**Процесс трансляции**

Транслятор - это программа-переводчик, предназначенная для преобразования исходного текста в объектный модуль, доступный для восприятия ЭВМ, т.е. машинный код.

Любой транслятор выполняет следующие функции:

• анализ исходного текста программы на выявление синтаксических ошибок;

• генерацию выходной (рабочей) программы (объектного модуля) на язык команд конкретного процессора (генерация может происходить в два этапа: сначала до уровня команд ассемблера, затем до машинного кода);

• распределение памяти для размещения программы в машинных кодах (выделение физических адресов участков памяти для каждого фрагмента программы, для таблиц переменных, констант, массивов и других объектов).

При проектировании трансляторов выделяются две системы (два подхода к процессу трансляции): компилирующая и интерпретирующая. Компилирующая система просматривает весь текст от начала до конца и в случае отсутствия синтаксических ошибок преобразует в машинный код. Интерпретирующая, или интерпретатор преобразует текст построчно (по командам) с непосредственным выполнением вне зависимости от последующих конструкций(предложений) языка.

Лексический анализатор - это наиболее простая часть компилятора для проектирования. Предложения (конструкции) формируются из лексем. Предложения исходной программы поступают на вход компилятора в виде последовательности лексем. Лексема - нечто неделимое, которое рассматривается как единица (объект) языка. Это могут быть и ключевые слова, и имена переменных, и символы операций и т.д.

Синтаксический и семантический анализаторы выполняют сложную алгоритмическую работу по выделению составных частей из исходного текста программы с целью формирования ее внутреннего представления под полным синтаксическим и семантическим контролем. Синтаксический анализ выполняется по правилам формальной грамматики. Семантический анализ п(роверка на содержание предложения или, если проще сказать, смысл) является наиболее трудной задачей для программной реализации. Семантика тесно связана с подготовкой и генерацией машинных команд. Результатом синтаксического анализатора является последовательность тетрад (четверок): код операции, два операнда и результат.

**Этапы трансляции. Генерация кода**

Логически процесс трансляции разделяется на два этапа: анализа и синтеза. Этап анализа (здесь анализируется исходный текст), как правило, имеет три отдельные фазы:

- лексический анализ;

- синтаксический анализ;

- семантический анализ.

Этап синтеза (здесь генерируется машинно-ориентированное представление кода программы) состоит из следующих всех перечисленных или выборочных фаз:

- генерация машинно-независимого кода;

- оптимизация машинно-независимого кода;

- распределение памяти;

- генерация машинного кода;

- оптимизация машинного кода.

Большинство современных трансляторов производят промежуточный код, что делает их независимыми от типа процессора. UIL (Universal Intermediate Language) – некий универсальный язык для всех языков на любую машину пока не создан. Но в роли UIL используются Р-код для Паскаля, байт-код для JAVA, язык С (С++) также часто выполняет такую задачу (например, транслятор для языка Clipper написан на С).

**Проектирование языковых интерпретаторов**

В области профессиональной деятельности узкий круг специалистов, тесно связанный с программированием, принимает непосредственное участие в создании и модификации полных трансляторов. Но в практической работе часто такие знание бывают необходимы:

- для формирования интерактивной среды;

- для разработки эффективных отладочных средств;

- для организации режима запросов к базам данных.

Для проектирования интерпретатора очень важным вопросом являются правила разбора выражений. Строгие правила алгебры и использование рекурсивного подхода позволяют упростить решение этой

задачи.

**Пример программы на лексический анализ**

Предположим, по выбранному варианту будем иметь следующий вид исходного текста:

***var a,b,c : integer;***

***begin***

***a=10-c\*b/2;***

***end***

Данный исходный текст необходим для проверки правильности составленной программы. Программа должна быть написана языке С++.

Для самостоятельной работы предлагается изучить алгоритм по тексту работающей программы, написанной на языке Паскаль и протокол результата выполнения лексического анализа.

Program stud\_work;

{ ЛЕКСИЧЕСКИЙ АНАЛИЗАТОР

таблица кодов лексем

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Лексема Код

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

var 1

begin 2

end 3

read 4

write 5

integer 6

+ 100

- 101

\* 102

/ 103

= 104

( 105

) 106

, 107

: 108

; 109

0 - тип служебных слов и перечисленных выше операций

ident 200 – тип идентификаторов

const 300 – тип констант }

Const

kT=6;

{ 6 - количество служебных слов: var, begin, end, read, write, integer}

n\_str=500;

{ предельный размер исходного текста в символах, включая пробелы}

n\_ident=20;

{ ограничение на количество служебных слов + переменных }

n\_lex=100;

{ ограничение на общее число лексем }

Type

TS=string[10];

{ длина имени лексемы не превышает 10 символов}

Var

a\_text : array[1..n\_str] of char;

{ входная программа - посимвольное представление исходного текста }

LL : array[1..n\_lex] of integer;

{ массив внутренних кодов лексем - результат лексического анализа }

TT : array[1..n\_lex] of integer;

{ массив типов для каждой лексемы исходного текста }

VV : array[1..n\_ident] of TS; { таблица идентификаторов }

in\_text : text; { текстовой файл программы }

out\_int : file of integer;

{ выходной файл кодов лексем с указанием типа }

out\_ident : file of TS;

{ выходной файл идентификаторов: служебные слова и

переменные }

prt : text; { файл для размещения протокола выполнения программы }

n\_text : integer; { счетчик длины массива a\_text }

nL : integer; { счетчик длины массива LL }

nT : integer; { счетчик длины массива TT }

i : integer; { вспомогательная переменная }

{ Начальная установка, заполнение таблицы идентификаторов всеми

служебными словами, предварительно зарезервированными в таблице

кодов лексем }

procedure ust;

begin VV[1]:='var';

VV[2]:='begin';

VV[3]:='end';

VV[4]:='read';

VV[5]:='write'; VV[6]:='integer';

nT:=kT; nL:=0;

end;

{ ввод исходного текста }

procedure input;

var a : char; { один символ }

begin

assign(in\_text,'prim.lex'); { связь файловой переменной с файлом }

{prim.lex – произвольно выбранное имя файла, имя расширения

указывает, что текст файла предназначен для лексического анализа }

reset(in\_text); { открытие файла для чтения }

n\_text:=0;

while not eof(in\_text) do

begin read(in\_text,a); n\_text:=n\_text+1;

a\_text[n\_text]:=a;

end;

end;

{ формирование результатов в виде файлов: cod.lex,ident.lex }

procedure out;

var i : integer;

begin

assign(out\_int,'cod.lex');

rewrite(out\_int);

assign(out\_ident,'ident.lex');

rewrite(out\_ident);

write(out\_int,nL,nT);

for i:=1 to nL do write(out\_int,LL[i],TT[i]);

for i:=1 to nT do write(out\_ident,VV[i]);

end;

{ Распознавание буквы - true(1), иначе false(0) }

function bukva(a : char): integer;

var

k,kz : integer;

begin k:=ord(a); { ord - фунция дает код символа в ASCII }

if ( k>=64 ) and ( k<=90 ) or

( k>=96 ) and ( k<=122 )

then kz:=1

else kz:=0; bukva:=kz;

end;

{ Распознавание цифры - true(1), иначе false(0) }

function cifra( a : char) : integer ;

var k,kz : integer;

begin k:=ord(a);

if ( k>=48 ) and ( k<=57 )

then kz:=1

else kz:=0;

cifra:=kz;

end;

{ Размер идентификатора (количество символов, входящих в образование

имени лексемы), где начало i, конец k }

function l\_ident( i : integer):integer;

var k:integer;

begin

k:=i;

while ( bukva(a\_text[k])=1 ) or ( cifra(a\_text[k])=1 )

do k:=k+1;

k:=k-1;

l\_ident:=k;

end;

{ Размер цифровой константы ( начало i, конец k ) }

function l\_const ( i : integer ) : integer;

var k:integer;

begin

k:=i;

while cifra(a\_text[k])=1 do k:=k+1;

k:=k-1;

l\_const:=k;

end;

{ Определение типа и кода лексемы ( строки str ), если данной лексемы нет в таблице служебных слов (массив - VV), тогда она добавляется в массив VV }

procedure ttin( kk :integer; str : string;

var kod,typ : integer );

var k,i : integer;

begin

{ поиск в таблице VV }

k:=0;

for i:=1 to nT do

if str=VV[i] then k:=i;

{ идентификатор имеется в таблице VV }

if k<=kT then

begin kod:=k; typ:=0; end; { служебные слова }

if k>kT then

begin kod:=k; typ:=kk; end; { имя переменной }

{ идентификатора нет в таблице, тогда он добавляется }

if k=0 then

begin nT:=nT+1; VV[nT]:=str; kod:=nT; typ:=kk; end;

end;

{ лексический анализ }

procedure lexan;

var

i,j,k,kod,typ : integer;

a: char; str : string;

begin

i:=0;

while i<=n\_str do

begin

i:=i+1;

a:=a\_text[i];

if ord(a) > 32 then

begin

if bukva(a)=1 then

{ если a - буква, то может быть только идентификатор }

begin k:=l\_ident(i);

str:='';

for j:=i to k do str:=str+a\_text[j];

ttin(200,str,kod,typ);

nL:=nL+1; LL[nL]:=kod; TT[nL]:=typ;

i:=k;

end;

{ если a - цифра, то далее может быть

только цифровая константа }

if cifra(a)=1 then

begin k:=l\_const(i); { цифровая константа }

str:='';

for j:=i to k do str:=str+a\_text[j]; ttin(300,str,kod,typ);

nL:=nL+1; LL[nL]:=kod; TT[nL]:=typ;

i:=k;

end;

{ если a - не буква и не цифра, то далее

может быть только операция }

if ( bukva(a)<>1 ) and ( cifra(a)<>1 ) then

begin

{ проверка на код операции }

case a of

'+': kod:=100;

'-': kod:=101;

'\*': kod:=102;

'/': kod:=103;

'=': kod:=104;

'(': kod:=105;

')': kod:=106;

',': kod:=107;

':': kod:=108;

';': kod:=109;

end;

typ:=0;

nL:=nL+1; LL[nL]:=kod; TT[nL]:=typ;

end;

end;

end; { end of while }

end;

{ основная программа }

begin

ust; { начальная установка }

input; { ввод программы }

lexan; { лексический анализ }

assign(prt,'prot.txt');

{ con - вывод на экран,

prot.txt – произвольное имя файла для записи протокола

выполнения программы }

rewrite(prt);

writeln(prt,' Количество лексем=',nL,

' Количество идентификаторов и констант=',nT);

writeln(prt);

writeln(prt,' Результат работы лексического анализатора');

writeln(prt); writeln(prt,' Исходная программа');

for i:=1 to n\_text do

write(prt,a\_text[i]);

writeln(prt); writeln(prt);

writeln(prt,' тип лексемы : код лексемы');

for i:=1 to nL do

begin write(prt,TT[i]:3,':',LL[i]:3,' ');

if i mod 4 = 0 then writeln(prt);

end;

writeln(prt); writeln(prt);

writeln(prt,' Таблица идентификаторов и констант');

for i:=1 to nT do

write(prt,VV[i],' '); writeln(prt);

writeln(prt,' Конец протокола');

close(prt);

out;

end.

**Протокол результата лексического анализа**

Представленный ниже протокол, сформированный в виде текстового файла, отражает процесс выполнения лексического анализа для конкретного варианта разбора исходного текста по кодам и типам лексем.

Количество лексем=21 Количество идентификаторов и констант=10

Результат работы лексического анализатора

Исходная программа

var a,b,c : integer;

begin

a=c-b\*b/2;

end

тип лексемы : код лексемы

0: 1 200: 7 0:107 200: 8

0:107 200: 9 0:108 0: 6

0:109 0: 2 200: 7 0:104

200: 9 0:101 200: 8 0:102

200: 8 0:103 300: 10 0:109

0: 3

Таблица идентификаторов и констант

var begin end read write integer a b c 2

Конец протокола

Подобный результат может быть получен для любого варианта.

Лабораторная работа считается выполненной и правильно оформленной при наличии титульного листа, текста задания, выбранному по своему шифру, исходной программы на языке С++ и протокола, подтверждающего работоспособность представленной программы.

**2 сентября 2020**

**Задание**

Разработать программу по выполнению лексического анализа для языка программирования, удовлетворяющего ниже перечисленным требованиям.

Упрощенный язык программирования должен обязательно включать:

***1. Оператор присваивания "=" и только одну из следующих арифметических операций:***

+ ( сложение ),

- ( вычитание ),

\* ( умножение ),

/ ( деление ),

++ ( логическое сложение ),

\*\* ( логическое умножение),

%% ( сложение по модулю 2).

***2. Один из следующих операторов:***

* оператор цикла, построенный в соответствии с предложенным синтаксисом конструкции:

*FOR <имя параметра цикла>=m TO n <тело цикла> NEXT;*

* оператор цикла, построенный в соответствии с предложенным синтаксисом конструкции:

*FOR <имя параметра цикла>=m TO n DO BEGIN <тело цикла>*

*END;*

* условный оператор, построенный в соответствии с предложенным синтаксисом конструкции:

*IF <условие> THEN BEGIN <операторы> END;*

здесь <условие> задается одной из форм: a<b, либо a=b, либо a>b;

* оператор процедуры, построенный в соответствии с предложенным синтаксисом конструкции:

*PROCEDURE <имя процедуры> BEGIN <тело процедуры> END*

для вызова процедуры используется оператор

*CALL <имя процедуры>;*

***3. Оператор вывода переменных***

*WRITE ( <список переменных через запятую> );*

***4. Программа языка имеет структуру***

*VAR <список переменных через запятую >: INTEGER*

*BEGIN <операторы программы> END*

При выполнении лабораторной работы в качестве образца можно использовать демонстрационную модель лексического анализатора, написанную на языке ПАСКАЛЬ.

На фазе лексического анализа необходимо выделить лексемы, закодировать их и поместить в таблицу идентификаторов. Для этого необходимо выделить символы операций, имена переменных и служебные слова, которые определены в задании.

Лабораторная работа считается выполненной и правильно оформленной при наличии титульного листа, текста задания, исходной программы на языке С++ и протокола, подтверждающего работоспособность представленной программы.