Московский государственный технический университет
имени Н.Э. Баумана

==========

Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Г.С. Иванова, Т.Н. Ничушкина

ЛЕКСИЧЕСКИЕ И СИНТАКСИЧЕСКИЕ АНАЛИЗАТОРЫ

Электронное учебное издание

Методические указания к выполнению
домашнего задания № 2 по дисциплине
Машинно-зависимые языки и основы компиляции

Москва

УДК 004.415.2

Рецензент: доцент, к.т.н., Владимир Алексеевич Мартынюк

Иванова Г.С., Ничушкина Т.Н.

Лексические и синтаксические анализаторы: Методические указания к выполнению домашнего задания № 2 по дисциплине Машинно-зависимые языки и основы компиляции для студентов 2 курса кафедры ИУ6 (бакалавры). Электронное учебное издание. – М.: МГТУ имени Н.Э. Баумана, 2021. – 33 с.

Издание содержит описание теоретический материал, посвященный методам разработки лексических и синтаксических анализаторов для языков, грамматика которых относится к 2 и 3 типам по классификации Хомского. Рассмотрены распознаватели регулярных языков, построенные на конечных автоматах, и методы рекурсивного спуска и стековый, предназначенные для разбора контекстно-свободных формальных языков. Приведены примеры программ, демонстрирующие особенности использования различных методов анализа. Определены цель домашнего задания, последовательность его выполнения и требования к отчету.

Для студентов 1 курса кафедры ИУ6 МГТУ имени Н.Э. Баумана.

Рекомендовано учебно-методической комиссией факультета «Информатика и системы управления» МГТУ им. Н.Э. Баумана

Электронное учебное издание

Иванова Галина Сергеевна

Ничушкина Татьяна Николаевна

Лексические и синтаксические анализаторы.

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

Оглавление

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, $VT \cap VN = \varnothing$;

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

P – множество порождающих правил – каждое правило состоит из пары строк (α, β), где $\alpha \in V+$ – левая часть правила, $\beta \in V^*$ – правая часть правила: $\alpha \rightarrow \beta$, где строка α 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 G0:

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

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

P = {<целое> → <знак><целое без знака>, <целое> → <целое без знака>,

<целое без знака> → <цифра><целое без знака>, //правосторонняя рекурсия

<целое без знака> → <цифра>,

<цифра> → 0, <цифра> → 1, <цифра> → 2, <цифра> → 3, <цифра> → 4,

<цифра> → 5, <цифра> → 6, <цифра> → 7, <цифра> → 8, <цифра> → 9,

<знак> → +, <знак> → - };

S = <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: «::=» – «можно заменить на»; «|» – «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:

<целое> ::= <знак><целое без знака>|<целое без знака>,

<целое без знака> ::= <цифра><целое без знака>|<цифра>,

<цифра > ::= 0|1|2|3|4|5|6|7|8|9,

<знак> ::= +| - .

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 токену.

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 чаще всего используют 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.

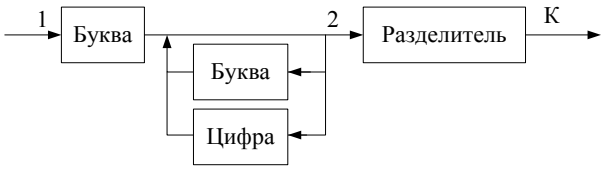

Рисунок 1 – Синтаксическая диаграмма лексемы Идентификатор

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;
```

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*.

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, поэтому обычно один и тот же автомат распознает несколько конструкций.

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 предшествования, называемых LR(k)-грамматиками.

1.3.1. Метод рекурсивного спуска для грамматик LL(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).

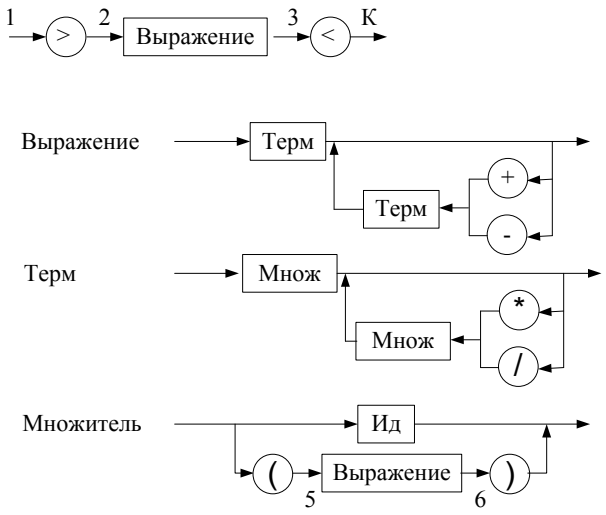

Рисунок 2 – Синтаксические диаграммы проверки правильности записи Выражения

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;
  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 Video1.avi.

1.3.2. Разбор грамматик предшествования LR(k)

Разбор грамматики по правилам грамматики предшествования 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:

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, в котором фиксируется ошибка 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 Scaner(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;
    {тело функции описано в разделе 1.2}

Function Scaner(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 Odna_osnova(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: Odna_osnova(St,Stack);
     4: begin
          Result:=true;
          Konec:=true;
          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;

Begin
  Writeln('Input Strings:'); Readln(St);
  R:=true;
  While (St<>'end') and R do
    Begin
      R:=Scaner(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 Vidio2.avi.

```
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
```

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:=hhh+(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 Scaner(St:shortstring;Razd:setofChar;
                          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;
```

Результат выполнения программы приведен ниже.

Пояснения к результатам выполнения программы представлены в учебном фильме Video3.avi.

```
Input Strings:
if aaaa>vvvv then j:=hhhh else if h then ffff:=hhh*(ppp+yyy);
Slugeb slovo if
Identify=aaaa
Slugeb simbol >
```

```
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

Варианты заданий приведены на странице дисциплины на сайте кафедры.

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

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.

2. Как определяется тип грамматики по Хомскому?

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

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

5. Что такое синтаксический анализ? Какие методы синтаксического анализа вы знаете? К каким грамматикам применяются перечисленные вами методы?

6. Что является результатом лексического анализа?

7. Что является результатом синтаксического анализа?

8. В чем заключается метод рекурсивного спуска?

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

10. Как с использованием таблицы предшествования осуществляют синтаксический анализ?

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).