catch (B x) { return 1;}
}
```

5. Внести необходимые изменения в программу, **не вводя дополнительных членов** класса *C*, **не меняя функцию *main()*** и ничего **не удаляя**, так, чтобы в ней не было ошибок, а на экран выдалось: **2.5 0.5** .

```
class P {
```

```
int main ()
```

```
        double d;
public:
        explicit P (double k = 0){ d = k; }
        void out_d(){ cout << d << " ";}
};
```

```
P a(5), b(2);
(a/b).out_d();
(a/b/a).out_d();
return 0;
```

6. Даны грамматики G_1 и G_2.

G_1: $S \to PRS \mid \varepsilon$
$PR \to RP$
$RP \to PR$
$R \to 0$
$P \to 1$

G_2: $S \to 1S0 \mid 0S1 \mid SS \mid \varepsilon$

е) Эквивалентны ли G_1 и G_2? Ответ обосновать. *Ответ*:_____

_____

ж) Определить тип грамматики G_1 : _____

з) Определить тип грамматики G_2 : _____

и) Определить тип языка $L(G_1)$: _____

к) Определить тип языка $L(G_2)$ _____

7. Дана грамматика многочленов от одной переменной x с односимвольными коэффициентами и степенями в троичной системе счисления, к которой применим метод рекурсивного спуска. Фигурные скобки в грамматике означают итерацию. Добавьте в грамматику действия только вида ⟨ *cout* <<″символы″; ⟩, реализующие формальный перевод $\tau = \{ (t, w) \mid t$ – многочлен, w – производная многочлена $t \}$.

Пример: 1*x^2 – 1 + 2*x^1 ➔ 1*x^1 *2 – 1*0 + 2*x^0

Ответ:

$S \to A \{ Z A \}$
$Z \to + \mid -$
$A \to KL$
$L \to$ *x^N $\mid \varepsilon$
$N \to 1 \mid 2$
$K \to 0 \mid 1 \mid 2$

8. Дана леволинейная грамматика G_{left} :

$S \to Sa \mid Ba \mid a$
$B \to Ba \mid b$

в) Применим ли метод рекурсивного спуска к данной грамматике? Обоснуйте ответ.

Ответ:

г) Постройте эквивалентную детерминированную праволинейную грамматику G_{right}

Ответ:

9. Дана заготовка КС-грамматики $G = \langle \{A, B, S\}, \{a, b\}, P, S \rangle$, где $P = \{ A \to AA \mid \varepsilon \; ; \; B \to b \; ; \; \boldsymbol{X \to \alpha} \}$.

Найти такие $X \in \{ A, B, S\}$ и $\alpha \in \{ A, B, S, a, b \}^*$ для G, чтобы одновременно выполнялись условия:

(а) G не является приведённой грамматикой;
(б) $L(G) = \{\varepsilon\}$;
(в) грамматика G неоднозначна.

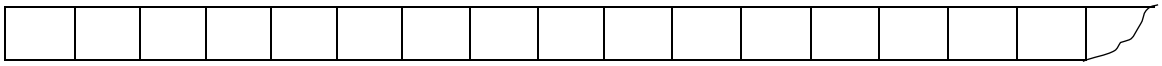
Ответ:

10. Что такое машинно-независимые преобразования при оптимизации программы? Ниже дан ПОЛИЗ фрагмента программы на языке Си. Оптимизируйте длину кода (постройте более короткий эквивалентный ПОЛИЗ), используя только те виды операций, которые есть в исходном ПОЛИЗе.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ПОЛИЗ	*a*	*b*	>	16	!F	*&c*	*d*	=	;	*&a*	*b*	=	;	20	!	*&c*	*d*	=	;

Ответ: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

ПОЛИЗ



Ответы и критерии: B2 - 2019

Каждая задача оценивается максимум в 10 баллов, минимум – 0.

№ 1. Следует добавить: **using namespace** SS; и

```
struct D:B {   или, если using ниже D:SS::B
      void g() {}
       D(){  cout << "Consrt_D\n"; }
      ~D(){ cout << "Destr_D\n"; }
};
```

Критерии: **1.** Нет **using namespace SS**: **-5**.
2. Нет метода **g()** в классе A: **-5**.
3. Другие ошибки : по **-2** за каждую.

№ 2. Следует добавить: …**int** f(**int** i = 0) { **return** this -> i;}

C(**int** a = 3) { i = a;} …

Критерии: **1.** Нет **this**: **-5**.
2. Нет конструктора умолчания: **-5**.
3. Другие ошибки: по **-2** за каждую.

№ 3. Ошибки*:* … ***template*** *<**class** T>* …

*t.**push_front**(t.back());//*заменить на *t.insert(t.begin(),t.back())*

return *(**typename** T::value_type)0;* ….

Напечатается: 50

Критерии: **1.** Не нашли ***typename***: **-5**.
2. Не нашли, неверно исправили ***push_front***: **-5**.
3. Другие ошибки: по **-2** за каждую.

№ 4. Следует добавить второй **catch (A & a)**.

Критерии: **1.** Нет **catch**: **-10**.
2. Нет только ссылки: **-5**.

№ 5. Следует описать и объявить в классе P дружественную функцию:

… friend P operator / (P, P); … или (без **friend**) добавить перед **double d** еще одно **public:**

```
P operator/(P a, P b){
      P r(a.d / b.d);
      return r;
}
```

Критерии: **1.** Нет **friend** или **public**: **-5**.
2. Нет/неверная функция **operator/()**: **-5**.
3. Другие ошибки: по **-2** за каждую.

№6. (а) да, обе грамматики порождают язык цепочек в алфавите $\{0, 1\}$, в которых символов *0* и *1* поровну;
(б) тип 0, то есть грамматика без ограничений;
(в) тип 2, контекстно-свободная грамматика;
(г) тип 2, контекстно-свободный язык;
(д) тип 2, контекстно-свободный язык.

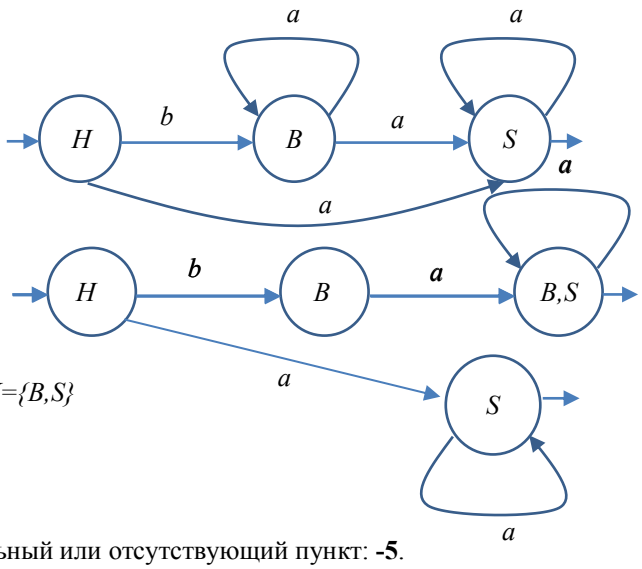
Критерии: (а) за неверный ответ **-4**, какое-то обоснование обязательно;
остальные пункты по **-2** за каждый неверный ответ.

№7.
$S \to A\ \{ Z\ A\}$
$Z \to +\ \langle$ cout << "+ " ;$\rangle \mid -\ \langle$ cout << "–";$\rangle$
$A \to K\ L$
$L \to$ *x^ $\langle$ cout << "*x^ ";$\rangle\ N \mid \varepsilon\ \langle$ cout << "*0";$\rangle$
$N \to\ 1\ \langle$ cout << "0";$\rangle \mid\ 2\ \langle$ cout << "1*2";$\rangle$
$K \to\ 0\ \langle$ cout << "0";$\rangle \mid\ 1\ \langle$ cout << "1";$\rangle \mid\ 2\ \langle$ cout << "2";$\rangle$

Критерии: за каждую неверную или отсутствующую печать в грамматике с действиями по **-2** . За альтернативный функционально эквивалентный ответ не снижаем.

№8 (а) нет, так как $first(Sa) \cap first(a) \supset \{a\} \neq \varnothing \Rightarrow$ нарушается критерий применимости;
(б)

$S \to Sa \mid Ba \mid a$
$B \to Ba \mid b \quad \Rightarrow$



⇒{детерминизация}⇒

$H \to bB \mid aS$
$\Rightarrow \quad B \to aX \quad , \quad$ где $X=\{B,S\}$
$X \to \varepsilon \mid aX$
$S \to \varepsilon \mid aS$

Критерии: за каждый неправильный или отсутствующий пункт: **-5**.
Возможна другая эквивалентная грамматика в качестве ответа в пункте (б).
Промежуточные преобразования необязательны. Допускается сведение заключительных состояний в одно новое заключительное состояния путем добавления перехода по маркеру конца цепочки $\perp$.

№9. Подойдет любое правило вида $S \to \alpha$, где α – любая непустая цепочка в алфавите $\{A\}$.
Правило $S \to \varepsilon$ не является правильным ответом, так как не выполняется пункт (в).
Критерии: за неправильный или отсутствующий ответ **-10** ; **-5** за каждое лишнее правило (альтернативу).

№10. Машинно-независимая оптимизация программы – машинно-независимые преобразования исходной программы производятся в основном над ее внутренним представлением и основаны на известных математических и логических преобразованиях. Исходный ПОЛИЗ соответствует фрагменту на Си:
if *(a>b) {c=d; a=b;}* ***else*** *c=d;*
Этот фрагмент эквивалентен фрагменту *c = d;* ***if*** *(a > b) a = b;* , которому соответствует более короткий ПОЛИЗ:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
&c	d	=	;	a	b	>	14	! F	&a	b	=	;				

Критерии: неверно приведено понятие машинно-независимой оптимизации программы: **-5**;
не приведен эквивалентный более короткий ПОЛИЗ: **-5**.

2021 год
Экзамен

Экзамен_2021_1 ФИО_____№ группы_____

- ✓ Проанализировать следующие грамматики:

1. $\begin{cases} S \to aSAc \mid \varepsilon \\ A \to bA \mid \varepsilon \end{cases}$ 2. $\begin{cases} S \to aSBc \mid \varepsilon \\ B \to aB \mid b \end{cases}$ 3. $\begin{cases} S \to Ac \mid Sb \\ A \to Ac \mid a \end{cases}$ 4. $\begin{cases} S \to Aa \mid Sb \\ A \to Ac \mid b \end{cases}$

Никак **не преобразовывая** грамматики, выбрать одну из них для построения по грамматике корректного анализатора **методом рекурсивного спуска**, а другую - для анализатора по **конечному автомату**.
Выбор **ОБОСНОВАТЬ**. *(По 10 баллов за каждый правильный и правильно обоснованный ответ)*

- ✓ Какие **языки** порождают выбранные грамматики? *(По 5 баллов за каждый правильный ответ)*
- ✓ Каков **тип** этих языков по Хомскому? *(По 5 баллов за каждый правильный ответ)*

- ✓ Описать **два** класса ***LexAn*** и ***SyntAn***, каждый из которых - **производный** от заданного класса ***PPP***: *(По 25 баллов за каждый правильно в соответствие условию задачи описанный класс)*

```
struct PPP {
    static int x;
    PPP (int t) { x += t;}
    virtual void gc ( ) = 0;                    // получает очередной символ анализируемой цепочки
    virtual const char * operator + (const char * str) = 0;      // выполняет анализ цепочки str
                                                // по выбранной грамматике, возвращает строку-сообщение о
                                                // принадлежности (или нет) цепочки языку, порождаемому грамматикой.
    virtual ~PPP( ) { cout << x << endl;}
};
int PPP::x = 0;
```

- ✓ а также **одну** функцию *run_an (**const char** * str), запускающую анализатор по входной цепочке, (10 баллов за правильно описанную функцию)*
такие, что в следующей функции *main ()* не будет ошибок:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string  str;
    cin >> str;                     // ввод цепочки для анализа по конечному автомату
    const char * s = str.c_str();   // возвращает указатель на строку(цепочку) в смысле языка Си
    run_an  <LexAn> (s);

    cin >> str;                     // ввод цепочки для анализа методом рекурсивного спуска
    s = str.c_str();
    run_an  <SyntAn> (s);
    return 0;
}
```

а при выполнении программы на экран будет выдано:

Yes!!! // или *No!* – в случае непринадлежности цепочки языку

Magic number – 3
Success!!! // или *Bad string!* – в случае непринадлежности цепочки языку
Magic number – 8

Экзамен_2021_1. Ответы и критерии оценки
Исходный балл - **100**

1. Выбираем для анализа по КА регулярную грамматику, для которой автомат детерминирован – это №4 (другие или не регулярные, или анализ по ним недетерминированный)
   - неверный выбор (нет обоснования) - **-10**
2. Для анализа РС-методом выбираем КС-грамматику, к которой метод применим, это № 1 (к другим метод неприменим: левая рекурсия или плохая ε-альтернатива)
   - неверный выбор (нет обоснования) - **-10**

1) $\begin{cases} S \to aSAc \mid \varepsilon \\ A \to bA \mid \varepsilon \end{cases}$ 2) $\begin{cases} S \to aSBc \mid \varepsilon \\ B \to aB \mid b \end{cases}$ 3) $\begin{cases} S \to Ac \mid Sb \\ A \to Ac \mid a \end{cases}$ 4) $\begin{cases} S \to Aa \mid Sb \\ A \to Ac \mid b \end{cases}$

3. $L(1) = \{ a^n (b^{m_n} c)^n \mid n >= 0, m_n >= 0\}$ - **-5**, тип 2 - **-5**
4. $L(4) = \{ b\, c^n\, a\, b^m \mid n, m >= 0\}$ - **-5**, тип 3 - **-5**

```
class LexAn : public PPP {
        const char * p;
        char c;
        void gc() { c = *p++; }                    // нет функции - -10, неверная - -5
public:
        LexAn () : PPP(3){ }                       // нет конструктора или списка инициализации - -10
        const char * operator + (const char * str) {          // неверный алгоритм - -15,
          try{  p = str;                                      // мелкие ошибки - -2 за каждую
                enum state {H, A, S} CS;
                CS = H;
                do {    gc();
                        switch(CS){
                                case H: if (c == 'b') CS = A;
                                        else  throw c;
                                        break;
                                case A: if (c == 'a') CS = S;
                                        else
                                        if (c == 'c');
                                                else   throw c;
                                        break;
                                case S: if (c == 'b');
                                        else
                                                if (c == '\0')
                                                        return "Yes!";
                                                else    throw c;
                        }
                }
                while(true);
          }
          catch(char){ return "NO!";}
        }
        ~LexAn() { cout << "Magic numder - "; }

};

class SyntAn : public PPP{
        const char * p;
        void gc() { c = *p++; }                    // нет функции - -10, неверная - -5
```

```
        char c;
        void S() {
                if(c =='a'){
                        gc(); S(); A() ;
                        if(c =='c')
                                gc();
                        else
                                throw c;
                }
        }
        void A() {
                if(c =='b'){
                        gc();
                        A();
                }
        }                                       // ошибки в РС-методе - -5 за каждую
public:
        SyntAn() : PPP(5) { }                   // ошибки как в LexAn – караются 1 раз
        ~SyntAn() {cout << "Magic numder - ";}
        const char * operator + (const char * str) {        // неверный алгоритм - -15,
          try {  p = str;                                   // мелкие ошибки - -2 за каждую
                gc();
                S();
                if (c == '\0')
                        return "Succsess!!!";
                else
                        throw c;
           }
            catch(char) { return "Bad string!"; }
        }
};

template <class T>                      // неверная функция - -10
void  run_an (const char * str) {
        T  t;
        cout << t + str  << endl;
}
```

Неверный вывод - **-5** за каждый
(1 и/или 3, 2 и/или 4 неверные строки вывода караются по одному разу)

Экзамен_2021_2 ФИО_____№ группы_____

- ✓ Проанализировать следующие грамматики:

1. $\begin{cases} S \to 0A \mid 1S \\ A \to 1A \mid 1 \end{cases}$ 2. $\begin{cases} S \to aSBc \mid \varepsilon \\ B \to bB \mid c \end{cases}$ 3. $\begin{cases} S \to A0 \mid S1 \\ A \to A1 \mid 0 \end{cases}$ 4. $\begin{cases} S \to cASa \mid \varepsilon \\ A \to aA \mid \varepsilon \end{cases}$

  Никак **не преобразовывая** грамматики, выбрать одну из них для построения по грамматике корректного анализатора **методом рекурсивного спуска**, а другую - для анализатора по **конечному автомату**.
  Выбор **ОБОСНОВАТЬ**. *(По 10 баллов за каждый правильный и правильно обоснованный ответ)*

- ✓ Какие **языки** порождают выбранные грамматики? *(По 5 баллов за каждый правильный ответ)*
- ✓ Каков **тип** этих языков по Хомскому? *(По 5 баллов за каждый правильный ответ)*

- ✓ Описать **два** класса ***LexAn*** и ***SyntAn***, каждый из которых - **производный** от заданного класса ***Base***: *(По 25 баллов за каждый правильно в соответствие условию задачи описанный класс)*

```
struct Base {
    static int n;
    Base (int t) { n *= t;}
    virtual void gc ( ) = 0;                    // получает очередной символ анализируемой цепочки
    virtual const char * operator * (const char * str) = 0;      // выполняет  анализ цепочки str
                                                // по выбранной грамматике,  возвращает строку-сообщение о
                                                // принадлежности  (или нет) цепочки языку, порождаемому грамматикой.
    virtual ~ Base ( ) { cout << n << endl;}
};
int Base::n = 1;
```

✓ а также **одну** функцию *analyse (**const char** * str), запускающую анализатор по входной цепочке, (10 баллов за правильно описанную функцию)*
такие, что в следующей функции *main ( )* не будет ошибок:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string  str;
    cin >> str;                    // ввод цепочки для анализа по конечному автомату
    const char * s = str.c_str();  // возвращает указатель на строку(цепочку) в смысле языка Си
    analyse  <LexAn> (s);

    cin >> str;                    // ввод цепочки для анализа методом рекурсивного спуска
    s = str.c_str();
    analyse  <SyntAn> (s);
    return 0;
}
```

а при выполнении программы на экран будет выдано:

```
Yes!!!          // или   No! – в случае непринадлежности цепочки языку
Lucky number – 7
Success!!!      // или   Bad string! – в случае непринадлежности цепочки языку
Lucky number – 777
```

Экзамен_2021_2. Ответы и критерии оценки
Исходный балл - **100**

5. Выбираем для анализа по КА регулярную грамматику, для которой автомат детерминирован – это №3 (другие или не регулярные, или анализ по ним недетерминированный)
   - неверный выбор (нет обоснования) - **-10**
6. Для анализа РС-методом выбираем КС-грамматику, к которой метод применим, это № 2 (к другим метод неприменим: левая рекурсия, плохая ε-альтернатива, пересечение множеств *first(...)* в альтернативах)
   - неверный выбор (нет обоснования) - **-10**

1) $\begin{cases} S \to 0A \mid 1S \\ A \to 1A \mid 1 \end{cases}$ 2) $\begin{cases} S \to aSBc \mid \varepsilon \\ B \to bB \mid c \end{cases}$ 3) $\begin{cases} S \to A0 \mid S1 \\ A \to A1 \mid 0 \end{cases}$ 4) $\begin{cases} S \to cASa \mid \varepsilon \\ A \to aA \mid \varepsilon \end{cases}$

7. $L(2) = \{ a^n (b^{m_n} cc)^n \mid n >= 0, m_n >= 0\}$ - **-5**, тип 2 - **-5**
8. $L(3) = \{ 0\ 1^n\ 0\ 1^m \mid n, m >= 0\}$ - **-5**, тип 3 - **-5**

```
class LexAn : public Base {
        const char * p;
        char c;
        void gc() { c = *p++; }                         // нет функции - -10, неверная - -5
public:
        LexAn () : Base (7){ }          // нет конструктора или списка инициализации -  -10
        const char * operator * (const char * str) {            // неверный алгоритм -  -15,
          try{  p = str;                                        // мелкие ошибки -  -2 за каждую
                enum state {H, A, S} CS;
                CS = H;
                do {    gc();
                        switch(CS){
                                case H: if (c == '0') CS = A;
                                        else  throw c;
                                        break;
                                case A: if (c == '0') CS = S;
                                        else
                                        if (c == '1');
                                                else throw c;
                                        break;
                                case S: if (c == '1');
                                        else
                                                if (c == '\0')
                                                        return "Yes!";
                                                else   throw c;
                        }
                }
                while(true);
          }
          catch(char){return "NO!";}
        }
        ~LexAn() { cout << "Magic number - "; }

};

class SyntAn : public Base {
        const char * p;
        void gc() { c = *p++; }                 // нет функции - -10, неверная - -5
        char c;
        void S() {
                if(c =='a'){
                        gc(); S(); B() ;
                        if(c =='c')
                                gc();
                        else
                                throw c;
                }
        }
        void B() {
                if(c =='b'){  gc();  B();
                }
                else
                if (c == 'c') gc();
                else
                        throw c;
        }                                               // ошибки в РС-методе - -5 за каждую
public:
        SyntAn() : Base (111) { }                       // ошибки как в LexAn – караются 1 раз
        ~SyntAn() {cout << "Lucky number - ";}
```

```
        const char * operator * (const char * str) {          // неверный алгоритм - -15,
          try {   p = str;                                    // мелкие ошибки - -2 за каждую
                  gc();
                  S();
                  if (c == '\0')
                          return "Succsess!!!";
                  else
                          throw c;
           }
          catch(char) { return "Bad string!"; }
        }
};

template <class T>                      // неверная функция - -10
void   analyse (const char * str) {
        T  t;
        cout << t * str  << endl;
}
```

Неверный вывод - **-5** за каждый
(1 и/или 3, 2 и/или 4 неверные строки вывода караются по одному разу)