Министерство образования и науки РФ

ГОУ ВПО Тамбовский государственный технический университет

Кафедра САПР

**Курсовая работа**

По дисциплине «Лингвистическое программное обеспечение систем

автоматизированного проектирования»

**Тема:** «Разработка трёхпроходного транслятора с исходного языка

на язык паскаль»

Выполнил студент гр.

.

Проверила: доцент.

Тамбов 20 г.

**Пояснительная записка**

**Задание на проектирование**

Разработать двухпроходный транслятор с исходного языка на язык паскаль. В качестве метода синтаксического анализа использовать метод операторного предшествования.

*Исходный текст программы:*

программа xy;

переменные k,l:целые;

sx,sy,x,y:вещественные;

начало

sx=0;

sy=0;

цикл k от 0 до 5 шаг 1 выполнять

{

sx:=sy+x;

sy:=0;

цикл l от 1 до 10 шаг 1 выполнять

sy:=sy+y

};

вывести (sx,sy)

конец.Дата принятия задания к исполнению 27.02.20

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Содержание**

|  |  |
| --- | --- |
| Пояснительная записка |  |
| Задание на проектирование |  |
| Содержание |  |
| Введение |  |
| Описание процесса решения задачи |  |
| Блок-схема основной программы и процедур |  |
| Распечатка программных модулей. |  |
| Описание программы |  |
| Назначение и общее описание программы |  |
| Описание логической структуры программы |  |
| Способ обращения к программе |  |
| Перечень технических средств. |  |
| Описание входных и выходных данных |  |
| Текстовые примеры работы программы |  |

## Введение

В состав любой вычислительной системы может входить комплекс программ, которые называются трансляторами. Транслятор обеспечивает автоматический перевод программ с алгоритмического языка в машинные коды.

По функциональному назначению трансляторы делятся на компиляторы (перевод программ на языке высокого уровня в машинные коды без выполнения), интерпретаторы (перевод каждой конструкции алгоритмического языка в машинные коды с одновременным выполнением) и ассемблеры (перевод программы с языка низкого уровня в машинные коды).

Более подробно остановимся на компиляторах. Компилятор – это не что иное, как программа, написанная на некотором языке, для которой входной информацией служит исходная программа, а результатом является эквивалентная ей объектная программа. Раньше компиляторы писались на автокоде. Часто это был единственно доступный язык. Однако сейчас существует тенденция писать компиляторы на языках высокого уровня, поскольку при этом уменьшается время, затрачиваемое на программирование и отладку, а также обеспечивается удобочитаемость компилятора, когда работа над ним завершена.

Компиляторам присущ ряд общих черт, что упрощает процесс создания компилирующих программ. Наша цель состоит в том, чтобы описать известные уже модельные представления структуры компиляторов и показать, как с их помощью создается работоспособная компилирующая программа.

Компилятор должен выполнить анализ исходной программы и синтез объектного кода. В соответствии с этим любой компилятор включает три основные части: лексический анализатор, синтаксический анализатор и генератор кода.

Взаимодействие между компонентами компилятора может осуществляться разнообразными способами.

В задании требуется, что бы программа обеспечивала перевод текста программы написанного на языке высокого уровня на язык высокого уровня паскаль, а также проверяла исходный текст программы на наличие ошибок.

Курсовая работа содержит текст и описание программы, в которой реализовано задание, описание процесса решения задачи, текстовые примеры работы программы, блок - схемы основных процедур.

Финальная версия программы «Транслятор» написана к 28.05.2012.

## Описание процесса решения задачи

**Грамматика исходного языка**

В качестве формы записи грамматики выбрана форма Бекуса-Наура (БНФ). Грамматика БНФ состоит из множества правил вывода, каждое из которых определяет синтаксис некоторой конструкции языка программирования. Строки символов, заключённые в угловые скобки < и >, называются нетерминальными символами, то есть являются именами конструкций, определенных внутри грамматики. То, что не заключено в угловые скобки, называется терминальными символами (лексемами).

<программа> -> программа <имя программы>;переменные <раздел переменных>;начало <раздел операторов>;конец.

<имя программы> -> идентификатор

<раздел переменных> -> {<список переменных> : <тип>}

<список переменных> -> идентификатор {, идентификатор}

<тип> -> вещественные/целые

<раздел операторов> -> <оператор>; {оператор;}

<оператор> -> <сложный оператор> / <присваивание> / <цикл> / <вывод>

<сложный оператор> -> { <оператор>{;оператор} }

<присваивание> -> идентификатор :=/= <выражение>

<выражение>-><слагаемое>{+<слагаемое>}

<слагаемое>-> идентификатор / константа/(<выражение>)

<цикл> -> цикл <выражение> от <выражение> до <выражение> шаг <выражение> выполнять <оператор>

<вывод>-> вывести ( <список переменных> )

В связи с тем, что оба языка – языки высокого уровня, необходимость в генерации кода из постфиксной записи отпадает (код будет генерироваться из лексем). Поэтому наличие матрицы операторного пришествия нецелесообразно. Вместо неё будет приведён ниже фрагмент текста программы с матрицей, имеющей отношения либо 0, либо 1. Отношение 0 – если данные конструкции по грамматике исходного языка не могут встречаться рядом, 1 – в обратном случае.

"00010000000000000110011111",

"00000000000000000000000110",

"00000000000001100000000000",

"00010000011010011000010100",

"00000000000000000000000000",

"00000100000000000000000110",

"00010000001000000000001000",

"00000000000000000000000110",

"00000000000000000000000110",

"00010000010000001000010100",

"00010000000000000000001000",

"00000000000000000000000001",

"00000000000000000000000100",

"00010000000000000000000000",

"00010000000000000000000000",

"00010000010000001000011100",

"00000000000000000000000100",

"00000000000000000000000110",

"00000000000000000000000110",

"00000000000000000000000110",

"00010000010000001000010100",

"00000100000000000000000000",

"00001000000000000000000000",

"01110011101000000111100000",

"01010010100000000011100000",

"00010000000000000000000000"

Работа транслятора начинается с распознавания имени программы и всех переменных, описанных между “переменные” и “:” через запятую. Имя программы и все переменные заносятся в массив. Тут же происходит проверка на некорректное имя программы (не должны содержаться пробелы в имени программы, точка с запятой обязательно ставится вплотную к имени программы) и на повторное описание идентификаторов, выдавая соответствующие ошибки соответственно. При нахождении этих ошибок программа тут же закрывается. Следующий этап, он же основной, это лексический анализатор вместе с синтаксическим анализатором. Рассмотрим этот этап подробнее.

|  |  |
| --- | --- |
| номер | Лексема |
| 1 | , |
| 2 | : |
| 3 | ; |
| 4 | . |
| 5 | ( |
| 6 | ) |
| 7 | = |
| 8 | + |
| 9 | { |
| 10 | } |
| 11 | Программа |
| 12 | Переменные |
| 13 | Целые |
| 14 | Вещественные |
| 15 | Начало |
| 16 | Цикл |
| 17 | От |
| 18 | До |
| 19 | Шаг |
| 20 | Выполнять |
| 21 | Вывести |
| 22 | Конец |
| 23 | Идентификатор |
| 24 | Константа |
| 25 | имя программы |

**Лексический и синтаксический анализ:**

Текст программы записывается в строку из поля Memo1,вызывается синтаксический анализатор, который в цикле до конца программы или этой строки вызывает лексический анализатор, тот считывает лексемы и записывает в массив curlex до конца оператора,причём в поле curlex[n][0] записывается номер идентификатора и если это переменная или константа то записывается номер переменной или значение константы соответственно в поле curlex[n][1],в ходе лексического анализа происходит также и анализ переменных. Лексический анализатор передаёт список лексем ясинтаксическому анализатору. Если произошла ошибка, анализатор выдаёт ошибку и синтаксический анализ кода не производится.

В синтаксическом анализаторе используется двумерный массив matr – описанная ранее матрица. Сравниваются соседние элементы этого массива и если между ними символ “0”, то выдаётся синтаксическая ошибка Был предусмотрен случай неровного количества открывающих и закрывающих скобок,

Текст из поля Memo1 записывается в массив syntmas и начинается следующий проход.

**Генератор кода:**

При отсутствии ошибок на стадии анализа происходит генерация кода

В цикле от начала до конца массива syntmas (именно туда записывались номера лексем) происходит распознание лексем и перевод их в код на языке паскаль. Анализ лексем тривиален. Обратим ещё внимание на генерирование констант и переменных. Для переменных смотрим номер после символа “-“ , например, для 26-3 это будет символ “3”. И по этому номеру обращаемся к массиву идентификаторов и извлекаем оттуда имя соответствующего идентификатора. Для констант мы смотрим на код после “27” и выводим его, то есть для 10 этот код будет 10. На этом этап завершён. Результат выводится в поле Memo2.

**Блок–схема лексического анализатора**

Известный идентификатор?

да нет

Идёт объявление переменных?

да нет

Установление флага ошибки в true

Объявить переменную

Запись лексемы в массив

Конец оператора/программы?

Нет да

Возврат в синтаксический анализатор

**Блок-схема синтаксического анализатора**

Вовращена ошибка?

да нет

Проверка соседних символов

Ошибка?

нет да

Вывести сообщение

Установление флага ошибки в true

Объявить переменную

Вывод сообщения

Конец программы?

Нет да

Были ошибки?

Вывод сгенерированного кода

**Блок-схем генерация кода**

Посимвольное чтение строки

конец

Окончание строки

Вывод в поле Memo2

Сравнение с номерами лексем

**Структура программы**

Начало

Memo1

Синтаксический анализатор

Генератор кода

Memo2

Вывод данных

Конец

Текст программы:

//---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"

//---------------------------------------------------------------------------

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

//---------------------------------------------------------------------------

String lexmas[]={" ",",",":",";",".","(",")","=","+","{","}","ïðîãðàììà","ïåðåìåííûå","öåëûå","âåùåñòâåííûå","íà÷àëî","öèêë",

"îò","äî","øàã","âûïîëíÿòü","âûâåñòè","êîíåö","èäåíòèôèêàòîð","êîíñòàíòà","èìÿ"};

String pasmas[]={" ",",",":",";\n","."," "," ",":=","+","begin\n","\nend","program ","var\n","integer","real","begin\n","for ",

" from "," to "," step "," do\n","writeln ","\nend","èäåíòèôèêàòîð","êîíñòàíòà","èìÿ"};

int syntmas[300][3],mn=0;

int lex[300],curlex[100][3],nlex=0,curc=1,constn=0,varn=0,curLine=0,curpl1,fsc,rsc;

char pname[30];

char var[30][10],cns[30];

int pos=0;

String pl1[100];

String str;

boolean firstVal,varIn=false,end=false,begIn=false,trueEnd=false,globerr=false,varSetEn=false,outEn=false,inEn=false;

boolean err=false;

boolean oper();

bool val();

char matr[26][26]={

"00010000000000000110011111",

"00000000000000000000000110",

"00000000000001100000000000",

"00010000011010011000010100",

"00000000000000000000000000",

"00000100000000000000000110",

"00010000001000000000001000",

"00000000000000000000000110",

"00000000000000000000000110",

"00010000010000001000010100",

"00010000000000000000001000",

"00000000000000000000000001",

"00000000000000000000000100",

"00010000000000000000000000",

"00010000000000000000000000",

"00010000010000001000011100",

"00000000000000000000000100",

"00000000000000000000000110",

"00000000000000000000000110",

"00000000000000000000000110",

"00010000010000001000010100",

"00000100000000000000000000",

"00001000000000000000000000",

"01110011101000000111100000",

"01010010100000000011100000",

"00010000000000000000000000"

};

//------------ñîîáùåíèå îá îøèáêå

void errmes(String st,String id,int line)

{

String erm=st+" "+id+"â ñòðîêå "+(line+1);

Form1->Memo2->Lines->Add(erm);

}

//----óçíàåì ëåêñåìó

int match(String st)

{

for(int i=0;i<=22;i++)

if(lexmas[i]==st) return i;

return -1;

}

int varMatch(char\* st)

{ int y=-1;

for(int i=0;i<varn;i++)

{if(!strcmp(var[i],st)) {y=i; break;}

}

return y;

}

//----äîáàâèòü ïåðåìåííóþ

int addVar(char\* st)

{ int y=-1;

for(int i=0;i<varn;i++)

if(!strcmp(var[i],st)) y=i;

if(y==-1) if(strcmp(pname,st)){

strcpy(var[varn],st);

varn++;

return varn-1;

}

return -1;

}

//---------÷èñëî ëè ýòî

int constmatch(char\* st)

{int i=-1;

if(TryStrToInt(st,i))

return i;

return -1;

}

//---------ëåêñè÷åñêèé àíàëèç

boolean getLex()

{ bool sloEnd=false;

//Form1->Memo2->Lines->Add("lexan");

char cc[2],cs[100];

boolean stop=false,progIn=false,error=false;

nlex=0;

int n,cul=0,lexc,cuc,curv,curConst;

nlex=0;

cs[cul]='\0';

for(int i=curc;!stop && i<=str.Length();i++)

{

//break;

cc[0]=str[i];

cc[1]='\0';

//Form1->Memo2->Lines->Add(cc.Length());

//break;

if(cc[0]=='\n' || cc[0]==10) {curLine++;

}

cuc=match(cc);

if((cuc>=0 && cuc<=10) || (cc[0]==13) || (cc[0]==10))

{

if(cs[0]!=13 && strlen(cs)!=0 && cs[0]!=10)

{lexc=match(cs);

//Form1->Memo2->Lines->Add(cs);

if(lexc!=-1)

{if(lexc==12) {varIn=true; }

if(varIn && lexc==15) {varIn=false; begIn=true;}

if(lexc==11) progIn=true;

curlex[nlex][0]=lexc;

nlex++;

}

else

{

if(progIn)

{strcpy(pname,cs);

progIn=false;

curlex[nlex][0]=25;

nlex++;}

else if(varIn){

curv=addVar(cs);

if(curv!=-1 && constmatch(cs)==-1) {curlex[nlex][0]=23; curlex[nlex][1]=curv; curlex[nlex][2]=curLine;nlex++;}

else{errmes("íåâåðíûé èäåíòèôèêàòîð",cs,curLine);

error=true;}}

else{

curConst=constmatch(cs);

if(curConst!=-1)

{curlex[nlex][0]=24; curlex[nlex][1]=curConst; curlex[nlex][2]=curLine; nlex++;} else

if(varMatch(cs)!=-1)

{curlex[nlex][0]=23; curlex[nlex][1]=varMatch(cs); curlex[nlex][2]=curLine;nlex++;}

else if(cs[0]!=10 && cs[0]!=0 && cs[0]!=13)

{error=true; errmes("íåèçâåñòíûé èäåíòèôèêàòîð",cs,curLine);}

}

}

}

if(cuc!=0 && cc[0]!=13 && cc[0]!=10)

{curlex[nlex][0]=cuc;

curlex[nlex][2]=curLine;

{if(cuc==7)

{if(nlex!=0) {if(curlex[nlex-1][0]==2) {curlex[nlex-1][0]=7; nlex--;}

}}

}

nlex++;}

if(!sloEnd)

if(cuc==4)

stop=true;

if(cuc==4) end=true;

cul=0;

cs[cul]='\0';

}

else{

if(cc[0]!=13 && cc[0]!=10)

{cs[cul]=cc[0];

cs[cul+1]='\0';

cul++;}

}

curc++;

if(curc==str.Length()) end=true;

}

return error;

}

//---------------ãåíåðàòîð êîäà

void gk()

{String str="";

for(int i=0;i<nlex;i++)

{

switch (curlex[i][0])

{

case 23: str+=var[curlex[i][1]]; break;

case 24: str+=curlex[i][1]; break;

case 25: str+=pname; break;

default: str+=pasmas[curlex[i][0]]; break;

}

}

Form1->Memo2->Lines->Text=str;

}

//------------ñèíòàêñè÷åñêèé àíàëèçàòîð

bool syntan()

{bool error;

int i,line=0;;

//cpos=nlex;

error=getLex();

if(false)

for(int i=0;i<nlex;i++)

{

int l=curlex[i][0];

Form1->Memo2->Lines->Add(lexmas[l]);}

if(error) globerr=true;

if(!error)

{

for( i=0;i<nlex-1;i++)

{if(curlex[i][0]==5) rsc++;

if(curlex[i][0]==6) rsc--;

if(curlex[i][0]==9) fsc++;

if(curlex[i][0]==10) fsc--;

if(matr[curlex[i][0]][curlex[i+1][0]]=='0') {error=true; line=curlex[i][2];break;}

}

}

if(error) globerr=true;

if(error)errmes("ñèíòàêñè÷åñêàÿ îøèáêà","",line);

return error;

}

\_\_fastcall TForm1::TForm1(TComponent\* Owner)

: TForm(Owner)

{

Memo1->Clear();

Memo2->Clear();

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{pos=0;

fsc=0;rsc=0;

str=Memo1->Text;

curc=1;nlex=0;

constn=0,varn=0,curLine=0,curpl1=0;

end=false,begIn=false,trueEnd=false,globerr=false,varIn=false,firstVal=false,varSetEn=false,outEn=false,inEn=false;

Form1->Memo2->Clear();

//boolean err=getLex();

bool eror;

while (!end)

{

eror=syntan();

if(end) if(!trueEnd) ;

}

if(fsc!=0 || rsc!=0)

{globerr=true;

errmes("ñèíòàêñè÷åñêàÿ îøèáêà","",curLine);

}

if(!globerr)

gk();

}

//---------------------------------------------------------------------------

**Описание программы:**

**1. Описание технических средств**

Программа была разработана на IBM совместимой машине с процессором AMD Athlon™ 64 X2 Dual, 2.20 GHz на языке C++ (Borland C++ Builder 6) под управлением ОС Microsoft Windows Vista Home Basic Service Pack 1. Особых системных требований не имеет. Предполагается работа программы на любом IBM совместимом компьютере под управлением Windows 95 / 98 / Me / Xp / Vista / Windows 7. Запуск в других системах не возможен.

**2. Описание входа файла**

Входной представляет собой программу на структурно языке высокого уровня. Вывод конечного результата работы анализатора осуществляется на экран.

**3. Способ обращения к программе**

Для запуска программы необходимо выполнить следующий порядок действий:

* Запустить на выполнение файл Project1.exe.
* В поле Memo1 ввести исходный текст.
* В поле Memo2 будет записан сгенерированный код.

**4. Описание логической структуры программы**

Программа состоит из одного логического модуля.

**5. Назначение и общее описание программы**

Данная программа предназначена для перевода кода с исходного языка на язык паскаль. Программа реализована на основе трёхпроходного транслятора.

Данная программа предназначена для перевода кода с исходного языка на язык ассемблера. Программа реализована на основе двухпроходного транслятора.

На вход лексическому анализатору, поступает цепочка отдельных терминальных символов, которые группируются им в единые синтаксические объекты – лексемы. Цепочка лексем представляет собой два массива чисел, объединённых в структуру. После распознания лексемы, исследуется цепочка лексем и устанавливается, принадлежит ли она структурным объектам заданного языка. Если проверка прошла успешно, то генерируется код.

Синтаксический и лексический анализаторы распознают следующие типы ошибок:

* Лексические ошибки (связанные с неправильным написанием идентификатора / неверным его обозначением)
* Синтаксические ошибки (связанные с использованием неверных синтаксических конструкций)
* Неизвестные конструкции (неверное написание ключевого слова)
* Двойное описание идентификатора

При обнаружении ошибки выводится сообщение но дальнейшее выполнение программы продолжается.

Процесс работы программы скрыт от пользователя и на выходе ему доступен только уже сгенерированный код. При ошибках пользователю выводятся соответствующие сообщения.

**Текстовые примеры работы программы:**

* **Программа с правильным исходным кодом:**

программа xy;

переменные k,l:целые;

sx,sy,x,y:вещественные;

начало

sx=0;

sy=0;

цикл k от 0 до 5 шаг 1 выполнять

{

sx:=sy+x;

sy:=0;

цикл l от 1 до 10 шаг 1 выполнять

sy:=sy+y

};

вывести (sx,sy)

конец.

*Сгенерированный код:*

program xy;

var

k,l:integer;

sx,sy,x,y:real;

begin

sx:=0;

sy:=0;

for k from 0 to 5 step 1 do

begin

sx:=sy+x;

sy:=0;

for l from 1 to 10 step 1 do

sy:=sy+y

end;

writeln sx,sy

end.

* **Программа с добавленными операторами и идентификаторами:**

программа xy;

переменные k,l:целые;

sx,sy,x,h,p,y:вещественные;

начало

sx=0;

sy=0;

цикл k от 0 до 5 шаг 1 выполнять

{

sx:=sy+x+p;

sy:=0;

цикл l от 1 до 10 шаг 1 выполнять

sy:=sy+y+h

};

вывести (sx,sy)

конец.

*Сгенерированный код:*

program xy;

var

k,l:integer;

sx,sy,x,h,p,y:real;

begin

sx:=0;

sy:=0;

for k from 0 to 5 step 1 do

begin

sx:=sy+x+p;

sy:=0;

for l from 1 to 10 step 1 do

sy:=sy+y+h

end;

writeln sx,sy

end.

* **Программа с лексической ошибкой**

прграмма xy;

переменные k,l:целые;

sx,sy,x,y:вещественные;

начало

sx=0;

sy=0;

цикл k от 0 до 5 шаг 1 выполнять

{

sx:=sy+x;

sy:=0;

цикл l от 1 до 10 шаг 1 выполнять

sy:=sy+y

};

вывести (sx,sy)

конец.

В результате анализа выдано следующее сообщение: " неизвестный идентификатор прграмма в строке 1

неизвестный идентификатор xy в строке 1

синтаксическая ошибка в строке 1".

* **Программа с синтаксической ошибкой**

программа xy;

переменные k,l:целые;

sx,sy,x,y:вещественные;

начало

sx=0;

sy=0;

цикл k от 0 до 5 шаг 1 выполнять

{

sx:=sy+x;

sy:=0

цикл l от 1 10 шаг 1 выполнять

sy:=sy+y

};

вывести (sx,sy)

конец.

В результате работы программы выдано сообщение: " синтаксическая ошибка в строке 10

синтаксическая ошибка в строке 15".