1	2

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 переходов
     // 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 $\mathcal{U}$. *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* 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(лексема)

  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	2

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(лексема)

  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 $\{ D \to Da \mid Ca ; C \to Cb \mid Db \mid \varepsilon \}$. 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 переходов
     // 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.

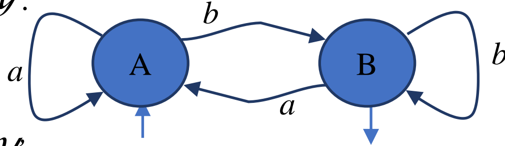
Вариант 1.

1. Ответы:

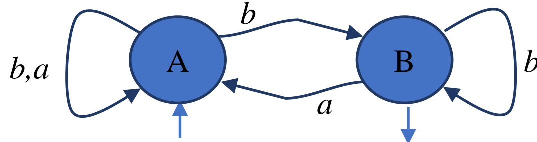
е) B – начальный символ в G_{left}.

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) $P=\{\ S \rightarrow aAbB\ ;\ A \rightarrow aA \mid \varepsilon\ ;\ B \rightarrow b\ \}$

ж) $first(aA) \cap first(\varepsilon) = \varnothing,\ first(A) \cap follow(A) = \varnothing$ $\Rightarrow$ выполняется критерий применимости.

з) Для неоднозначных грамматик нарушается хотя бы один пункт критерия применимости рекурсивного спуска. Из (б) следует, что грамматика G однозначна.

и) $S \rightarrow aAbB \mid abB\ ;\ A \rightarrow aA \mid a\ ;\ B \rightarrow b$

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.

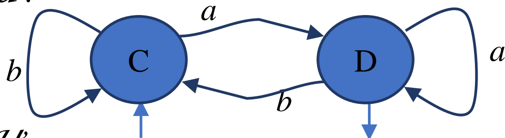
Вариант 2.

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)

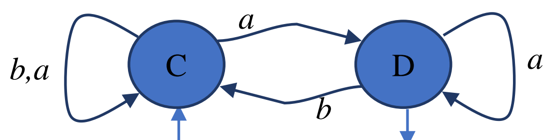
```
void analyze(){
  try{ trans.clear();
      gc();
      S();
      if (c!='#') throw c;
      cout<< "Результат перевода для входной цепочки: ";
      for (list<char>::iterator p=trans.begin();p!=trans.end();p++)
           cout<<*p;

      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.