Экзамен_2023_вар.1 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых директив включений и `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. Вычеркните неразрешимые вызовы функции g(). Что напечатает получившаяся программа?

```
struct B { operator int () {return 3;} };

void g(B& b, double d){cout << "g(B&, double)\n";}
void g(char c, const char * b){cout << "g(char, const char*)\n";}
void g(char c, int d){cout << "g(char, int)\n";}
```

```
int main() {
  B b;
  g (1.5, 0);
  g (b, 2);
  g (2, b);
}
```

2. Что такое **абстрактный тип данных** (АТД, не путать с абстрактным классом) в языке Си++? Привести пример описания и использования такого класса.

3. Если при компиляции следующей программы будут ошибки, **объясните их.** Исправьте ошибки**,** модифицируя **только функцию main(), ничего в ней не удаляя** каким-либо образом.

```
namespace N1 {int x=1, y=3;
              void f() {cout << "f()\n";}
              void g() {cout << "g()\n";}
}
namespace N2 {int x=5, y=7;
              int f() { return 1; }
              int g (int a) {return x = a; }
}
```

```
using namespace N1;
using namespace N2;

int main ( ) {   g();
                 y = x + g(1) + f();
}
```

4. Опишите одну функцию *F* так, чтобы вычисление выражения *c = F(a, b)* с переменными любого одинакового типа (имеющего соответствующую операцию присваивания) приводило к тому, что в переменной *a* установится исходное значение *b*, в переменной *b* – значение, получаемое выражением *b*=1, а в переменной *c* – первоначальное значение *a*. Приведите пример описания подходящего класса для переменных *a*, *b*, *c*.

5. Следующая программа построена по конечному автомату $\mathcal{A}$ с состояниями *A* и *B* и допускает те и только те входные цепочки, которые допускает автомат $\mathcal{A}$. (а) Опишите конечный автомат $\mathcal{A}$ (в виде диаграммы состояний). (б) Какой язык задает автомат $\mathcal{A}$? (охарактеризовать множество цепочек).

```
class Scanner {
    static char c;
class State {
public:
    State * next = this;
    virtual State * operator->() const=0;
    virtual ~State(){}
};
class stA: public State { public:
    State * operator ->() const{
      if (c=='b') return  &B;
      else if (c=='a') return  &A;
      else throw c;
    }
};
class stB: public State { public:
    State * operator ->() const{
      if (c=='b') return  &A;
      else if (c=='a') return  &B;
      else  throw c;
    }
};
static stB B; // вершина  B
static stA A; // вершина  A
State  *vert; // указатель текущей вершины
        //на диаграмме состояний автомата
```

```
public:
void gc(){cin.get(c);}//считать очередной
                      //символ
bool accept(){
  try{
      vert=&A;
      while (gc(),!cin.eof()){
          vert=(*vert)->next;
      }
      return typeid(*vert)==typeid(stA);
    }
    catch(...) {return false;}
  }
}; // конец класса Scanner

char Scanner::c;
Scanner::stA Scanner::A;
Scanner::stB Scanner::B;

int main()
{
  Scanner G1;
  cout<<(G1.accept()?"\nYes":"\nNo")
      <<endl;
}
```

6. Дана программа синтаксического анализа по принципу рекурсивного спуска для грамматики G с нетерминалами A и S, печатающая анализируемую входную цепочку, выводимую в грамматике.

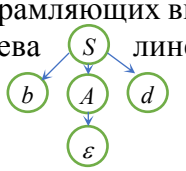
```
class Parser {

  static char c; // текущий символ (лексема)
  static void gc(){cin>>c;} // получить
                //следующий символ (лексему)

  class S {
  public:
    void parse(){
      if (c=='b') {cout<<'b'; gc ();
         A().parse();
         if (c=='d') { cout <<'d'; gc();
         } else throw c;
      } else if (c=='d') {cout<< 'd'; gc();
      } else throw c;
    }
  };
```

```
  class A { public:
    void parse() {
      if (c=='a') {
        cout<<'a'; gc (); A().parse();
      }
    }
  };
public:
  void analyze(){
    try{ gc(); S().parse();
         if (!cin.eof()) throw -1;
    }
    catch(...){cout<< "\nERROR";}
    cout<<endl;
  }
};
char Parser::c;
int main(){  Parser().analyze();}
```

а) Восстановите грамматику G по программе.

б) Не изменяя методы *parse*(), добавьте недостающие описания в классах A и S так, чтобы правильные цепочки переводились данной программой в линейную запись соответствующего дерева вывода. Символу S в линейной записи соответствует пара скобок (и), обрамляющих выводимую из S цепочку, символу A соответствует пара скобок < и >. Например, для дерева 

линейная запись такова: $(b\langle\rangle d)$.

7. Дана грамматика G: $S \to Sa \mid Ab$
$A \to Aa \mid Ab \mid c$

(а) Обоснуйте следующее утверждение: любую автоматную грамматику через детерминизацию диаграммы состояний можно преобразовать в эквивалентную праволинейную, к которой примени́м метод рекурсивного спуска. (б) Преобразуйте заданную грамматику G в соответствии с пунктом (а).

8. Дана КС-грамматика G_a с действиями над глобальными переменными *depth* и *level*. (а) Какой язык задает данная грамматика (какое множество цепочек) ? (б) Построить КС-грамматику G без действий для языка $L(G_a)$, содержащую не более 4-х правил вывода, считая альтернативы.

G_a : $S \to \langle\ depth{=}0;\ level{=}0;\ \rangle\ A\ \langle \textbf{\textit{if}}\ (depth{>}2)\ error();\ \rangle$
$A \to (\ \langle level{+}{+};\ depth = level > depth\ ?\ level\ :\ depth;\ \rangle\ A\ )\ \langle level{-}{-};\ \rangle\ A \mid\ \varepsilon$

9. Может ли быть неоднозначной грамматика, порождающая пустой язык? Обоснуйте ответ.

10. Дан ПОЛИЗ программы на модельном М-языке. (а) Восстановить по ПОЛИЗу тело программы на М-языке. (б) Оптимизируйте ПОЛИЗ по длине, т.е. постройте эквивалентный (печатающий тот же результат) ПОЛИЗ минимальной длины, считая, что вводимое значение b всегда положительное. Операции *mod* (остаток от деления) в М-языке нет, есть деление с отбрасыванием остатка (/), сложение, умножение, вычитание и сравнения. Сокращенный оператор if-then без else есть.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&*a*	5	:=	&*b*	*read*	*a*	*b*	<>	30	!F	*a*	*b*	>	23	!F	&*a*	*a*

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
b	–	:=	28	!	&*b*	*b*	*a*	–	:=	6	!	*a*	*write*		⟩

ПОЛИЗ (оптим.)	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	
														⟩

Экзамен_2023_вар.2 Ф.И.О._____№ группы_____

NB! **Во всех задачах, где это требуется, предполагается наличие необходимых директив включений и `using namespace std`.**

1	2	3	4	5	6	7	8	9	10

1. Вычеркните неразрешимые вызовы функции g(). Что напечатает получившаяся программа?

```
struct B { B(int n = 0){} };

void g(B& b, double d){cout << "g(B&, double)\n";}
void g(char c, const char * b){cout << "g(char, const char*)\n";}
void g(char c, int d){cout << "g(char, int)\n";}
```

```
int main() {
      B b;
      g(1.5, 0);
      g(b, 2);
      g(2, b);
}
```

2. Что такое **функция-друг** в языке Си++? Привести пример описания и использования функции-друга.

3. Если при компиляции следующей программы будут ошибки, **объясните их.** Исправьте ошибки**,** модифицируя **только функцию main(), ничего в ней не удаляя** каким-либо образом.

```
namespace N1 {int x=1, y=3;
              void f() {cout << "f()\n";}
              void g() {cout << "g()\n";}
}
namespace N2 {int x=5, y=7;
              int f() { return 1; }
              int g (int a) { return x = a; }
}
```

```
using namespace N1;
using namespace N2;

int main() { int y;
             y = x + f() + g(3);
}
```

4. Опишите не более двух перегруженных функций *G* так, чтобы вычисление выражения *c=G(a, b)*, с переменными любого одинакового типа *a* и *b* (имеющего операцию сравнения >) и целой переменной *c* приводило к тому, что в переменной *c* будет значение 1, если *a>b*, и значение 0 – иначе. Сравнение Си-строк *a* и *b* должно происходить лексикографически с использованием *strcmp*().

5. Следующая программа построена по конечному автомату $\mathcal{A}$ с состояниями *A* и *B* и допускает те и только те входные цепочки, которые допускает автомат $\mathcal{A}$. (а) Опишите конечный автомат $\mathcal{A}$ (в виде диаграммы состояний). (б) Какой язык задает автомат $\mathcal{A}$*?* (охарактеризовать множество цепочек).

```
class Scanner {
    static char c;
class State {
public:
    State * next = this;
    virtual State * operator->() const=0;
    virtual ~State(){}
};
class stB: public State { public:
    State * operator ->() const{
      if (c=='a') return  &A;
      else if (c=='b') return  &B;
      else throw c;
    }
};
class stA: public State { public:
    State * operator ->() const{
      if (c=='a') return  &B;
      else if (c=='b') return  &A;
      else  throw c;
    }
};
static stA A; // вершина  A
static stB B; // вершина  B
State  *vert; // указатель текущей вершины
         //на диаграмме состояний автомата
```

```
public:
void gc(){cin.get(c);} //считать очередной
                       //символ
bool accept(){
  try{
      vert=&B;
      while (gc(),!cin.eof()){
          vert=(*vert)->next;
      }
      return typeid(*vert)==typeid(stB);
    }
    catch(...) {return false;}
 }
}; // конец класса Scanner

char Scanner::c;
Scanner::stB Scanner::B;
Scanner::stA Scanner::A;

int main()
{
  Scanner G1;
  cout<<(G1.accept()?"\nYes":"\nNo")
      <<endl;
}
```

6. Дана программа синтаксического анализа по принципу рекурсивного спуска для грамматики G с нетерминалами A и S, печатающая анализируемую входную цепочку, выводимую в грамматике.

```
class Parser {

  static char c; // текущий символ (лексема)
  static void gc(){cin>>c;} // получить
                //следующий символ (лексему)

  class A {
  public:
    void parse(){
      if (c=='b') {cout<<'b'; gc ();
         A().parse();
         if (c=='d') { cout <<'d'; gc();
         } else throw c;
      } else if (c=='c') {cout<< 'c'; gc();
      } else throw c;
    }
  };
```

```
  class S { public:
    void parse() {
      if (c=='a') {
        cout<<'a'; gc (); S().parse();
      }
      else A().parse();
    }
  };
public:
  void analyze(){
    try{ gc(); S().parse();
         if (!cin.eof()) throw -1;
    }
    catch(...){cout<< "\nERROR";}
    cout<<endl;
  }
};
char Parser::c;
int main(){  Parser().analyze();}
```

а) Восстановите грамматику G по программе.

б) Не изменяя методы *parse*(), добавьте недостающие описания в классах A и S так, чтобы правильные цепочки переводились данной программой в линейную запись соответствующего дерева вывода. Символу S в линейной записи соответствует пара скобок < и >, обрамляющих выводимую из S цепочку, символу A соответствует пара скобок (и). Например, для дерева 

линейная запись такова: <(*c*)> .

7. Дана грамматика G: $S \to Ac \mid Ab$
$A \to Ac \mid b \mid Sa$

(а) Обоснуйте следующее утверждение: любую автоматную грамматику через детерминизацию диаграммы состояний можно преобразовать в эквивалентную праволинейную, к которой примени́м метод рекурсивного спуска. (б) Преобразуйте заданную грамматику G в соответствии с пунктом (а).

8. Дана КС-грамматика G_a с действиями над глобальными переменными *depth* и *level*. (а) Какой язык задает данная грамматика (какое множество цепочек) ? (б) Построить КС-грамматику G без действий для языка $L(G_a)$, содержащую не более 4-х правил вывода, считая альтернативы.

G_a: $S \to \langle\ depth{=}0;\ level{=}0;\ \rangle\ A\ \langle \textbf{\textit{if}}\ (depth{<}2)\ error();\ \rangle$
$A \to (\ \langle level{+}{+};\ depth = level > depth\ ?\ level\ :\ depth;\ \rangle\ A\ )\ \langle level{-}{-};\ \rangle\ A \mid\ \varepsilon$

9. Может ли быть неоднозначной грамматика, порождающая язык $\{\varepsilon\}$? Обоснуйте ответ.

10. Дан ПОЛИЗ программы на модельном М-языке. (а) Восстановить по ПОЛИЗу тело программы на М-языке. (б) Оптимизируйте ПОЛИЗ по длине, т.е. постройте эквивалентный (печатающий тот же результат) ПОЛИЗ минимальной длины, считая, что вводимое значение b всегда положительное. Операции *mod* (остаток от деления) в М-языке нет, есть деление с отбрасыванием остатка (/), сложение, умножение, вычитание и сравнения. Сокращенный оператор if-then без else есть.

ПОЛИЗ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	&a	7	:=	*&b*	*read*	*a*	*b*	<>	32	!F	*a*	*b*	<	25	!F	*&c*	*b*

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	
:=	*&b*	*a*	:=	*&a*	*c*	:=	*&a*	*a*	*b*	–	:=	6	!	*a*	*write*		

ПОЛИЗ (оптим.)	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	