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.

$S \to A \ \{+ \langle + \rangle A\}$
$A \to B \ \{* \ \langle * \rangle B\}$
$B \to N F$
$N \to - \ M \mid \varepsilon$
$M \to - \ N \mid \varepsilon \ \langle - \rangle$
$F \to a \langle a \rangle \mid \ b \langle b \rangle \mid \mathbf{(} \langle (\rangle S \mathbf{)} \langle \ ) \rangle$

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.

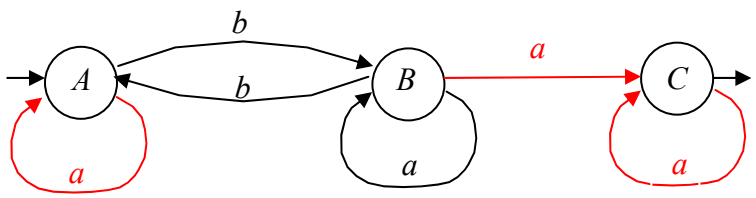
3. Даны названия программных систем: *CVS, SVN, GIT, Make*. Вычеркнуть среди них одно лишнее и дать определение понятию, примерами которого являются все оставшиеся системы.

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 $\mathcal{A} = \langle \{X, Y, Z\}, \{a, b\}, \delta, \{X\}, \{Z\} \rangle$.

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) $\mathcal{A}$ получается из G_L алгоритмом построения конечного автомата по леволинейной грамматике; (в) $\mathcal{A}$ получается из G_R алгоритмом построения конечного автомата по праволинейной грамматике; (г) $\mathcal{A}$ не является детерминированным; (д) количество дуг в $\mathcal{A}$ 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.

	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: ***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