

## Оглавление

ВВЕДЕНИЕ .....	2
ЦЕЛЬ, ЗАДАЧИ И ОБЪЕМ ДОМАШНЕГО ЗАДАНИЯ.....	3
ТЕОРЕТИЧЕСКАЯ ЧАСТЬ .....	4
1.1. Грамматики языков программирования.....	4
1.2. Лексические анализаторы.....	5
1.3. Синтаксические анализаторы.....	8
1.3.1. Метод рекурсивного спуска для грамматик LR(k) .....	9
1.3.2. Разбор грамматик с предшествованием LL(k) .....	13
ВАРИАНТЫ ЗАДАНИЙ .....	28
ПОРЯДОК ВЫПОЛНЕНИЯ ДОМАШНЕГО ЗАДАНИЯ.....	35
ТРЕБОВАНИЯ К ОТЧЕТУ .....	36
КОНТРОЛЬНЫЕ ВОПРОСЫ .....	37
ЛИТЕРАТУРА .....	40

## **ВВЕДЕНИЕ**

Теоретические основы построения компилирующих программ – неотъемлемая часть знаний и умений современного программиста. Большинству программистов в процессе работы приходится использовать те или иные методы, предлагаемые теорией формальных языков, для анализа синтаксиса небольших языков, встроенных в сложное программное обеспечение. Примером может служить необходимость включения языка описания математических формул при создании программных систем экономического характера. Кроме того несложные формальные языки часто используют в качестве промежуточной формы описания разного рода сложных объектов предметной области и др.

Для углубления понимания теории формальных языков и обретения соответствующих навыков необходимо в процессе обучения самостоятельно описать синтаксис некоторого формального языка, определить тип грамматики по Хомскому, выбрать методы лексического и синтаксического анализа его предложений и разработать программы, реализующие разбор предложений выбранным методом.

Язык реализации программ анализа студент выбирает самостоятельно, также самостоятельно студент разрабатывает интерфейс пользователя и средства его реализации.

## ЦЕЛЬ, ЗАДАЧИ И ОБЪЕМ ДОМАШНЕГО ЗАДАНИЯ

**Целью** домашнего задания № 2 по дисциплине «Машинно-зависимые языки и основы компиляции» является закрепление знаний теоретических основ и основных методов приемов разработки лексических и синтаксических анализаторов регулярных и контекстно-свободных формальных языков.

**Задачами** домашнего задания являются:

- изучение математических основ построения формальных языков;
- применение на практике различных методов разбора предложений формального языка программирования.

**Объем работы:** 8 часов.

## ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

### 1.1. Грамматики языков программирования

Любой язык программирования подчиняется правилам, описанным его грамматикой.

*Формальная грамматика* – это математическая система, определяющая язык посредством порождающих правил – правил продукции. Она определяется как четверка:

$$G = (VT, VN, P, S),$$

где  $VT$  – алфавит языка или множество терминальных (незаменяемых) символов;

$VN$  – множество нетерминальных (заменяемых) символов – вспомогательный алфавит, символы которого обозначают допустимые понятия языка,  $VT \cap VN = \emptyset$ ;

$V = VT \cup VN$  – словарь грамматики;

$P$  – множество порождающих правил – каждое правило состоит из пары строк  $(\alpha, \beta)$ , где  $\alpha \in V^+$  – левая часть правила,  $\beta \in V^*$  – правая часть правила:  $\alpha \rightarrow \beta$ , где строка  $\alpha$  должна содержать хотя бы один нетерминал;

$S \in VN$  – начальный символ – аксиома грамматики.

Для описания синтаксиса языков с бесконечным количеством различных предложений используют рекурсию.

**Пример.** Определим грамматику записи десятичных чисел  $G_0$ :

$VT = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -\}$ ;

$VN = \{<\text{целое}>, <\text{целое без знака}>, <\text{цифра}>, <\text{знак}>\}$ ;

$P = \{<\text{целое}> \rightarrow <\text{знак}><\text{целое без знака}>, <\text{целое}> \rightarrow <\text{целое без знака}>, \\ <\text{целое без знака}> \rightarrow <\text{цифра}><\text{целое без знака}>, // \text{правосторонняя рекурсия} \\ <\text{целое без знака}> \rightarrow <\text{цифра}>, \\ <\text{цифра}> \rightarrow 0, <\text{цифра}> \rightarrow 1, <\text{цифра}> \rightarrow 2, <\text{цифра}> \rightarrow 3, <\text{цифра}> \rightarrow 4, \\ <\text{цифра}> \rightarrow 5, <\text{цифра}> \rightarrow 6, <\text{цифра}> \rightarrow 7, <\text{цифра}> \rightarrow 8, <\text{цифра}> \rightarrow 9, \\ <\text{знак}> \rightarrow +, <\text{знак}> \rightarrow - \}$ ;

$S = <\text{целое}>$ .

Для записи правил продукции обычно используют более компактные и наглядные формы (модели): форму Бэкуса-Наура или синтаксические диаграммы.

Форма Бэкуса-Наура (БНФ) связывает терминальные и нетерминальные символы, используя две операции:  $\langle::=\rangle$  – «можно заменить на»;  $\langle|\rangle$  – «или». Основное достоинство –



группировка правил, определяющих каждый нетерминал. Нетерминалы при этом записываются в угловых скобках. Например, правила продукции грамматики, рассмотренной выше можно записать следующим образом:

$\langle \text{целое} \rangle ::= \langle \text{знак} \rangle \langle \text{целое без знака} \rangle | \langle \text{целое без знака} \rangle,$   
 $\langle \text{целое без знака} \rangle ::= \langle \text{цифра} \rangle \langle \text{целое без знака} \rangle | \langle \text{цифра} \rangle,$   
 $\langle \text{цифра} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9,$   
 $\langle \text{знак} \rangle ::= + | - .$

Формальная грамматика, таким образом, определяет правила определения допустимых конструкций языка, т. е. его синтаксис.

Все грамматики относятся к определенным типам в соответствии с классификаций Хомского. Нас интересуют грамматики типа 2 – контекстно-свободные (КС) и типа 3 – регулярные, поскольку соответствующие языки сравнительно легко распознавать.

## 1.2. Лексические анализаторы

Первый этап процесса компиляции текста, написанного на формальном языке программирования, называется *лексическим анализом*. При выполнении лексического анализа текст разбивают на «предложения» - операторы языка, а операторы – на «слова», которые применительно к компиляции называют *лексемами*. Для разделения лексем в языке могут использоваться специальные разделители, например пробелы. Однако разделителями могут служить и служебные символы, например запятые.

Программу, выполняющую лексический анализ, называют соответственно *лексическим анализатором* или *сканером*. Сканер выполняет преобразование исходного текста в строку однородных символов. Каждый символ результирующей строки – *токен* соответствует слову языка – лексеме и характеризуется набором атрибутов, таких как тип, адрес и т. п., поэтому строку токенов часто представляют таблицей, строка которой соответствует одному токenu.

Термин «лексема» обозначает относительно простое понятие языка. Всего существует 2 типа лексем:

а) лексемы, соответствующие символам алфавита языка, такие как «Служебные слова» и «Служебные символы»;

б) лексемы, соответствующие базовым понятиям языка, такие как «Идентификатор» и «Литерал».

Для поиска служебных слов и символов можно использовать обычное сравнение, однако следует учитывать, что служебное слово должно иметь пробел перед ним и после

него, т.е. оно не должно являться частью идентификатора. Идентификаторы и литералы должны описываться специальной синтаксической диаграммой и для их распознавания можно использовать конечные автоматы.

Идентификатор – сравнительно простая лексема, поэтому его можно распознавать как с использованием конечного автомата, так и без него, просто выделяя слово до ближайшего разделителя.

**Пример 2.** Разработать функцию распознавания идентификатора в операторе. Функция должна проверять, является ли следующая лексема программы идентификатором и, если является, то строить идентификатор и возвращать в качестве результата true.

1 вариант – функция просто проверяет фрагмент программы до ближайшего разделителя на наличие «запрещенных» символов. Параметры – строка и множество разделителей:

```
Const Razd:setofChar=[' ', '+', '-', '*', '/', ')'];
```

Текст функции:

```
Function Id1(Var St:shortstring;Razd:setofChar):boolean;
  Var S:shortString;
  Begin
    Probel(St); {процедура удаления пробелов}
    S:='';
    if St[1] in ['A'..'Z','a'..'z'] then
      Begin
        S:=S+St[1]; Delete(St,1,1);
        While (St<>'') and (St[1] in ['A'..'Z','a'..'z'])
          or (St[1] in ['0'..'9']) do
          Begin
            S:=S+St[1]; Delete(St,1,1);
          End;
        if (St='') or (St[1] in Razd) then
          Begin
            Result:=true; WriteLn('Identify=',S);
          End
        else
          Begin
            Result:=false;
```

```

        WriteLn('Wrong symbol *', St[1], '*');
    End;
End
else
    Begin
        Result:=false;
        WriteLn('Identifier waits...', St);
    End;
End;

```

2 вариант – функция реализует конечный автомат, построенный по синтаксической диаграмме на рисунке 1.

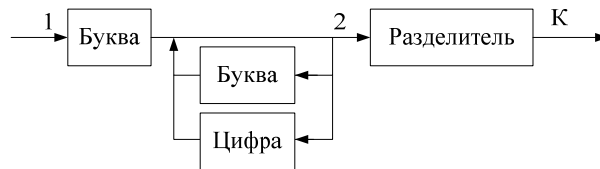


Рисунок 1 – Синтаксическая диаграмма лексемы Идентификатор

Таблица автомата, построенная по синтаксической диаграмме, учитывает возможное присутствие неразрешенных символов:

Состояние	Буква	Цифра	Разделитель	Другой символ
1	2	Error	Error	Error
2	2	2	10	Error

Текст программы выглядит следующим образом:

```

Function Id2(Var St:shortstring; Razd:setofChar):boolean;
Var S:shortString; State,Ind,Col:byte;
Const TableId:array[1..2,1..4] of Byte=
        ((2,0,0,0),(2,2,10,0));
Begin
    Probel(St); {процедура удаления пробелов}
    State:=1;
    S:='';
    while (State<>0) and (State<>10) and (Length(St)<>0) do
    begin

```

```

if St[1] in ['A'..'Z','a'..'z'] then Col:=1
else if St[1] in ['0'..'9'] then Col:=2
      else if (St[1] in Razd) then Col:=3
            else Col:=4;
State:=TableId[State,Col];
if (State<>0) and (State<>10) then
begin
    S:=S+St[1];
    Delete(St,1,1);
end;
end;
if length(st)=0 then State:=10;
if (State=10) and (S<>'') then
    Begin
        Result:=true;  WriteLn('Identify=',S);
    End
else
    if (State=0) then
        Begin
            Result:=false;
            WriteLn('Wrong symbol *',St[1],'*');
        End
    else
        Begin
            Result:=false;
            WriteLn('Identifier waits...', St);
        End;
End;
End;

```

Несмотря на то, что оба варианта функции работают, второй вариант предпочтителен, поскольку реализует более общий метод.

### 1.3. Синтаксические анализаторы

Второй этап работы компилятора называют *синтаксическим анализом*.

[Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина «Лексические и синтаксические анализаторы»

Синтаксический анализ – процесс распознавания конструкций языка в строке токенов. Главным результатом, помимо распознавания заданной конструкции, является информация об ошибках в выражениях, операторах и описаниях программы.

Способ построения синтаксического анализатора определяется типом грамматики языка:

- для регулярных грамматик используют конечные автоматы;
- для КС грамматик – автоматы с магазинной памятью.

Построение конечного автомата особой сложности не составляет. По синтаксической диаграмме строится таблица, а затем пишется универсальная программа, которой передается таблица автомата. От количества конструкций сложность программы практически не возрастает.

Построение же автомата с магазинной памятью для языка, включающего десятки конструкций – достаточно трудоемко, поэтому на практике либо используют метод рекурсивного спуска, разработанный для LL(k)-грамматик, либо строят программу с применением свойств грамматик с предшествованием, называемых LR(k)-грамматиками.

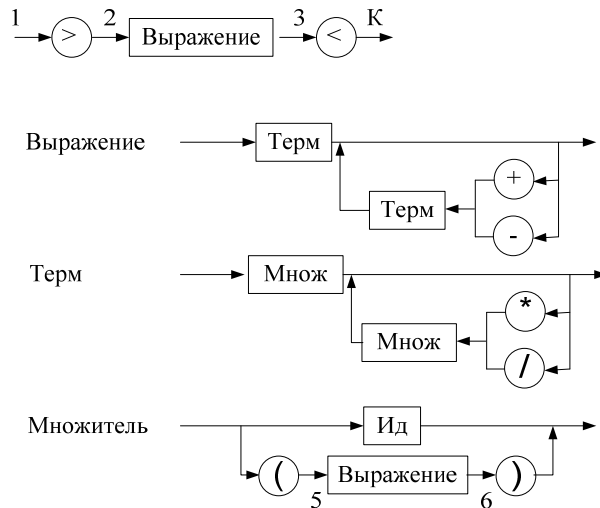
### ***1.3.1. Метод рекурсивного спуска для грамматик LR(k)***

Метод рекурсивного спуска предполагает, что сначала следует построить синтаксические диаграммы всех разбираемых конструкций, потом по диаграммам разработать функции проверки конструкций, а затем составить основную программу, начинающую вызов функций с функции, реализующей аксиому языка.

При этом отдельно сканер, как правило, не строят. Функции, сканирующие исходный текст, встраивают в текст анализатора, передавая им управление, если необходимо проверить правильность написания лексем.

**Пример 3.** Разработать программу, проверяющую правильность записи выражения.

Вначале строим синтаксические диаграммы (рисунок 2).



**Рисунок 2** – Синтаксические диаграммы проверки правильности записи Выражения

Текст программы:

```

program Compiler;
{$APPTYPE CONSOLE}
uses SysUtils;
Type SetofChar=set of AnsiChar;
Const Razd:setofChar=[' ', '+', '-', '*', '/', ')'];
Var St:shortstring;    R:boolean;

Function Culc(Var St:shortstring;
              Razd:setofChar):boolean;forward;

Procedure Error(St:shortstring); {вывод сообщений об ошибках}
Begin      WriteLn('Error *** ', st, ' ***'); End;

Procedure Probel(Var St:shortstring); {удаление пробелов}
Begin      While (St<>'') and (St[1]=' ') do Delete(St,1,1); End;

{распознаватель идентификатора на конечном автомате}
Function Id(Var St:shortstring;Razd:setofChar):boolean;
... {текст функции приведен выше в разделе 1.2}

Function Mult(Var St:shortstring;Razd:setofChar):boolean;

```

[Оглавление](#)

```

Var R:boolean;
Begin
  Probel(St);
  if St[1]='(' then
    begin
      Delete(St,1,1); Probel(St);
      R:=Culc(St,Razd);
      Probel(St);
      if R and (St[1]=')') then
        Delete(St,1,1) else Error(St);
      end
    else R:=Id(St,Razd);
    Mult:=R;
  End;

```

```

Function Term(Var St:shortstring;Razd:setofChar):boolean;
Var R:boolean;
Begin
  R:=Mult(St,Razd);
  if R then
    begin
      Probel(St);
      While ((St[1]='*') or (St[1]='/')) and R do
        begin
          Delete(St,1,1);
          R:=Mult(St,Razd);
        end;
      end;
    Term:=R;
  End;

```

```

Function Culc(Var St:shortstring;Razd:setofChar):boolean;
Var R:boolean;
Begin
  R:=Term(St,Razd);

```

```

    if R then
        begin
            Probel(St);
            While ((St[1]='+') or (St[1]='-')) and R do
                begin
                    Delete(St,1,1);
                    R:=Term(St,Razd);
                end;
            end;
            Culc:=R;
        End;

Begin
    Writeln('Input Strings:'); Readln(St);
    R:=true;
    While (St<>'end') and R do
        Begin
            R:=Culc(St,Razd);
            if R and (length(st)=0) then Writeln('Yes')
                else Writeln('No');
            Writeln('Input Strings:');
            Readln(St);
            R:=true;
        End;
        Writeln('Press Enter');
        Readln;
    End.

```

Результат работы программы:

```

Input Strings:
tyy+(hjh-hj)*hj
Identify=tyy
Identify=hjh
Identify=hj

```

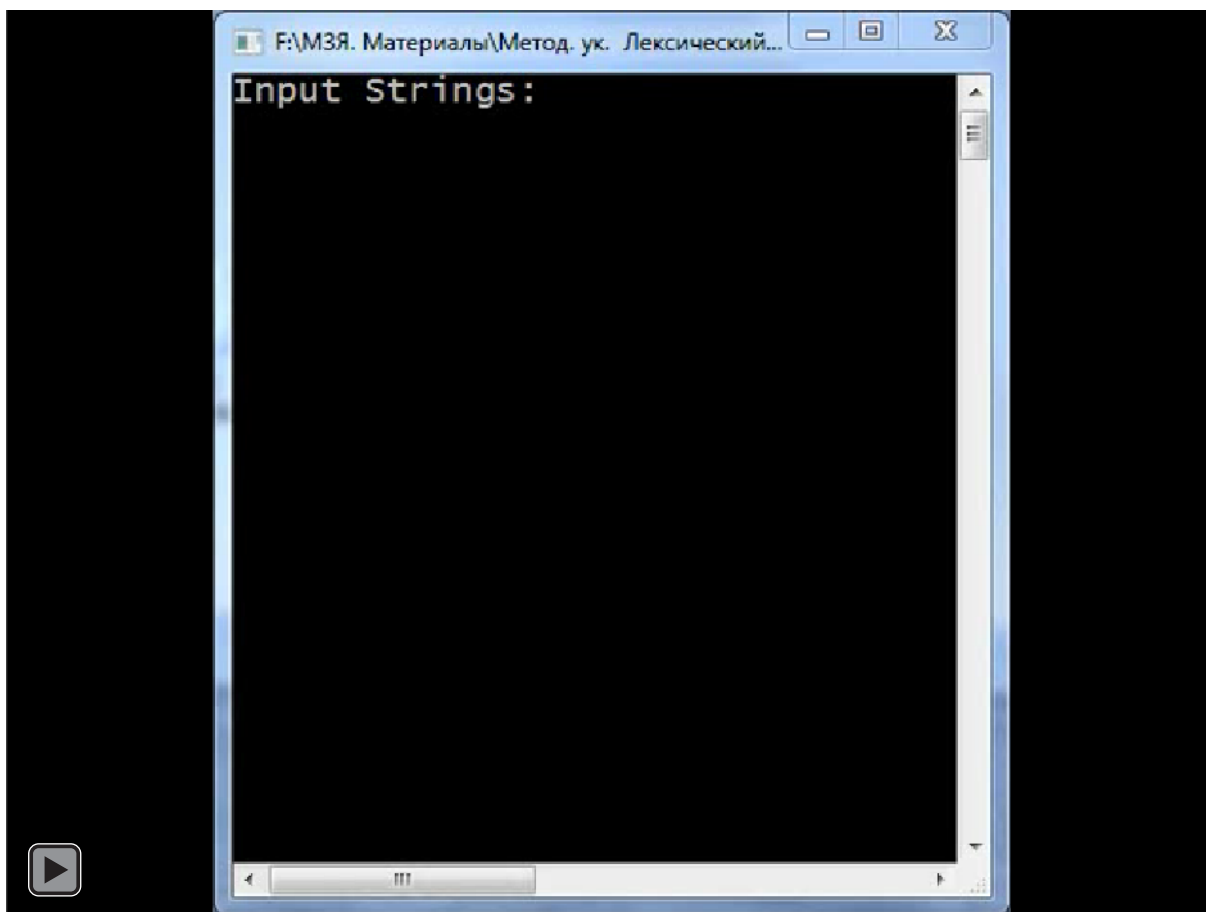


```

Identify=hj
Yes
Input Strings:
end

```

Пояснения к результатам выполнения программы представлены в учебном фильме:



### 1.3.2. Разбор грамматик с предшествованием $LL(k)$

Разбор грамматики по правилам грамматики с предшествованием предполагает, что сначала выполняется лексический анализ программы, результатом которого является *строка токенов*. Затем уже строка токенов анализируется с использованием стекового метода или польской записи.

**Пример 3** (второй вариант решения).

Выполним синтаксический анализ Выражения с использованием стекового метода.

Сначала строка будет обработана сканером. В результирующей строке токенов будем использовать следующие обозначения:

[Оглавление](#)

I – идентификатор; +, -, \*, / – символы операций;

@ - специальный символ, используемый в качестве «пустого» операнда для символа «(», что позволит в любом случае выбирать из строки токенов при синтаксическом анализе по два символа.

Строка токенов разбирается в соответствии с таблицей предшествования:

	+	*	(	)	◀
▶	<.	<.	<.	?	Выход
+	.>	<.	<.	.>	.>
*	.>	.>	<.	.>	.>
(	<.	<.	<.	)	?
)	.>	.>	?	.>	.>

Обозначения:

? – ошибка;

< - начало основы;

> - конец основы;

() – скобки – принадлежат одной основе;

▶ - начало выражения;

◀ - конец выражения.

В программе таблица кодируется следующим образом:

1 – начало основы;

2 – конец основы;

3 – скобки;

4 – нормальное завершение;

10 – ошибка.

Кроме этого в программе в таблицу добавлен столбец справа, фиксирующий ошибку для ошибочно появившейся «другой» операции, и удалена последняя строка, поскольку в соответствии с алгоритмом обработки закрывающая скобка никогда не попадает в стек.

В функцию, реализующую стековый метод, встроены три процедуры, выполняющие свертку по «концу основы», перенос по «началу основы» и обработку ситуации «принадлежат одной основе».

Текст программы:

```
program Compiler3;
{$APPTYPE CONSOLE}
uses SysUtils;
Type SetofChar=set of AnsiChar;
    Troyki=array[1..10] of shortstring;
Const Razd:setofChar=[' ', '+', '-', '*', '/', '(', ')'];
```

```
Var St,StS:shortstring;  Comands:Troyki;
    R:boolean;
```

```
Function Scanner(St:shortstring;Razd:setofChar;Var
StS:shortstring):boolean;forward;
```

```
Function Stack_metod(St:shortstring;Var
Comands:Troyki):boolean;forward;
```

```
Procedure Error(St:shortstring); {вывод сообщений об ошибках}
Begin      WriteLn('Error *** ', st, ' ***');  End;
```

```
Procedure Probel(Var St:shortstring); {удаление пробелов}
Begin      While (St<>'') and (St[1]=' ') do Delete(St,1,1);  End;
```

```
Function Id(Var St:shortstring;Razd:setofChar):boolean;
    {телo функции описано в разделе 1.2}
```

```
Function Scanner(St:shortstring;Razd:setofChar;
                Var StS:shortstring):boolean;
```

```
Var R:boolean;
Begin
    R:=true;
    StS:='';
    Probel(St);
    while (length(St)<>0) and R do
    begin
        if not (St[1] in Razd) then
        begin
            R:=Id(St,Razd);
            if R then StS:=StS+'I';
        end
        else
        begin
            if St[1]='(' then StS:= StS+'@';
```

```

        StS:=StS+St[1];
        Delete(St,1,1);
    end;
    Probel(St);
end;
WriteLn('After Scan:',StS);
Result:=R;
End;

Function Stack_metod(St:shortstring;
                    Var Comands:Troyki):boolean;
Const TablePred:array[0..3,1..6] of byte=
    ((1, 1, 1, 10, 4, 10),
     (2, 1, 1, 2, 2, 10),
     (2, 2, 1, 2, 2, 10),
     (1, 1, 1, 3, 10, 10));
Var Stack:shortString;
    i,i1,IndStr,IndCol:byte;
    Konec:boolean;
Procedure Perenos(Var St,Stack:shortstring);
Begin
    Stack:=Stack+Copy(St,1,2);
    Delete(St,1,2);
end;
Procedure Svertka(Var i:byte;Var St,Stack:shortstring;
                 Var Comands:Troyki);
begin
    i:=i+1;
    Comands[i]:=Copy(Stack,length(Stack)-1,2);
    Delete(Stack,length(Stack)-1,2);
    Comands[i]:=Comands[i]+St[1];
    St[1]:='R';
end;
Procedure Odnosnova(Var St,Stack:shortstring);
Begin

```

```

Delete (Stack,length (Stack),1);
if Stack[length (Stack)]='@' then
    Delete (Stack,length (Stack),1);
Delete (St,2,1);
end;
Begin
    i:=0;
    Konec:=false;
    Result:=false;
    Stack:='>';
    St:=St+'<';
    while not Konec do
    begin
        case Stack[length (Stack)] of
            '+','-' : IndStr:=1;
            '*','/' : IndStr:=2;
            '(' : IndStr:=3;
            '>' : IndStr:=0;
            else IndStr:=0;
        end;
        case St[2] of
            '+','-' : IndCol:=1;
            '*','/' : IndCol:=2;
            '(' : IndCol:=3;
            ')' : IndCol:=4;
            '<' : IndCol:=5;
            else IndCol:=6;
        end;
        case TablePred[IndStr,IndCol] of
            1: Perenos (St,Stack);
            2: Svertka (i,St,Stack,Comands);
            3: Odnosnova (St,Stack);
            4: begin
                Result:=true;
                Konec:=true;
            end;
        end;
    end;
end;

```

```

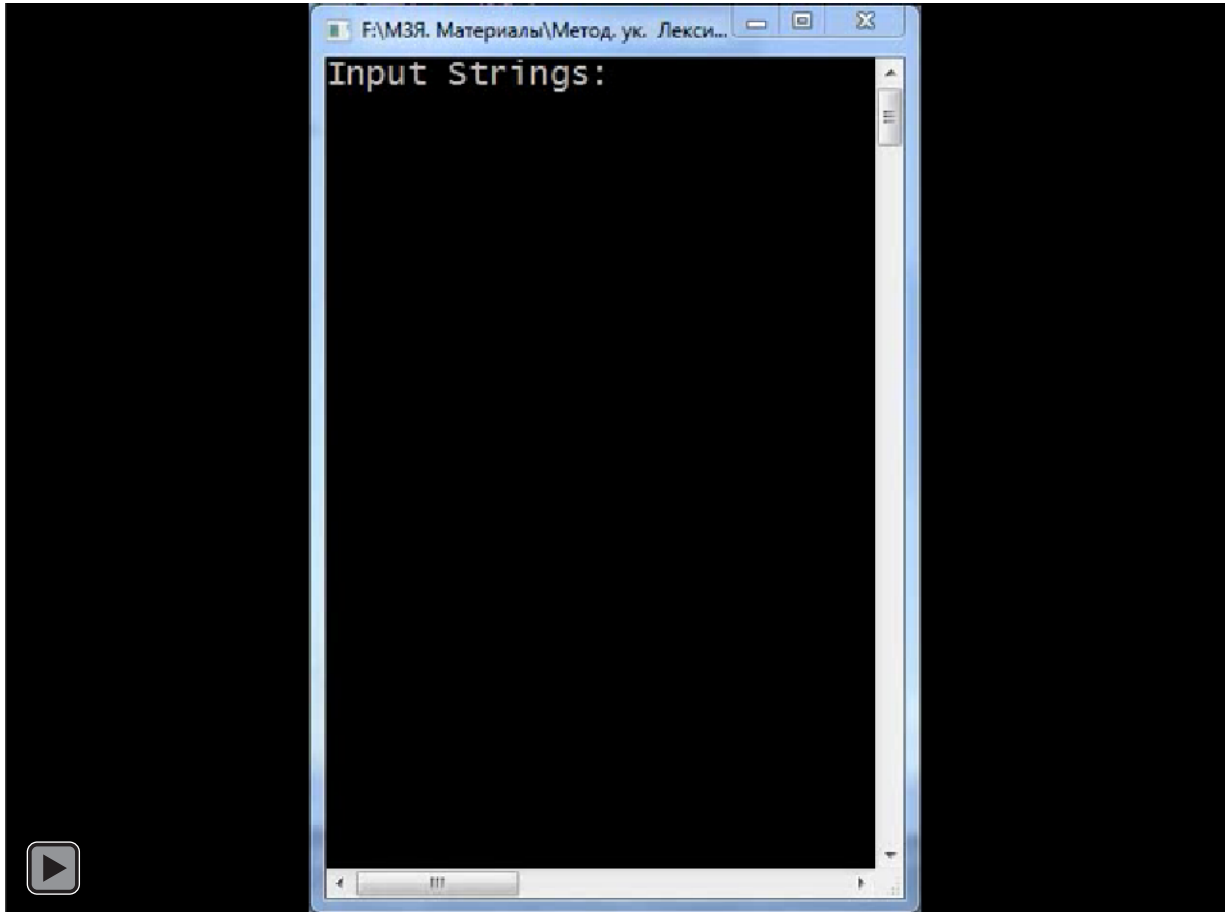
        For i1:=1 to i do
            WriteLn('Comands:',Comands[i1]);
        end;
    10: begin
        Konec:=true;
        For i1:=1 to i do
            WriteLn('Comands:',Comands[i1]);
            WriteLn('St=',St);
        end;
    else
        begin
            Konec:=true;
            For i1:=1 to i do
                WriteLn('Comands:',Comands[i1]);
                WriteLn('St=',St);
            end;
        end;
    end;
end;
end;

Begin
    Writeln('Input Strings:'); Readln(St);
    R:=true;
    While (St<>'end') and R do
        Begin
            R:=Scanner(St,Razd,StS);
            if R then R:=Stack_metod(StS,Comands);
            if R then Writeln('Yes')
                else Writeln('No');
            Writeln('Input Strings:'); Readln(St);
            R:=true;
        End;
        writeln('Press Enter');
        readln;
    End.

```

Результат работы программы представлен ниже (полужирным шрифтом выделена строка, введенная пользователем).

Пояснения к результатам выполнения программы представлены в учебном фильме:



Input Strings:

**Qr34+ghj\*(hj+yi)**

Identify=Qr34

Identify=ghj

Identify=hj

Identify=yi

After Scan:I+I\*@(I+I)

Comands:I+I

Comands:I\*R

Comands:I+R

Yes

Input Strings:

**end**

[Оглавление](#)

Поскольку приведенный пример не показывает всех особенностей использования стекового метода, рассмотрим еще один, более сложный пример.

**Пример 4.** Разработать программу, осуществляющую лексический анализ идентификаторов и служебных слов, а также синтаксический анализ сравнений (не более одной операции сравнения вида =, <>, >, <,>=,<=), выражений с операциями +, -, \*, / и скобками, операторов условной передачи управления и присваивания в синтаксисе языка Паскаль. Например:

```
if aaaa>vvvv then j:=hhhh else if h then ffff:=hhhh+(ppp+yyy);
```

Аналогично предыдущему примеру при составлении программы будем использовать сканер, строящий строку токенов (каждый токен длиной 2 символа), и анализатор, который разбирает эту строку.

Токены:

V\_ – идентификатор - операнд;

@@ – пустой операнд – дополнительный токен, который позволяет при разборе считывать строго по два токена <операнд-оператор>;

if – служебное слово if;

th – служебное слово then;

el – служебное слово else;

>\_, <\_, =\_, <>\_, >=, <= – операции сравнения;

+\_, -\_, \*\_\_, /\_, (, )\_ – операторы выражения;

:= - служебное слово «присвоить»;

\_; – конец оператора.

Для примера, приведенного в задании, результат работы сканера должен выглядеть так:

@@	if	V_	>_	V_	th	V_	:=	V_	el	@@	if	V_	th
----	----	----	----	----	----	----	----	----	----	----	----	----	----

V_	:=	V_	*_	@@	(	V_	+_	V_	)	;	_
----	----	----	----	----	---	----	----	----	---	---	---

Разделители:

```
Const Razd:setofChar=
```

```
[' ', '+', '-', '*', '/', '(', ')', ':', ';', '<', '>', '='];
```

Функция-сканер:

```
Function Scanner(St:shortstring;Razd:setofChar;
```

[Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина «Лексические и синтаксические анализаторы»



```

Var StS:shortstring):boolean;
Const SlSl:array[1..3] of shortString=('if','then','else');
Var R:boolean; i,k:byte; Stl:shortstring;
Begin
  R:=true;
  StS:='';
  Probel(St);
  while (length(St)<>0) and R do
  begin
    if not (St[1] in Razd) then
    begin
      k:=Pos(' ',St);
      if k<>0 then
      begin
        Stl:=Copy(St,1,k-1);
        i:=1;
        while (Stl<>SlSl[i]) and (i<4) do i:=i+1;
        if i<>4 then
        begin
          WriteLn('Slugeb slovo ',Stl);
          Delete(St,1,k-1);
          if (Stl[1]='I') or (Stl[1]='i') then
            StS:=StS+'@@';
          StS:=StS+Copy(Stl,1,2);
          R:=true;
        end
      else
      begin
        R:=Id(St,Razd);
        if R then StS:=StS+'V ';
      end;
    end
  else
  begin
    R:=Id(St,Razd);

```

```

        if R then StS:=StS+'V ';
    end
end
else
    begin
        if St[1]='(' then Sts:=StS+'@@';
        Sts:=StS+St[1];
        Delete(St,1,1);
        if St[1] in ['=','>'] then
            begin
                StS:=StS+St[1];
                Delete(St,1,1);
            end
        else StS:=StS+' ';
        WriteLn('Slugeb simbol ',
                Copy(StS,length(StS)-1,2));
    end;
    Probel(St);
end;
WriteLn('After Scan:',StS);
Result:=R;
End;

```

Функция разбора реализует таблицу предшествования, разработанную по синтаксическим диаграммам оператора if, упрощенного сравнения и оператора присваивания, в правой части которого может быть задано выражение:

	=	+	*	(	)	:=	If	Th	El	;	??
#	E	E	E	E	E	<.	<.	E	E	K	E
=	E	E	E	E	E	E	E	>.	E	E	E
+	E	>.	<.	<.	>.	E	E	E	E	>.	E
*	E	>.	>.	<.	>.	E	E	E	E	>	E
(	E	<.	<.	<.	)	E	E	E	E	E	E
:=	E	<.	<.	<.	E	E	E	E	>.	>.	E
If	<.	E	E	E	E	E	E	=.	E	E	E
Th	E	E	E	E	E	<.	<.	E	=.	>.	E
El	E	E	E	E	E	<.	<.	E	>.	>.	E

При реализации использовано следующее кодирование: 1 – начало основы; 2 – середина основы; 3 – конец основы; 4 – скобки; 5 – выход; 50 – ошибка. Функция включает 4 подпрограммы, реализующие соответствующие операции.

Результат работы функции – последовательность образов команд (по типу троек арифметического выражения), где операндами являются:

Op – результат оператора присваивания;

OI – результат вложенного условного оператора;

L – результат простейшего сравнения;

C – результат арифметического выражения;

V – идентификатор.

```
Function Predshest(St:shortstring;
                    Var Comands:Troyki):boolean;

Const TablePred:array[0..8,1..11] of byte=
    { 1  2  3  4  5  6  7  8  9 10 11
      =  +  *  (  )  := If Th El  ; ?? }
{0 #   } ((50,50,50,50,50,01,01,50,50,05,50),
{1 =   } (50,50,50,50,50,50,50,03,50,50,50), // все сравнения
{2 +   } (50,03,01,01,03,50,50,50,50,03,50), // арифм. выр.
{3 *   } (50,03,03,01,03,50,50,50,50,03,50), // арифм. выр.
{4 (   } (50,01,01,01,04,50,50,50,50,50,50), // арифм. выр.
{5 :=  } (50,01,01,01,50,50,50,50,03,03,50), // присваивание
{6 If  } (01,50,50,50,50,50,50,02,50,50,50), // начало ветв.
{7 Th  } (50,50,50,50,50,01,01,50,02,03,50), // то
```

```

{8 El } (50,50,50,50,50,01,01,50,03,03,50)); // иначе

Var Stack:shortString;
    i,i1,IndStr,IndCol:byte;
    Konec:boolean;

Procedure Begin01(Var St,Stack:shortstring);
Begin
    Stack:=Stack+'<.';
    Stack:=Stack+Copy(St,1,4);
    Delete(St,1,4);
end;

Procedure End03(Var i:byte; Var St,Stack:shortstring;
                Var Comands:Troyki);

Var k,k1:byte;
begin
    i:=i+1;
    k:=0;
    k1:=length(Stack)-1;
    While k=0 do
    begin
        if copy(Stack,k1,2) = '<.' then k:=k1;
        k1:=k1-4;
    end;
    if Stack[k+2]='@' then
        Comands[i]:=Copy(Stack,k+4,length(Stack)-k-3)
    else Comands[i]:=Copy(Stack,k+2,length(Stack)-k-1);
    Delete(Stack,k,length(Stack)-k+1);
    if Stack[length(Stack)]='@' then
        Delete(Stack,length(Stack)-1,2);
    Comands[i]:=Comands[i]+copy(St,1,2);
    Delete(St,1,2);
    // Здесь должна быть проверка
    if (Comands[i][1]='i') or (Comands[1]='I') then

```

```

        Insert('OI',St,1) // вложенный оператор 'if'
    else if Comands[i][3]in ['+', '-', '*'] then
        Insert('C ',St,1) // выражение
    else if Comands[i][3]= ':' then
        Insert('Op',St,1) // оператор присваивания
    else if Comands[i][3]in ['<', '>', '='] then
        Insert('L ',St,1); // логическое выражение
end;
```

```

Procedure Middle02 (Var St, Stack:shortstring);
Begin
    Stack:=Stack+Copy(St,1,4);
    Delete(St,1,4);
end;
```

```

Procedure Skobki (Var St, Stack:shortstring);
begin
    Delete(Stack,length(Stack)-1,2);
    if Stack[length(Stack)]='@' then
        Delete(Stack,length(Stack)-1,2);
    if Stack[length(Stack)-1]='<' then
        Delete(Stack,length(Stack)-1,2);
    Delete(St,2,2);
end;
```

```

Begin
    i:=0;
    Konec:=false;
    Result:=false;
    Stack:='# ';
    while not Konec do
    begin
        case Stack[length(Stack)-1] of
            '=', '>', '<': IndStr:=1;
            '+', '-': IndStr:=2;
```

```

'*','/':      IndStr:=3;
'(': IndStr:=4;
':': IndStr:=5;
'I','i': IndStr:=6;
'T','t': IndStr:=7;
'E','e': IndStr:=8;
'#': IndStr:=0;
else IndStr:=0;
end;
case St[3] of
  '=', '>', '<':  IndCol:=1;
  '+', '-':      IndCol:=2;
  '*', '/':      IndCol:=3;
  '(': IndCol:=4;
  ')': IndCol:=5;
  ':': IndCol:=6;
  'I','i': IndCol:=7;
  'T','t': IndCol:=8;
  'E','e': IndCol:=9;
  ';': IndCol:=10;
  else IndStr:=11;
end;
case TablePred[IndStr,IndCol] of
  1: Begin01(St,Stack);
  2: Middle02(St,Stack);
  3: End03(i,St,Stack,Comands);
  4: Skobki(St,Stack);
  5: begin
      Result:=true;
      Konec:=true;
      For i1:=1 to i do
          WriteLn('Comands:',Comands[i1]);
      end;
  50: begin
      Konec:=true;

```

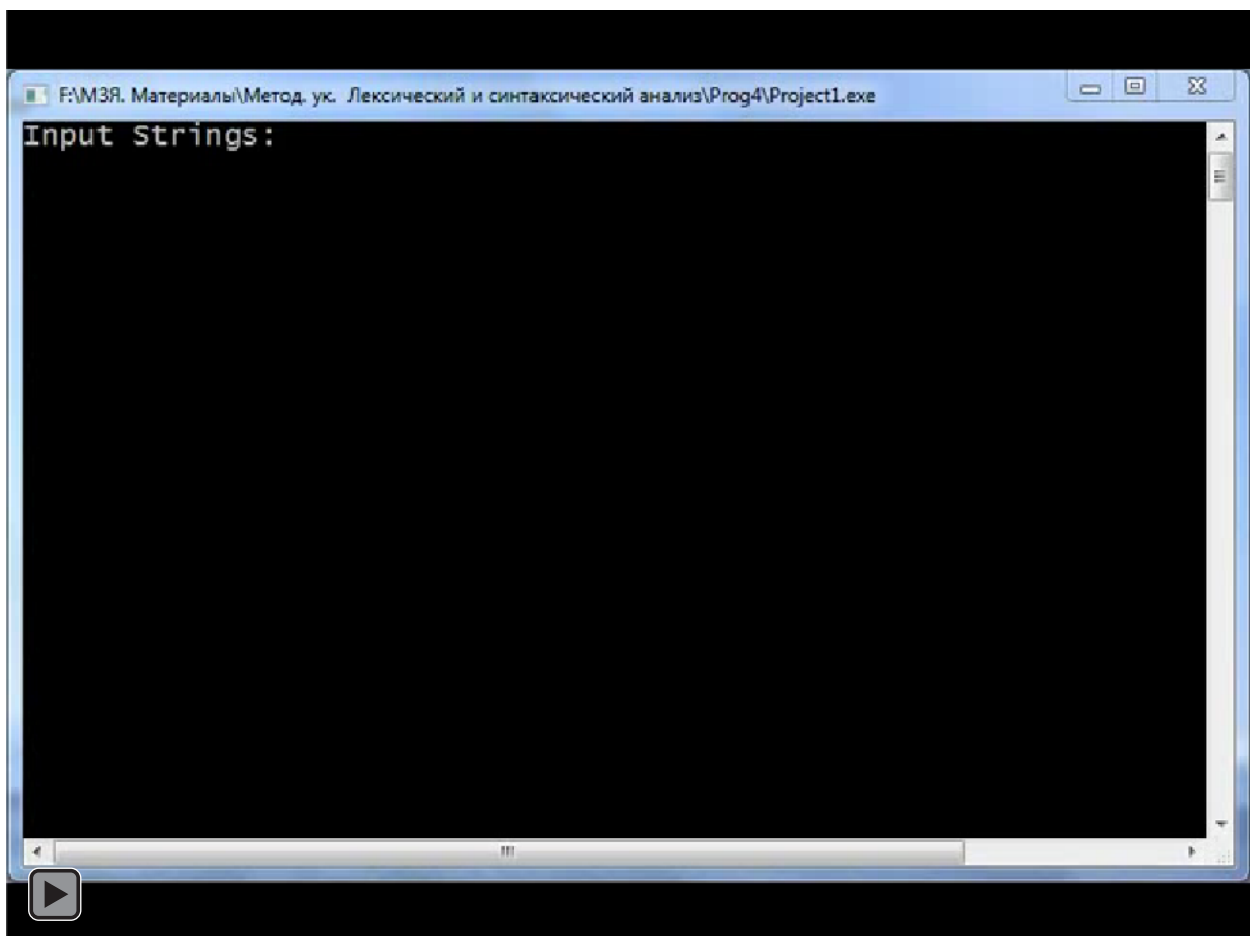
```

        For i1:=1 to i do
            WriteLn('Comands:', Comands[i1]);
            WriteLn('St=', St);
        end;
    end;
end;
end;
end;

```

Результат выполнения программы приведен ниже.

Пояснения к результатам выполнения программы представлены в учебном фильме:



Input Strings:

**if aaaa>vvvv then j:=hhhh else if h then ffff:=hhh\*(ppp+yyy);**

Slugeb slovo if

Identify=aaaa

Slugeb simbol >

[Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина «Лексические и синтаксические анализаторы»

```

Identify=vvvv
Slugeb slovo then
Identify=j
Slugeb simbol :=
Identify=hhhh
Slugeb slovo else
Slugeb slovo if
Identify=h
Slugeb slovo then
Identify=ffff
Slugeb simbol :=
Identify=hhh
Slugeb simbol +
Slugeb simbol (
Identify=ppp
Slugeb simbol +
Identify=yyy
Slugeb simbol )
Slugeb simbol ;
After Scan:@@ifV > V thV :=V el@@ifV thV :=V * @@( V + V ) ;
Comands:V > V // сравнение
Comands:V :=V // присваивание в ветви «да» внешнего if
Comands:V + V // сложение
Comands:V * C // умножение
Comands:V :=C // присваивание в ветви «да» вложенного if
Comands:ifV thOp // вложенный if (без else)
Comands:ifL thOpelOI // внешний if
Yes
Input Strings: end

```

## ВАРИАНТЫ ЗАДАНИЙ

**Вариант 1.** Паскаль. Разработать грамматику и распознаватель выражений,

[Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина «Лексические и синтаксические анализаторы»



оперирующих вещественными числами в формате с фиксированной точкой. Например:

**-34.3456 + 0.56\* 0.7989**

**Вариант 2.** Паскаль. Разработать грамматику и распознаватель имен переменных. Предусмотреть: простые переменные и переменные – элементы массива. Учесть, что индексы элементов массива могут быть выражениями.. Например, **A47** – простая переменная, **Brk[(k1+r)/u, 6]** – элемент массива.

**Вариант 3.** Паскаль. Разработать грамматику и распознаватель имени процедуры или функции. Считать, что только параметры только стандартных (скалярных) типов. Например:

**Procedure as3(Var a\_34,bjt:real; car:integer; const ddd:byte);**

**Function hjhk(Var k:integer):Char;**

**Вариант 4.** Паскаль. Разработать грамматику и распознаватель описания типизированных констант – массивов целых чисел. Предусмотреть задание размерности массива в виде диапазона литералами целого типа. Семантические проверки соответствия количества уровней скобок размерности массива и количества констант размерности массива не выполнять. Например:

**Const A7867:array[1..3,-5..-4] of integer =((6, -6),(59,0),(89,30));**

**Вариант 5.** Паскаль. Разработать грамматику и распознаватель описания типизированных констант – множеств с элементами символьного типа. Тип может быть задан как Char или диапазон типа Char. Конструктор может содержать диапазоны и константы символьного типа. Например:

**Const A7867:set of Char = ['h'..'z','a']; vbg:set of 'a'..'z' = [];**

**Вариант 6.** Паскаль. Разработать грамматику и распознаватель описания языка, включающего операторы присваивания и счетного цикла. Считать, что границы изменения переменной цикла могут быть заданы только целочисленными константами, язык оперирует только простыми переменными и целыми числами, тело цикла содержит не более одного оператора, а оператор присваивания в правой части содержит только идентификаторы или целые константы. Например:

**For i:=1 to 10 do for jj:= 20 downto 10 do a:=5;**

**Вариант 7.** Паскаль. Разработать грамматику и распознаватель описания языка,

включающего оператор присваивания и оператор Case. Считать, что выбор происходит по значению переменной целого типа, альтернативы определяются списком констант целого типа, тело альтернативы содержит не более одного оператора, а оператор присваивания в правой части содержит только идентификаторы или целые константы. Например:

```

Case Index of
1,2: a:=5;
3:Case jj of
    -6:  b23:=3;
    1,7: b23:=5
end
end;

```

**Вариант 8.** Паскаль. Разработать грамматику и распознаватель описания языка, включающего оператор условной передачи управления и оператор присваивания. Считать, что условие – значение переменной логического типа, тело альтернативы содержит не более одного оператора, а оператор присваивания в правой части содержит только идентификаторы или целые константы. Например:

```

if d then gyu:=5 else if r then u:=-78 else gju:=ty;

```

**Вариант 9.** Паскаль. Разработать грамматику и распознаватель описания типизированных констант – записей с полями символьных (Char) и целых (Integer) типов. Конструктор может содержать литералы целого и символьного типов. Семантические проверки не выполнять. Например:

```

Const uuu: record x,y:integer; ch:char; end = (x:6,y:8,ch:'u');

```

**Вариант 10.** Паскаль. Разработать грамматику и распознаватель описания языка, включающего оператор цикла-пока и оператор присваивания. Считать, что условие – значение переменной логического типа, тело цикла содержит не более одного оператора, а оператор присваивания в правой части содержит только идентификаторы или целые константы. Например:

```

while d do while ii do gyu:=5;

```

**Вариант 11.** Паскаль. Разработать грамматику и распознаватель языка описаний структурных типов данных, включающего описание массивов, строк, множеств и записей без вариантов. Предусмотреть следующие типы индексов: Char, Byte. Предусмотреть следующие типы элементов: Real, integer, Char, Byte. Например:

```

Var aer:array[Char, Byte] of real;
    se:record I,k:byte; h:array[Byte] of char; end;
    r:set of Char;
    s:string[67];

```

**Вариант 12.** Паскаль. Разработать грамматику и распознаватель языка описаний записей с вариантами. Предусмотреть следующие типы элементов: Real, integer, Char, Byte. Например:

```

Var se:record I,k:byte; case f of 1:(h:integer); 2:(ch:char;) end;

```

**Вариант 13.** Паскаль. Разработать грамматику и распознаватель языка описаний функций. Предусмотреть передачу параметров только по значению, операторы присваивания, в правой части которых используются переменные и константы целого типа. Учесть, что допустимо описание вложенных функций.

```

Function DD(y,u:integer):integer;
Var I,j:integer;
Begin g:=4; uu:=y; End;

```

**Вариант 14.** C++. Разработать грамматику и распознаватель выражений, оперирующих вещественными числами в формате с плавающей точкой. Например:

```

-34.3456E-6 + 0.56E+3* 0.7989E2

```

**Вариант 15.** C++. Разработать грамматику и распознаватель имен переменных. Предусмотреть: простые переменные и переменные – элементы массива, учесть, что индексы элементов массива могут быть выражениями.. Например, **A47** – простая переменная, **Brk[(k1+r)/u][6]** – элемент массива.

**Вариант 16.** C++. Разработать грамматику и распознаватель прототипов функции. Считать, что параметры только стандартных (скалярных) типов. Например:

```

int as3(int a_34, float bjt, unsigned char car);

```

**Вариант 17.** C++. Разработать грамматику и распознаватель описания инициализированных или неинициализированных массивов целых чисел. Предусмотреть задание размерности массива в виде диапазона литералами целого типа. Семантические проверки соответствия количества уровней скобок размерности массива и количества констант размерности массива не выполнять. Например:

```
Const int A7867[2][4]={6, -6,0,6},{59,0,78,6}};  
int a[8][9];
```

**Вариант 18.** C++. Разработать грамматику и распознаватель описания языка, включающего операторы присваивания и счетного цикла. Считать, что границы изменения переменной цикла могут быть заданы только целочисленными константами, условие – операцией отношения, язык оперирует только простыми переменными и целыми числами, тело цикла содержит не более одного оператора, а оператор присваивания в правой части содержит только идентификаторы или целые константы. Например:

```
For (i=1;i<= 10;i=i+1) for (jj= 20; i>= 10; i=i-1) a:=5;
```

**Вариант 19.** C++. Разработать грамматику и распознаватель описания языка, включающего оператор присваивания и оператор Switch. Считать, что выбор происходит по значению переменной целого типа, альтернативы определяются константами целого типа, а оператор присваивания в правой части содержит только идентификаторы или целые константы. Например:

```
switch (nday)  
{ case 1: case 2: case 3 case 4: case 5: a=b; break;  
case 6: swich (j)  
    {case 7:a=d; case 8: a=h;}  
case 7: a=0; }
```

**Вариант 20.** C++. Разработать грамматику и распознаватель описания языка, включающего оператор условной передачи управления и оператор присваивания. Считать, что условие – значение переменной логического типа, тело альтернативы содержит не более одного оператора, а оператор присваивания в правой части содержит только идентификаторы или целые константы. Например:

```
if (d) gyu=5 if (r) u=-78 else gju:=ty;
```

**Вариант 21.** C++. Разработать грамматику и распознаватель описания языка, включающего оператор цикла-пока и оператор присваивания. Считать, что условие – значение переменной логического типа, тело цикла содержит не более одного оператора, а оператор присваивания в правой части содержит только идентификаторы или целые константы. Например:

```
while (d) while (ii) gyu=5;
```

**Вариант 22.** C++. Разработать грамматику и распознаватель языка описаний структурных типов данных, включающего описание массивов, строк и структур. Предусмотреть следующие типы элементов: float, int, char, unsigned char. Например:

```
int aer[8];
char record[80];
struct student { char name[22]; char family[22]; int old; } st1,ft;
```

**Вариант 23.** C++. Разработать грамматику и распознаватель языка описаний функций. Предусмотреть передачу параметров только по значению, операторы присваивания, в которых используются переменные и константы целого типа.

```
int DD(char y,int u)
{ int i,j; i=y; j=u;}
```

**Вариант 24.** C++. Разработать грамматику и распознаватель языка описаний структур (через оператор определения типа). Предусмотреть инициализацию переменных структурного типа. Элементы и константы – символьные строки и целые числа. Семантику совпадения типов не проверять. Например:

```
typedef struct { char name[22];
               char family[22];
               int old; } student;
student stud1={"Петр","Петров",18}, stud2={"Иван","Иванов",19};
```

**Вариант 25.** C++. Разработать грамматику и распознаватель языка описания функций форматного ввода-вывода. Считать, что формат может быть задан строковой константой и переменной. Предусмотреть форматы %d, %c, %s, %f и \n. Например

```
scanf("%f %s",&s1, str);
printf("В строке %2d чисел/n", i);
```

**Вариант 26.** C++. Разработать грамматику и распознаватель описаний условной операции: <выражение1>?<выражение2>:<выражение3>; .Учесть, что Выражение 1 может включать операции отношения. В выражениях могут быть использованы только простые переменные и константы целого типа. Например:

```
a>45?h*(a+6):k;
```

**Вариант 27.** C++. Разработать грамматику и распознаватель описания языка, включающего оператор цикла-до и оператор присваивания. Считать, что условие – значение переменной логического типа, тело цикла содержит не более одного оператора, а оператор

присваивания в правой части содержит только идентификаторы или целые константы. Например

**do do a=90 while f while g;**

## ПОРЯДОК ВЫПОЛНЕНИЯ ДОМАШНЕГО ЗАДАНИЯ

Общая формулировка задания выглядит следующим образом.

Разработать программу, которая выполняет лексический и синтаксический анализ указанных конструкций языков Паскаль или C++. Программа должна обеспечивать многократный ввод предложений, их обработку с выводом на экран результатов лексического и синтаксического анализов и завершать работу при вводе слова "все". Для каждого введенного предложения анализатор должен возвращать «Конструкция распознана» или «Обнаружена ошибка».

При выполнении задания студент должен:

- разработать, записать в форме Бекуса-Наура и изобразить в виде синтаксических диаграмм грамматику заданных конструкций формального языка;
- используя формальные признаки определить тип грамматики по классификации Хомского;
- проанализировать правила грамматики и выбрать метод синтаксического анализа конструкций языка;
- в соответствии с выбранным методом синтаксического анализа выбрать способ реализации лексического анализа: построение подпрограммы сканера или использование распознавателей лексем по мере разбора предложений языка,
  - при необходимости выбрать и обосновать формат строки токенов;
  - разработать алгоритм и реализовать подпрограммы лексического анализа;
  - разработать алгоритм и реализовать подпрограммы анализа конструкций языка;
  - разработать тесты для тестирования программы;
  - тестировать и отладить программу;
  - оставить отчет по домашнему заданию;
  - продемонстрировать работу программы преподавателю;
  - защитить домашнее задание преподавателю.

## ТРЕБОВАНИЯ К ОТЧЕТУ

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Отчет по каждой части домашнего задания должен содержать:

- 1) текст задания;
- 2) описание грамматики в форме Бэкуса –Наура, указав тип грамматики;
- 3) обоснование выбора метода разбора;
- 4) текст программы;
- 5) таблицы тестов;
- 6) выводы.

Кроме того, все отчеты должны иметь титульный лист, на котором указывается:

- а) наименование факультета и кафедры;
- б) название дисциплины;
- в) номер и тема домашнего задания;
- г) фамилия преподавателя, ведущего занятия;
- д) фамилия, имя и номер группы студента;
- е) номер варианта задания.



## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение формального языка и формальной грамматики.

Ответ.

2. Как определяется тип грамматики по Хомскому?

Ответ.

3. Поясните физический смысл и обозначения формы Бэкуса –Наура.

Ответ.

4. Что такое лексический анализ? Какие методы выполнения лексического анализа вы знаете?

Ответ.

5. Что такое синтаксический анализ? Какие методы синтаксического анализа вы знаете?

К каким грамматикам применяются перечисленные вами методы?

Ответ.

6. Что является результатом лексического анализа?

Ответ.

7. Что является результатом синтаксического анализа?

Ответ.

8. В чем заключается метод рекурсивного спуска?

Ответ.

9. Что такое таблица предшествования и для чего она строится?

Ответ.

10. Как с использованием таблицы предшествования осуществляют синтаксический анализ?

Ответ.

## ЛИТЕРАТУРА

1. Иванова Г.С. Слайды лекций по дисциплине Машинно-зависимые языки и основы компиляции. Режим доступа: <http://e-learning.bmstu.ru/moodle/mod/resource/view.php?id=35> (дата обращения 3.03.2014).
2. Иванова Г.С., Ничушкина Т.Н. Основы конструирования компиляторов. Учебное пособие. – М.: МГТУ им. Н.Э. Баумана, 2010. Режим доступа: <http://e-learning.bmstu.ru/moodle/mod/resource/view.php?id=35> (дата обращения 3.03.2014).
3. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий = Compilers: Principles, Techniques, and Tools. — 2-е изд. — М.: Вильямс, 2010.
4. Креншоу Дж. Давайте создадим компилятор! Режим доступа: <http://www.kulichki.net/kit/crenshaw/crenshaw.html> (дата обращения: 3.03.2014).