Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

Тамбовский государственный технический университет

Кафедра САПР

Курсовая работа

по дисциплине «Лингвистическое и

программное обеспечение САПР»

Выполнила: студент группы

.

Проверила: .

Тамбов 20

1. Пояснительная записка

1.1 Задание на проектирование

Согласно варианту №21, необходимо реализовать трехпроходный транслятор с исходного языка на язык Паскаль:

main( )

{ int i;

double h,b,a,m,n,d,x;

scanf (“%le%le",&a,&b);

h=(b-a)/10;

for(I=0; i<=10;i++)

{ x=a+h\*i;

m=(2\*h\*h-4)/(2\*h);

n=(2–h)/(2+h);

d=m\*h\*h–m\*n;

printf (“\n%le,%le”,x,d);

}

}

Дата принятия задания к исполнению «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_20г.

Содержание

1.Пояснительная записка……………………………………………….................2

1.1 Задание на проектирование………………………………………………..…2

1.2 Содержание…………………………………...……………….…….................3

1.3 Введение……………………………………………………………………….4

1.4 Описание процесса решения программы………………………………........5

1.5 Блок-схема решения……………………………………………………...….10

2.Текст программы……………………………………………………………....11

3. Описание программы…………………………………………………………26

3.1. Назначение и общее описание программы…………………………….....26

3.2. Описание логической структуры программы………………….……..…..26

3.3.Способ обращения к программе………………………………………....…26

4. Описание входных и выходных данных………………………………........27

5. Тестовые примеры работы программы…………………………………......27

1.3 Введение

Приведенная курсовая работа направлена на разработку транслятора с языка указанной структуры и порядка следования операторов на язык «Паскаль».

Основная цель работы – закрепление знаний по разработке лингвистического программного обеспечения, начиная с рассмотрения формальной грамматики языка, операторов, констант и идентификаторов, заканчивая, рассмотрением методов разбора и реализации их в составе лексического и синтаксического анализаторов, генератора готового кода

Разрабатываемый транслятор является трехпроходным, что сказывается на принципе построения и дальнейшего взаимодействия элементов в него входящих, что будет рассмотрено далее.

Лексический анализатор производит перевод указанной последовательности операторов текста в набор обозначенных лексем. Результатом работы синтаксического анализатора является подтверждение ошибки или наоборот, ее отсутствия, в зависимости от правильности задания исходного кода пользователем транслятора. Генератор кода является заключающим звеном взаимодействия элементов – осуществляет генерацию кода внутреннего представления программы в набор лексем конечного языка приведения – «Паскаль»

Окончательным решением курсовой работы будет приложение Windows, разработанное в среде Borland Builder C++.

Помимо пояснительной записки и пояснений по коду программы приложения и методам реализации синтаксических единиц, будет также рассмотрено тестирование приложения на правильность работы и на возможность обнаружения им ошибок исходного входного кода разного уровня сложности.

Дата разработки: « » 20г.

Модуль разработан.

1.4 Описание процесса решения задачи

Для внутреннего представления кода будем использовать замены:  
0 - main

1 - (

2 - )

3 - {

4 - int

5 - ;

6 - double

7 - ,

8 - scanf

9 - &

10 - =

11 - -

12 - /

13 - for

14 - <

15 - +

16 - \*

17 - printf

18 - }

19 - "

20 - %le

21 - \n

22 - идентификатор

23 - константа

Опишем грамматику исходного языка согласно примеру кода, приведенному в постановке задачи проектирования

<программа> → **main(){**<раздел операторов> }

<оператор> →<приведение типов> /<присваивание> / <ввод> /<вывод> /<цикл>**/{/}/;**

<приведение типов> →<тип> <список переменных>

<тип> →**int/double**

<список переменных> → ид {**,** ид}

<присваивание> →ид **=** <арифметическое выражение>

<арифметическое выражение> → <слагаемое> {**+**<слагаемое>}{**-**<слагаемое>}/**(**<слагаемое> **+**<слагаемое> **)**/**(**<слагаемое> **-**<слагаемое> **)**

<слагаемое> → <значение>/<значение> \*<значение>

<значение> →ид / константа

<ввод> → **scanf(**<флаги>,&<список переменных>)

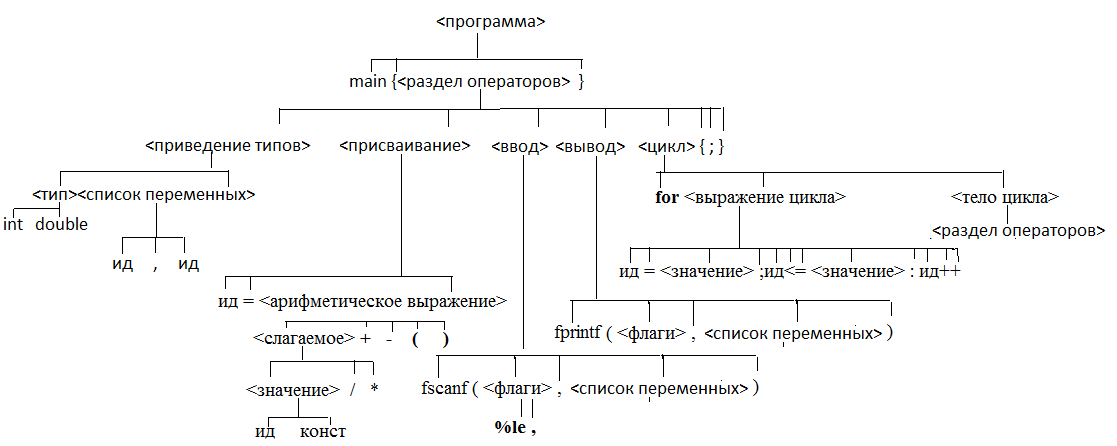
<флаги>→**”%le{%le}**/**{\n}””**

<вывод> → **printf(**<флаги>,<список переменных>**)**

<цикл> → **for** <выражение цикла><тело цикла>

<выражение цикла> → ид=<значение>**;**ид**<** <значение>/ ид**=** <значение>/ ид**<=** <значение>; ид++

<тело цикла> →<оператор>/<раздел операторов>

Далее, построим синтаксическое дерево

Перейдем к заполнению матрицы отношений между операторами, поскольку рассматривается операторного сравнения.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | main | ( | ) | { | Int | ; | , | Double | Scanf | & | = | - | / | for | < | + | \* | printf | } | “ | %le | \n | идентификатор | константа |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | = |  |  |  |  |  |  |  | = |  |  | < | < | < |  |  | < | = |  |  |  |  |  |  |
| 2 |  |  | = |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | > | > |
| 3 |  |  | < |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | < |  |  |  |  |  |
| 4 |  |  |  | < |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  | > |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | < |  |  |  |  |  |
| 6 |  |  |  | < |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | < |  | = |  |
| 8 |  |  |  | < |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  | > |  |  |  | < |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  | > |  |  |  |  |  |  |  |  |  |  |  | = |  |  |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | > | > |
| 12 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | = | = |
| 13 |  |  |  | < |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 15 |  |  | > |  |  |  |  |  |  |  |  |  |  |  |  | = |  |  |  |  |  |  | > | > |
| 16 |  |  | > |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | = | = |
| 17 |  |  |  | < |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 18 |  |  |  | = |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 19 |  |  |  |  |  |  | < |  |  |  |  |  |  |  |  |  |  |  |  | = |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | < |  | = |  |  |
| 21 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | = |  |  |  |
| 22 |  |  |  | < | < | < | = | < |  | = | < | < | = |  |  |  | = |  |  |  |  |  |  |  |
| 23 |  |  |  | < |  |  |  |  |  |  | < | < | = |  |  |  | = |  |  |  |  |  |  |  |

Согласно указанному методу, осуществляется сравнение соседних лексем ведется до тех пор, пока не будет найдена часть кода, обособленная отношениями меньше-больше, которая разбирается синтаксически и выбрасывается из дальнейшего, повторного циклически, рассмотрения. Лексемы не имеющие, согласно матрице, отношения не могут быть употреблены совместно парами в исходном тексте данных. Это будит использовано при обнаружении ошибок в ходе разбора лексического анализатора.

Программный интерфейс представляет собой окно с разделенными полями вывода данных из файла с исходным текстом и результатом трансляции. Поле файла исходного текста допускает редактирование данных, что отбрасывает необходимость существования файла исходных данных, т.е. исходные данные могут вводится в само поле непосредственно и не извне. Программа допускает возможность открытия файла исходных данных средствами стандартного проводника Windows, аналогичную по структуре возможность сохранения результата работы. Все эти средства позволяют пользователю использовать любые имена входных и выходных файлов данных, исключая случай фиксированного набора имен файлов, что делает программу более гибкой функционально и удобной к пользованию. Также, имеется возможность просмотра данных работы программы, посредством кнопки «Данные», автоматически средствами ОС открывается текстовый файл данных о программе, метода анализа, лексем исходного текста, идентификаторов и т.п.

Схема лексического анализатора: исходные данные считываются из построчно соответствующего поля интерфейса программы (здесь, как было сказано выше, данные могут быть прогружены с .txt файла любого каталога или введены пользователем непосредственно в поле) параллельно с процедурой сравнения начала текущей считанной строки (массив Buffer программы) с соответствующими лексемами из внутреннего представления (двумерный массив leksemy). При условии совпадения текущей единицы строки с какой-либо лексемой таблицы символов происходит загрузка текущего кода лексемы в массив строки программы (строка StrokaPredstav). Если такого не произошло, на этапе лексического анализа неизвестная лексема помечается как constID и анализируется на предмет вхождения символов кириллицы/латиницы (т.е. нашелся символ не число). Если такое вхождение есть, то текущая лексема – идентификатор (ограничение по длине 10 символов), если все элементы constID есть цифры, то эта константа. Далее производится соответствующая запись в строку части исходной программы номера лексемы и ее порядкового номера в программе (номера могут и совпадать при абсолютном совпадении элементов), а также запись имени идентификатора или числового значения константы для последующего их извлечения в процессе генерации кода и анализа ошибок в массивы Ident и Const соответственно. Стоит также упомянуть о файле "внутр\_пред.txt", в который происходит запись таблицы идентификаторов и хода кодов лексем программы по мере ее разбора Таким образом, по завершению работы лексического анализатора сформулированы необходимые массивы и ничто не мешает перейти к анализу правильности содержимого массивов и генерации кода

Согласно заданию на курсовую работу, синтаксический анализатор работает по принципу операторного предшествования. На вход синтаксического анализа поступает строка с частью кода исходных данных, представленная, соответствующей лексическому анализатору, кодировкой лексем. Каждый код лексемы из массива StrokaPredstav анализируется на предмет логического отношения с соседним посредством сдвига указателя строки на единицу влево и получения из матрицы отношения лексем (двумерный массив matrizaOtnoshenij задан на этапе разработки программы) по строке соответствующей текущему значению кода лексемы и столбцу соответствующему коду соседней слева лексемы. Полученный элемент матрицы указывает отношение соседних лексем (больше, меньше или равно, в коде реализовано: >,<,=) Если такового отношения нет, то лексемы недопустимы к употреблению парно. Разбор ведется до тех пор, пока не получена часть кода программы, обособленный отношениями меньше-больше, эта часть анализируется на предмет ошибок грамматики и в положительном исходе выбрасывается из рассмотрения. Далее, программа циклически повторяет процедуру лексического анализа, считывая новую строку текста, отправляя данные результата анализатора на синтаксический анализ до тех пор, пока не будут разобраны все отношения между лексемами

Отметим, что в случае ошибки при синтаксическом анализе, глобальная переменная-флаг globalError получает единичное значение и обрывает исполнение программного кода анализатора с информированием пользователя об ошибке, строке, в которой она встретилась и ее содержании.

Особая роль отводится ошибкам описания идентификаторов, а именно: повторное его описание или неописанный идентификатор. Будем использовать некое подобие алгоритма хеш-адресации, но сразу оговорим такой момент, что в ходе трансляции, в отличии от компиляции, исходного заданного текста программы не важна регистрация типов как таковых и значений переменных, важен лишь тот факт, было ли описание переменной до ее первого использования, как идентификатора и была ли описана переменная раннее при ее описании и приведении типов. Таким образом, задаем массив ObjavlPerem в который записываются идентификаторы по мере их объявления в программе по следующему принципу: если анализируется строка объявления идентификатора, просматривается массив ObjavlPerem, если переменная была объявлена раннее, то в массиве имеется соответствующая запись кода идентификатора (согласно лексическому представлению) и имеет место ошибка, как повторное описание идентификаторов, иначе, идентификатора нет в списке и справедливо будет его добавить в ObjavlPerem, обозначив, тем самым, его описание для дальнейшего анализа. По мере просмотра заданного текста программы, каждый встретившийся идентификатор проверяется на наличие в массиве объявленных переменных и, если речь не идет об объявлении идентификаторов, а, например, выполняется арифметическая операция, при отсутствии идентификатора в ObjavlPerem будет выведена ошибка «неописанный идентификатор». Таким образом, как было сказано выше, алгоритм напоминает метод хеш-таблиц, только исключена регистрации типов и значений переменных, а хэш-функция представляет собой лишь порядковый номер идентификатора, полученный при лексическом анализе.

Что касается генерации кода, этот процесс происходит сразу после выхода с синтаксического анализатора. Как было сказано выше, по причине равности уровней языков входного и выходного (язык Паскаль) текста данных, внутреннее представление программы ограничится последовательностью кодов лексем, фактически, результатом работы лексического анализатора при отсутствии ошибок при синтаксическом анализе.

Поскольку искомый транслятор должен быть трехпроходным, данные сохраняются в битовый массив с выхода лексического анализатора и передаются, при успешном анализе, в генератор кода

Для непосредственного оформления программы будем использовать конструктор Borland Builder C++ 6.0, поддерживаемый на 64-разрядных системах Компиляция искомого приложения осуществляется без временных библиотек и стандартных пакетов Builder C++, что дает возможность беспрепятственно использовать приложение на машинах без установленной среды разработки, делая программу универсальной и независимой от имеющихся ресурсов.

* 1. Блок-схема решения



2. Текст программы

#include <vcl.h>

#pragma hdrstop

#include "string.h"

#include "stdio.h"

#include "stdlib.h"

#include "conio.h"

#include "Unit1.h"

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

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

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

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

: TForm(Owner)

{

}

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

int globalError=0;//ïåðåìåííàÿ çàïðåùàþùàÿ äàëüíåéøèå äåéñòâèÿ ïðè âîçíèêíîâåíèè îøèáêè

FILE\*vnutrPREDSTAVLENIE;//ôàéë âíóòðåííåãî ïðåäñòàâëåíèÿ

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

{

Memo1->Lines->Clear();

Memo2->Lines->Clear();//î÷èùàåì ïîëÿ âûâîäà èíôîðìàöèè ïðè çàãðóçêå ôîðìû

}

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

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

{

if(OpenDialog1->Execute())

Memo1->Lines->LoadFromFile(OpenDialog1->FileName);//êíîïêà "Îòêðûòü"

}

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

void \_\_fastcall TForm1::Button3Click(TObject \*Sender)

{

if(SaveDialog1->Execute())

{char buffer[500];

sprintf(buffer,"%s.txt",SaveDialog1->FileName);

Memo2->Lines->SaveToFile(buffer);//êíîïêà "Ñîõðàíèòü êàê..."

}}

//-------

**- Функцмя сокращения строки**

char socrStr(char \*s1,int n)//функция вырезает n первых символов строки, поступающей на вход

{char a[25];

int i;

i=strlen(s1)-n;

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

a[j]=s1[j+n];

strcpy(s1,a);

s1[i+1]='\0';

return(\*s1);}

- Обозначение переменных массива лексем, матрицы предшествования

globalError=0;

int i,j,f=0,Flag=0,Flag1=0,nomerStroki=0,nomerConst=-1,nomerID=-1,dlinaStroki=0,nomerPeremen=-1,War=0,Dlina[100];//nomerStroki---текущее положение строки

char Ident[15][10],Const[15][10];//массив переменных(идентификаторы) и числовых констант

int ObjavlPerem[40];//массив обьявленных идентификаторов, нужно для исключения вариантов ошибки

char leksemy[15][22];//массив лексем исходного текста

sprintf(leksemy[0],"main");

sprintf(leksemy[1],"(");

sprintf(leksemy[2],")");

sprintf(leksemy[3],"{");

sprintf(leksemy[4],"int");

sprintf(leksemy[5],";");

sprintf(leksemy[6],"double");

sprintf(leksemy[7],",");

sprintf(leksemy[8],"scanf");

sprintf(leksemy[9],"&");

sprintf(leksemy[10],"=");

sprintf(leksemy[11],"-");

sprintf(leksemy[12],"/");

sprintf(leksemy[13],"for");

sprintf(leksemy[14],"<");

sprintf(leksemy[15],"+");

sprintf(leksemy[16],"\*");

sprintf(leksemy[17],"printf");

sprintf(leksemy[18],"}");

sprintf(leksemy[19],"\"");

sprintf(leksemy[20],"");

strcat(leksemy[20],"%le");

sprintf(leksemy[21],"\\n");

char matrizaOtnoshenij[22][22];

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

for(j=0;j<22;j++)

matrizaOtnoshenij[i][j]='n';// забиваем матрицу специальным символом, означающим отсутствие отношения между лексемами

matrizaOtnoshenij[0][1]='=';

matrizaOtnoshenij[1][2]='=';

matrizaOtnoshenij[3][4]='<';

matrizaOtnoshenij[3][6]='<';

matrizaOtnoshenij[3][8]='<';

matrizaOtnoshenij[3][13]='<';

matrizaOtnoshenij[3][17]='<';

matrizaOtnoshenij[3][18]='<';

matrizaOtnoshenij[3][22]='<';

matrizaOtnoshenij[3][23]='<';

matrizaOtnoshenij[4][22]='<';

matrizaOtnoshenij[22][7]='=';

matrizaOtnoshenij[7][22]='=';

matrizaOtnoshenij[22][5]='>';

matrizaOtnoshenij[6][22]='<';

matrizaOtnoshenij[8][1]='=';

matrizaOtnoshenij[1][19]='<';

matrizaOtnoshenij[1][22]='<';

matrizaOtnoshenij[1][13]='<';

matrizaOtnoshenij[19][20]='=';

matrizaOtnoshenij[19][19]='=';

matrizaOtnoshenij[20][7]='=';

matrizaOtnoshenij[7][20]='=';

matrizaOtnoshenij[20][19]='=';

matrizaOtnoshenij[19][7]='>';

matrizaOtnoshenij[1][19]='<';

matrizaOtnoshenij[7][9]='<';

matrizaOtnoshenij[9][22]='=';

matrizaOtnoshenij[22][11]='<';

matrizaOtnoshenij[22][16]='<';

matrizaOtnoshenij[22][15]='=';

matrizaOtnoshenij[23][11]='<';

matrizaOtnoshenij[23][16]='<';

matrizaOtnoshenij[23][15]='=';

matrizaOtnoshenij[11][1]='<';

matrizaOtnoshenij[16][1]='<';

matrizaOtnoshenij[15][1]='<';

matrizaOtnoshenij[2][11]='>';

matrizaOtnoshenij[2][16]='>';

matrizaOtnoshenij[2][15]='>';

matrizaOtnoshenij[2][5]='>';

matrizaOtnoshenij[22][10]='<';

matrizaOtnoshenij[10][22]='<';

matrizaOtnoshenij[10][23]='>';

matrizaOtnoshenij[3][3]='=';

matrizaOtnoshenij[18][18]='=';

matrizaOtnoshenij[10][1]='<';

matrizaOtnoshenij[11][22]='=';

matrizaOtnoshenij[11][1]='<';

matrizaOtnoshenij[10][23]='=';

matrizaOtnoshenij[2][12]='>';

matrizaOtnoshenij[2][15]='>';

matrizaOtnoshenij[2][16]='>';

matrizaOtnoshenij[12][22]='<';

matrizaOtnoshenij[12][23]='<';

matrizaOtnoshenij[12][1]='<';

matrizaOtnoshenij[13][1]='=';

matrizaOtnoshenij[22][14]='=';

matrizaOtnoshenij[14][10]='=';

matrizaOtnoshenij[5][22]='<';

matrizaOtnoshenij[5][23]='<';

matrizaOtnoshenij[15][15]='=';

matrizaOtnoshenij[15][2]='>';

matrizaOtnoshenij[15][22]='<';

matrizaOtnoshenij[15][23]='<';

matrizaOtnoshenij[16][22]='<';

matrizaOtnoshenij[16][23]='<';

matrizaOtnoshenij[15][22]='<';

matrizaOtnoshenij[1][23]='<';

matrizaOtnoshenij[17][1]='=';

matrizaOtnoshenij[19][21]='<';

matrizaOtnoshenij[21][7]='>';

matrizaOtnoshenij[21][19]='>';

matrizaOtnoshenij[21][20]='=';

matrizaOtnoshenij[20][21]='=';

**Блок лексического анализа**

{if(globalError==0){

sprintf(Buffer1,"%s",Memo1->Lines->Strings[nomerStroki]);

j=-1;

for(i=0;i<strlen(Buffer1);i++)

if(Buffer1[i]!=' ')

{j++; Buffer[j]=Buffer1[i];}//удаляем пробелы в строке. готовим ее к лексическому анализатору

Buffer[j+1]='\0';

/////////////////////////////

j=-1;

vnutrPREDSTAVLENIE=fopen("внутр\_пред.txt","a+t");

while(strlen(Buffer)>0){

Flag=0,Flag1=0;

for(i=0;i<=21;i++)

if(strncmp(Buffer,leksemy[i],strlen(leksemy[i]))==0){Flag=1;break;}

if(Flag==1)

{j++;

StrokaPredstav[nomerStroki][j]=i;

socrStr(Buffer,strlen(leksemy[i]));

fprintf(vnutrPREDSTAVLENIE,"%d ",StrokaPredstav[nomerStroki][j]);

}

else{int m=-1;

Flag1=0;

while(Flag1==0&&strlen(Buffer)>0)

{m++;

constID[m]=Buffer[0];

socrStr(Buffer,1);

if(strlen(Buffer)>0){

for(i=0;i<=21;i++)

if(strncmp(Buffer,leksemy[i],strlen(leksemy[i]))==0){Flag1=1;break;}

}}

constID[m+1]='\0';

Flag1=0;

for(i=0;i<strlen(constID);i++) // выясняем, идентификатор или же константа неподходящая лексемма

if(constID[i]!='–'&&constID[i]!='0'&&constID[i]!='1'&&constID[i]!='2'&&constID[i]!='3'&&constID[i]!='4'&&constID[i]!='5'&&constID[i]!='6'&&constID[i]!='7'

&&constID[i]!='8'&&constID[i]!='9'){Flag1=1;break;}

if(Flag1==0)//это константа

{nomerConst++;sprintf(Const[nomerConst],"%s",constID);j++;

StrokaPredstav[nomerStroki][j]=23;j++;//23 - номер константы в таблице лексем

StrokaPredstav[nomerStroki][j]=nomerConst;

fprintf(vnutrPREDSTAVLENIE,"%d %d ",StrokaPredstav[nomerStroki][j-1],StrokaPredstav[nomerStroki][j]);

}

else//это идентификатор

{j++;

StrokaPredstav[nomerStroki][j]=22;j++;//22 - номер идентификатора в таблице лексем

Flag1=0;

for(i=0;i<=nomerID;i++)

if(strcmp(Ident[i],constID)==0){Flag1=1;break;}//он уже у нас встречался

if(Flag1==1) StrokaPredstav[nomerStroki][j]=i;

else

{nomerID++;sprintf(Ident[nomerID],"%s",constID);

StrokaPredstav[nomerStroki][j]=nomerID;}

fprintf(vnutrPREDSTAVLENIE,"%d %d ",StrokaPredstav[nomerStroki][j-1],StrokaPredstav[nomerStroki][j]);

}

}

}

dlinaStroki=j;

fprintf(vnutrPREDSTAVLENIE,"\n");

fclose(vnutrPREDSTAVLENIE);

Dlina[nomerStroki]=dlinaStroki;

}}

ShowMessage("Блок внутреннего представления успешно завершил работу");

/////////////////////////////////////////////////////!!!!!конец работы блока внутреннего представления

int NACH=0,KON=0,CN=0,CK=0,PER=0;

/////////////////////////////////////////////////////!!!!!начало работы блока синтаксически-лексического анализатора

for(nomerStroki=0; nomerStroki<Memo1->Lines->Count; nomerStroki++)

{if(globalError==0){

// ShowMessage(IntToStr(Dlina[nomerStroki]));

dlinaStroki=Dlina[nomerStroki];

/////////////////////////////////////////////////////!!!!!

if(nomerStroki==0)

if(StrokaPredstav[nomerStroki][0]!=0||StrokaPredstav[nomerStroki][1]!=1||StrokaPredstav[nomerStroki][2]!=2)

{ShowMessage("Неправильная конструкция заголовка main() программы");globalError=1;}///проверка заголовка

/////////////////

if(globalError==0){

Flag1=0;

for(i=0;i<=dlinaStroki;i++)

if(StrokaPredstav[nomerStroki][0]==4||StrokaPredstav[nomerStroki][0]==6) {Flag1=1;break;}

if(Flag1==1)

{

int Flag2=0;

for(i=0;i<=dlinaStroki;i++)

{Flag2=0;

if(StrokaPredstav[nomerStroki][i]==22&&globalError==0)

{for(j=0;j<=nomerPeremen;j++)

if(StrokaPredstav[nomerStroki][i+1]==ObjavlPerem[j])

{Flag2=1;break;}

if(Flag2==1){globalError=1;

sprintf(Buffer1,"Ошибка\nповторное описание идентификатора %s \nв строке %d",Ident[ObjavlPerem[j]],nomerStroki+1);

ShowMessage(Buffer1);}

else{nomerPeremen++;ObjavlPerem[nomerPeremen]=StrokaPredstav[nomerStroki][i+1];}

}}///проверка на повторное описание идентификатора }}

Блок синтаксического анализа

/////проверка на необъявленные идентификаторы

if(globalError==0&&nomerStroki!=0)

{Flag1=0;

for(i=0;i<=dlinaStroki;i++)

if(StrokaPredstav[nomerStroki][0]==4||StrokaPredstav[nomerStroki][0]==6) {Flag1=1;break;}

if(Flag1==0)

{for(i=0;i<=dlinaStroki;i++)

{Flag2=0;

if(StrokaPredstav[nomerStroki][i]==22&&globalError==0)

{for(j=0;j<=nomerPeremen;j++)

if(StrokaPredstav[nomerStroki][i+1]==ObjavlPerem[j])

{Flag2=1;break;}

if(Flag2==0){globalError=1;

sprintf(Buffer1,"Ошибка\n неописанный идентификатор %s \nв строке %d",Ident[StrokaPredstav[nomerStroki][i+1]],nomerStroki+1);

ShowMessage(Buffer1);}

}}}}///проверка на наличие неописанные идентификаторов

///////////////////////////////////////////////////////////////////////////////

//все идентификаторы проверили, перейдем к рассмотрению синтаксиса

///////////////////////////////////////////////////////////////////////////////!!!

int dlinaBuffer=-1;//рассматриваем строку буффер без уточнения идентификаторов и констант

int Buffer3[40];

if(globalError==0){

for(i=0;i<=dlinaStroki;i++)

if(StrokaPredstav[nomerStroki][i]!=22&&StrokaPredstav[nomerStroki][i]!=23)

{dlinaBuffer++; Buffer3[dlinaBuffer]=StrokaPredstav[nomerStroki][i];}

else{dlinaBuffer++; Buffer3[dlinaBuffer]=StrokaPredstav[nomerStroki][i];i++;}

}

/////////////////////////

//for(i=0;i<=dlinaBuffer;i++)

//ShowMessage(IntToStr(Buffer3[i]));

if(globalError==0){

float f=0;

for(i=0;i<dlinaBuffer;i++)

{if(matrizaOtnoshenij[Buffer3[i]][Buffer3[i+1]]=='n')

{globalError=1;

sprintf(Buffer1,"Ошибка\n неверная последовательность лексем \nв строке %d",nomerStroki+1);

ShowMessage(Buffer1);break;}

if(matrizaOtnoshenij[Buffer3[i]][Buffer3[i+1]]=='<') f++;

}}

if(globalError==0){

int l=0,r=0;

for(i=0;i<=dlinaBuffer;i++)

{if(Buffer3[i]==1) l++;

if(Buffer3[i]==2) r++;}

if(l!=r){globalError=1;sprintf(Buffer1,"Ошибка\n незакрытая скобка \nв строке %d",nomerStroki+1);

ShowMessage(Buffer1);break;}

}

//////////////////////

if(globalError==0){

Flag1=0;

for(i=0;i<=dlinaBuffer;i++)

if(Buffer3[i]==0||Buffer3[i]==3||Buffer3[i]==18||Buffer3[i]==13) Flag1=1;

if(Flag1==0&&Buffer3[dlinaBuffer]!=5){globalError=1;sprintf(Buffer1,"Ошибка\n пропущен оператор ; \nв строке %d",nomerStroki+1);

ShowMessage(Buffer1);break;}

}

//////////////////

if(globalError==0&&nomerStroki==Memo1->Lines->Count-1){

if(Buffer3[dlinaBuffer]!=18){globalError=1;sprintf(Buffer1,"Ошибка\n не найден оператор конца программы } \nв строке %d",nomerStroki+1);

ShowMessage(Buffer1);break;}

}

//////

if(globalError==0&&nomerStroki==1&&Buffer3[0]!=3){

globalError=1;sprintf(Buffer1,"Ошибка\n не найден оператор начала программы { \nв строке %d",nomerStroki+1);

ShowMessage(Buffer1);break;}

//////////////////

if(globalError==0){

for(i=0;i<=dlinaBuffer;i++)

{if(Buffer3[i]==3) NACH++; if(Buffer3[i]==18) CK++;}

}

if(globalError==0){

if(Buffer3[0]==13){

int uk=0,tz=0;

for(i=0;i<=dlinaBuffer;i++)

{if(Buffer3[i]==22) uk++; if(Buffer3[i]==5) tz++;}

if(uk!=3||tz!=2){

globalError=1;sprintf(Buffer1,"Ошибка\n не правильное выражение цикла \nв строке %d",nomerStroki+1);

ShowMessage(Buffer1);break;}

}

}

//////////////////////

}}

**Блок генератора кода**

if(globalError==0)

ShowMessage("Пуск генератора кода");

if(globalError==0){ bool BEGINFLAG=false;

//////////////////!!!!!!!!!!!!!!!!!!!!!

Memo2->Lines->Add("program Result;");

Memo2->Lines->Add("uses crt;");

Memo2->Lines->Add("var");

for(nomerStroki=1; nomerStroki<Memo1->Lines->Count; nomerStroki++){

dlinaStroki=Dlina[nomerStroki];

sprintf(Buffer2,"");

//aaaaa

for(i=0;i<=dlinaStroki;i++){

///

if(StrokaPredstav[nomerStroki][0]==13) {strcat(Buffer2,"for ");

strcat(Buffer2,Ident[StrokaPredstav[nomerStroki][3]]);

strcat(Buffer2,":=");

if(StrokaPredstav[nomerStroki][5]==22)strcat(Buffer2,Ident[StrokaPredstav[nomerStroki][6]]);

else strcat(Buffer2,Const[StrokaPredstav[nomerStroki][6]]);

strcat(Buffer2," to ");

if(StrokaPredstav[nomerStroki][11]==22)strcat(Buffer2,Ident[StrokaPredstav[nomerStroki][12]]);

if(StrokaPredstav[nomerStroki][11]==23)strcat(Buffer2,Const[StrokaPredstav[nomerStroki][12]]);

if(StrokaPredstav[nomerStroki][12]==22)strcat(Buffer2,Ident[StrokaPredstav[nomerStroki][13]]);

if(StrokaPredstav[nomerStroki][12]==23)strcat(Buffer2,Const[StrokaPredstav[nomerStroki][13]]);

strcat(Buffer2," do ");

Memo2->Lines->Add(Buffer2);

sprintf(Buffer2,"");

nomerStroki++;

}

///

if(StrokaPredstav[nomerStroki][i]==22) {i++;strcat(Buffer2,Ident[StrokaPredstav[nomerStroki][i]]);i++;}

if(StrokaPredstav[nomerStroki][i]==23) {i++;strcat(Buffer2,Const[StrokaPredstav[nomerStroki][i]]);i++;}

if(StrokaPredstav[nomerStroki][i]==3) {if(BEGINFLAG==true){strcat(Buffer2,"begin ");}}

if(StrokaPredstav[nomerStroki][i]==17) {strcat(Buffer2,"write");}

if(StrokaPredstav[nomerStroki][i]==10) {strcat(Buffer2,":=");}

if(StrokaPredstav[nomerStroki][i]==11) {strcat(Buffer2,"-");}

if(StrokaPredstav[nomerStroki][i]==12) {strcat(Buffer2,"/");}

if(StrokaPredstav[nomerStroki][i]==14) {strcat(Buffer2,"<");}

if(StrokaPredstav[nomerStroki][i]==15) {strcat(Buffer2,"+");}

if(StrokaPredstav[nomerStroki][i]==16) {strcat(Buffer2,"\*");}

if(StrokaPredstav[nomerStroki][i]==8) {strcat(Buffer2,"read");}

if(StrokaPredstav[nomerStroki][i]==1) {strcat(Buffer2,"(");}

if(StrokaPredstav[nomerStroki][i]==2) {strcat(Buffer2,")");}

if(StrokaPredstav[nomerStroki][i]==7&&StrokaPredstav[nomerStroki][i-1]!=19) {strcat(Buffer2,",");}

if(StrokaPredstav[nomerStroki][i]==5&&StrokaPredstav[nomerStroki][0]!=4&&StrokaPredstav[nomerStroki][0]!=6

&&StrokaPredstav[nomerStroki][dlinaStroki]!=3&&StrokaPredstav[nomerStroki][dlinaStroki]!=18

&&StrokaPredstav[nomerStroki][0]!=13) {strcat(Buffer2,";");}

if(StrokaPredstav[nomerStroki][i]==20&&StrokaPredstav[nomerStroki][i+1]==7) i++;

if(StrokaPredstav[nomerStroki][i]==7&&StrokaPredstav[nomerStroki][i+1]==9) i++;

if(StrokaPredstav[nomerStroki][i]==18)

if(nomerStroki==Memo1->Lines->Count-1) {strcat(Buffer2,"end.");}

else {strcat(Buffer2,"end;");}

}

///aaa

if(StrokaPredstav[nomerStroki][0]==4) {strcat(Buffer2,":integer;");}

if(StrokaPredstav[nomerStroki][0]==6) {strcat(Buffer2,":real;");}

if(BEGINFLAG==false&&(StrokaPredstav[nomerStroki][0]!=4&&StrokaPredstav[nomerStroki][0]!=6

&&StrokaPredstav[nomerStroki][0]!=3))

{Memo2->Lines->Add("begin");BEGINFLAG=true;}

Memo2->Lines->Add(Buffer2);

}

3. Описание программы

3.1. Назначение и общее описание программы

Программа разработана как транслятор с представленного языка на язык Pascal. Реализована методика отыскания лексических и синтаксических ошибок с соответствующим оповещением пользователя

Программа демонстрирует список лексем с их числовым кодом, принятым внутри кода, список используемых идентификаторов и списком соответствующих констант

Программа представляет приложение Windows Project.exe с реализуемым пользовательским интерфейсом

Программа создана на языке программирования C++ на компиляторе Borland Builder C++ 6.0 позволяющим создавать приложения, совместимые с 64-разрядными системами

3.2. Описание логической структуры программы

Как было описано выше, транслятор работает по следующей схеме:

--- открытие исходного текста

--- лексический анализ

--- запуск синтаксического анализатора и генератора кода

На этом этапе происходит одновременное выявление ошибок и генерация кода. В случае нахождения таковой, программа обрывает свою работу, завершая и генератор кода, сообщая соответствующую ошибку пользователю

3.3.Способ обращения к программе

Программа запускается при открытии приложения Project.exe при условии наличия текстового документа транслируемой программы расширения.txt в любом каталоге.

Что касается файла результата, это – файл расширения .txt Его наличие на момент запуска транслятора не нужно, поскольку он создается автоматически по желанию пользователя, при сопровождении стандартного проводника сохранения файлов Windows.

При допущении пользователем ошибки ввода, поступит соответствующее сообщение пользователю содержащее возможные пути решения проблемы.

Коррекция текста программы исходного кода допустима в самом окне приложения

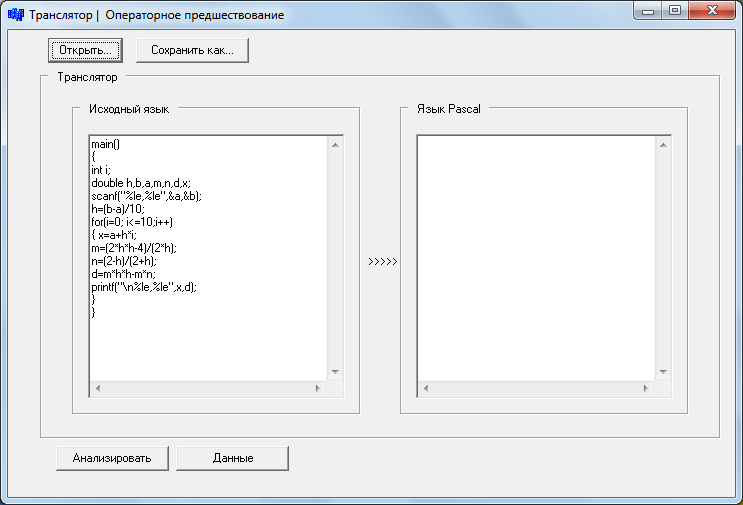
4. Описание входных и выходных данных

Обязательно наличие транслируемого файла. Расположение лексем файла (строки, пробелы) не имеет значения

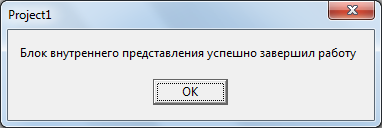
Файл выходных данных, как было сказано выше, может отсутствовать на момент запуска транслятора, т.к. он создается по желанию пользователя, а все данные результата операции трансляции выводится в поле результата приложения автоматически.

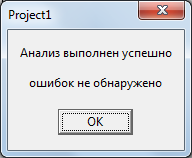
5. Тестовые примеры работы программы

Откроем исходный код

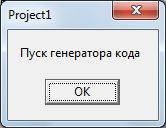


Проанализируем код, как видно, ошибок нет

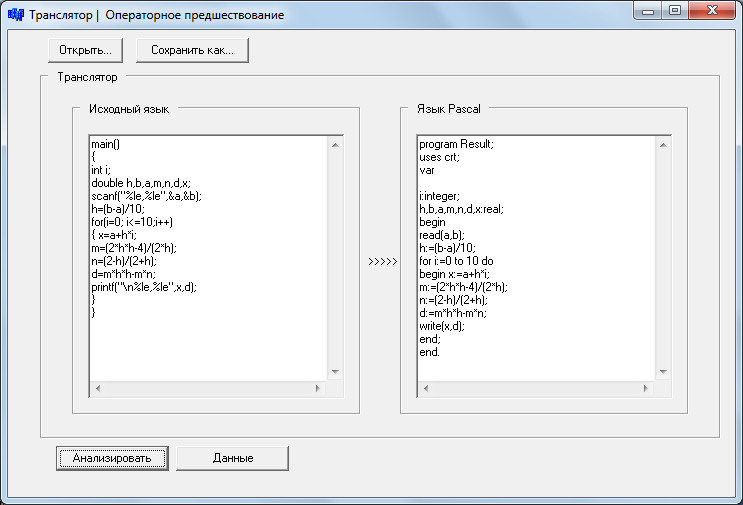




Все три блока успешно отработали

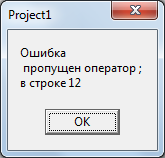


Выходной код имеет вид:



Испытаем программу в случае ошибок. Уберем знак ;

Как видим. Первый проход отработан удачно, а блок синтаксического анализа показал ошибку



Теперь рассмотрим ошибки на неописанный идентификатор

