Міністерство освіти і науки України

Національний технічний університет України

"Київський політехнічний інститут"

Факультет прикладної математики

Кафедра спеціалізованих комп’ютерних систем

Розрахунково-графічна робота

Дисципліна:

"Основи проектування трансляторів"

Тема:

" Розробка синтаксичного аналізатора (parser)"

Варіант 14

|  |  |
| --- | --- |
| Виконав  Студент групи КВ-92 | Перевірив  Марченко О.І. |

Сміюн В.В.

Київ 2011

Постановка задачі

Розробити програму синтаксичного аналізатора (СА) для підмножини мови програмування SIGNAL.

Програма має забезпечувати наступне:

* читання рядку лексем та таблиць, згенерованих лексичним аналізатором, який було розроблено в лабораторній роботі №1;
* синтаксичний аналіз (розбір) програми, поданої рядком лексем (алгоритм розбору вибирається згідно з варіантом);
* побудову дерева розбору;
* формування таблиць ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;
* формування лістингу вхідної програми з повідомленнями про лексичні та синтаксичні помилки.

**Входом** СА має бути наступне:

* закодований рядок лексем;
* таблиці ідентифікаторів, числових, символьних та рядкових констант (якщо це передбачено граматикою варіанту), згенеровані лексичним аналізатором;
* вхідна програма на підмножині мови програмування SIGNAL згідно з варіантом (необхідна для формування лістнигу програми).

**Виходом** СА має бути наступне:

* дерево розбору вхідної програми;
* таблиці ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;
* лістинг вхідної програми з повідомленнями про лексичні та синтаксичні помилки.

**Індивідуальне завдання:**

* низхідний розбір за алгоритмом рекурсивного спуску

**Тест програма:**

PROGRAM ONE;

LABEL 10;

BEGIN

56:

IF IDN = 22 THEN

GOTO 56;

ELSE

GOTO 20;

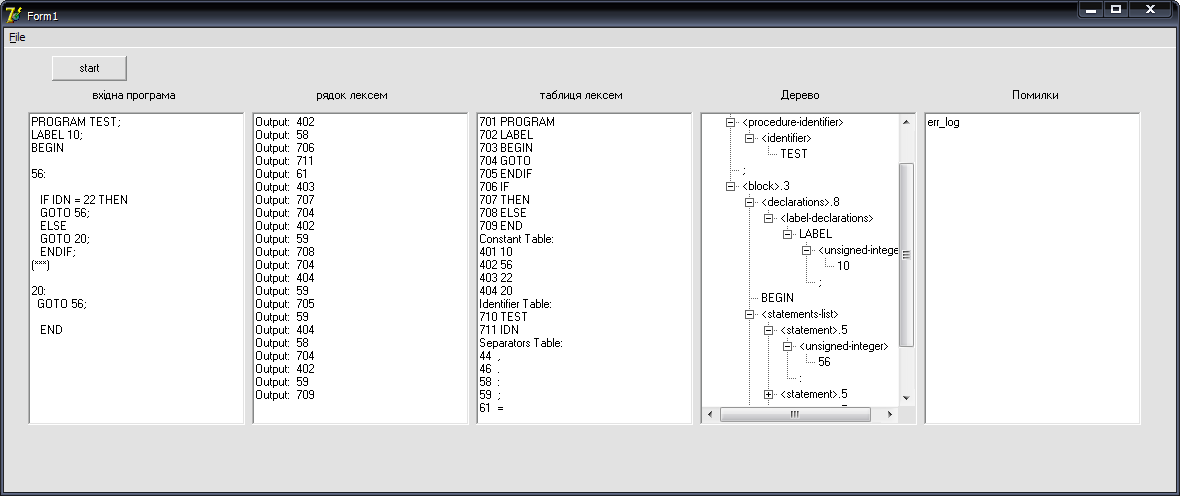
ENDIF;

20:

GOTO 56;

END

(\*\*\*)

****

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Menus, StdCtrls, CheckLst, ComCtrls;

type

Tabl= record

key: string;

number: integer;

end;

TSymbol = record

value: Char;

attr: Byte;

end;

TForm1 = class(TForm)

OpenDialog1: TOpenDialog;

MainMenu1: TMainMenu;

Ffle1: TMenuItem;

Open1: TMenuItem;

Exit1: TMenuItem;

StaticText1: TStaticText;

StaticText2: TStaticText;

StaticText3: TStaticText;

Button1: TButton;

Memo1: TMemo;

Memo2: TMemo;

tree: TTreeView;

Label1: TLabel;

Label2: TLabel;

err\_log: TMemo;

Memo3: TMemo;

procedure Open1Click(Sender: TObject);

procedure Button1Click(Sender: TObject);

function Gets: TSymbol;

procedure FillAttributes;

Procedure ConstTabSearch(bf1:string;Te:integer; var num:integer;var C:boolean);

Procedure IdnTabSearch(bf2:string; Te:integer; var num:integer; var IP:boolean);

Procedure KeyTabSearch(bf:string; var num:integer; var K:boolean);

Procedure ConstTabForm(bf3:string;num:integer);

Procedure IdnTabForm(bf4:string;num:integer);

//parser

Procedure ParserIdnSearch(var bf2:string; k:integer; var B:boolean);

Procedure ParserConstSearch(bf1:string; var num:integer;var C:boolean);

Procedure idnt(var tree:TTreeView; s2:string; var Node:TTreeNode);

Procedure cond(var TNode: TTReeNode; var sind:integer; var tree:TTreeView);

Procedure statements(var TNode: TTreeNode; var sind: integer; var tree:TTreeView);

Procedure label\_decl(var TNode:TTreeNode; var sind:integer; var tree:TTreeView);

Procedure declarations(var TNode:TTreeNode; var sind:integer; var tree:TTreeView);

Procedure block(var TNode:TTreeNode;var sind:integer;var tree:TTreeView);

Procedure procedure\_identifier(var node:TTreeNode; var sind:integer; var tree:TTreeView);

Procedure signal\_program(var tree:TTReeView; var sind:integer);

Procedure signal(var tree:TTReeView);

//code generator

procedure main();

procedure Exit1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

Attributes: array [1..256] of Byte;

symbol: TSymbol;

lexCode: Word;

jmpcount,gtcnt:byte;

buf, buf1: string;

fname: TFileName;

SuppressOutput,CT,KT,IT,fl: Boolean;

FINP,FOUT, tablf, TREEF: TextFile;

char,chIdN, chKT:integer;

i, ind, col, lin, sind:integer;

j,CTE,IDN,ch:byte;

ifflag,thenflag,elseflag,errflag:boolean;

KeyTabl:array[1..9] of Tabl;

ConstTabl:array[1..99] of Tabl;

IdnTabl:array[1..99] of Tabl;

lexstr: array[1..1000] of integer;

lexpos: array[1..1000] of integer;

node: TTreenode;

implementation

{$R \*.dfm}

procedure TForm1.Open1Click(Sender: TObject);

begin

// створення об'єкту OpenDialog - назначення на нашу змінну OpenDialog

openDialog1 := TOpenDialog.Create(self);

// Установка початкового каталогу, щоб зробити його поточним

openDialog1.InitialDir := GetCurrentDir;

// Тільки дозволені існуючі файли можуть бути вибрані

openDialog1.Options := [ofFileMustExist];

// Дозволено вибирати тільки .txt і .sig файли

openDialog1.Filter :=

'text files|\*.txt|signal files|\*.sig';

// Вибір файлів мови Signal за замовчуванням

openDialog1.FilterIndex := 2;

// Показ відкриття діалогу

if openDialog1.Execute

then begin

ShowMessage('File : '+openDialog1.FileName);

fname :=openDialog1.FileName;

end

else ShowMessage('Відкриття відмінено');

// Звільнення діалогу

openDialog1.Free;

end;

function TForm1.Gets: TSymbol;

begin

if EOLN(FINP) then

inc(lin);

Read(FINP, Result.value);

ch:= Ord(Result.value);

Result.attr := Attributes[ch];

end;

//Опис таблиць

procedure TForm1.FillAttributes;

var i: integer;

begin

//error

for i:=1 to 256 do

Attributes[i]:=5;

//числова константа

for i:=48 to 57 do

Attributes[i]:=1;

for i:=65 to 90 do

//ідентифікатор

Attributes[i]:=2;

//коментар

Attributes[40]:=3;

//роздільник

Attributes[44]:=4;

Attributes[46]:=4;

Attributes[58]:=4;

Attributes[59]:=4;

Attributes[61]:=4;

//whitespace

Attributes[9]:=0;

Attributes[10]:=0;

Attributes[13]:=0;

Attributes[32]:=0;

KeyTabl[1].key:='PROGRAM';

KeyTabl[1].number:=701;

KeyTabl[2].key:='LABEL';

KeyTabl[2].number:=702;

KeyTabl[3].key:='BEGIN';

KeyTabl[3].number:=703;

KeyTabl[4].key:='GOTO';

KeyTabl[4].number:=704;

KeyTabl[5].key:='ENDIF';

KeyTabl[5].number:=705;

KeyTabl[6].key:='IF';

KeyTabl[6].number:=706;

KeyTabl[7].key:='THEN';

KeyTabl[7].number:=707;

KeyTabl[8].key:='ELSE';

KeyTabl[8].number:=708;

KeyTabl[9].key:='END';

KeyTabl[9].number:=709;

end;

Procedure TForm1.ConstTabSearch(bf1:string;Te:integer; var num:integer;var C:boolean);

var i:byte;

Begin

for i:=1 to CTE do

begin

if bf1=ConstTabl[i].Key then

begin

num:=ConstTabl[i].number;

C:=true;

break;

end;

C:=false;

end;

end;

Procedure TForm1.IdnTabSearch(bf2:string; Te:integer; var num:integer; var IP:boolean);

var i:byte;

Begin

for i:=1 to IDN do

begin

if bf2=IdnTabl[i].Key then

begin

num:=IdnTabl[i].number;

IP:=true;

break;

end;

IP:=false;

end;

end;

Procedure TForm1.KeyTabSearch(bf:string; var num:integer; var K:boolean);

var i:byte;

Begin

for i:=1 to 9 do

begin

if bf=KeyTabl[i].Key then

begin

num:=KeyTabl[i].number;

K:=true;

break;

end

else

K:=false;

end;

end;

Procedure TForm1.ConstTabForm(bf3:string;num:integer);

Begin

ConstTabl[num].key:=bf3;

ConstTabl[num].number:=num+400;

end;

Procedure TForm1.IdnTabForm(bf4:string;num:integer);

Begin

IdnTabl[num].key:=bf4;

IdnTabl[num].number:=num+709;

end;

//parser procedures

Procedure TForm1.ParserIdnSearch(var bf2:string; k:integer; var B:boolean);

var i:byte;

Begin

B:=false;

for i:=1 to IDN do

begin

if k = IdnTabl[i].number then

begin

bf2:=IdnTabl[i].key;

B:=true;

end;

end;

end;

Procedure TForm1.ParserConstSearch(bf1:string; var num:integer;var C:boolean);

var i:byte;

Begin

for i:=1 to CTE do

begin

if StrToInt(bf1)=ConstTabl[i].number then

begin

num:=i;

C:=true;

break;

end;

C:=false;

end;

end;

Procedure TForm1.idnt(var tree:TTreeView; s2:string; var Node:TTreeNode);

var Node1:TTreeNode;

Node2:TTreeNode;

begin

Node1 := tree.Items.AddChild(Node, '<identifier>');

Node2 := tree.Items.addchild(Node1, s2);

end;

Procedure TForm1.label\_decl(var TNode:TTreeNode; var sind:integer; var tree:TTreeView);

var i, k:integer;

Node, Node1, Node2:TTreeNode;

B:boolean;

begin

ParserConstSearch(IntToStr(lexstr[sind]),k,B);

if B=true then

begin

Node := tree.Items.AddChild(TNode,'<unsigned-integer>');

Node1:= tree.Items.AddChild(Node,ConstTabl[k].key);

inc(sind);

end

else

begin

errflag:=true;

err\_log.Lines.Add('Error, constant expected in line: '+inttostr(lexpos[sind]));

tree.Enabled := false;

end;

case lexstr[sind] of

59: Node2 := tree.Items.AddChild(TNode,';');

44: begin

Node2 := tree.Items.AddChild(TNode,',');

inc(sind);

label\_decl(TNode,sind,tree);

end;

else

begin

errflag:=true;

err\_log.Lines.Add('Error: , or ; expected in line: '+inttostr(lexpos[sind]));

tree.Enabled := false;

end;

end;

end;

Procedure TForm1.declarations(var TNode:TTreeNode; var sind:integer; var tree:TTreeView);

var j,i,k:integer;

Node,Node1,Node2:TTreeNode;

B:boolean;

begin

Node := tree.Items.AddChild(TNode, '<declarations>.8');

Node1:= tree.Items.AddChild(Node,'<label-declarations>');

KeyTabSearch('LABEL',k,B);

if lexstr[sind] <> k then

begin

Node2 := tree.Items.AddChild(Node1,'<empty>');

inc(sind);

end

else

begin

inc(sind);

Node2 := tree.Items.AddChild(Node1,'LABEL');

label\_decl(Node2, sind, tree);

end;

end;

Procedure TForm1.cond(var TNode: TTreeNode; var sind: integer; var tree: TTreeView);

var MNode,Node,Node1,Node2:TTreeNode;

i,k:integer;

s:string;

B:boolean;

begin

node:=tree.Items.AddChild(TNode,'IF');

inc(sind);

ParserIdnSearch(s, lexstr[sind], B);

if B=true then

begin

node1:=tree.Items.AddChild(node,'<variable-identifier>');

node2:=tree.Items.AddChild(node1,s);

inc(sind);

end

else

begin

err\_log.Lines.Add('Error, identifier expected in line:'+IntToStr(lexpos[sind]));

tree.Enabled := false;

errflag:=true;

end;

if lexstr[sind]=61 then

begin

node1:=tree.Items.AddChild(node,'=');

inc(sind);

end

else

begin

errflag:=true;

err\_log.Lines.Add('Error, = expected in line:'+IntToStr(lexpos[sind]));

tree.Enabled := false;

end;

B:=False;

ParserConstSearch(IntToStr(lexstr[sind]),k,B);

if B=true then

begin

node1:=tree.Items.AddChild(node,'<unsigned-integer>');

node2:=tree.Items.AddChild(node1,ConstTabl[k].key);

inc(sind);

end

else

begin

errflag:=true;

err\_log.Lines.Add('Error constant expected in line:'+IntToStr(lexpos[sind]));

tree.Enabled := false;

end;

if lexstr[sind]=707 then

begin

node1:=tree.Items.AddChild(TNode,'THEN');

thenflag:=true;

inc(sind);

statements(node1,sind,tree);

end

else

begin

errflag:=true;

err\_log.Lines.Add('Error THEN expected in line:'+IntToStr(lexpos[sind]));

tree.Enabled := false;

end;

end;

Procedure TForm1.statements(var TNode: TTreeNode; var sind: integer; var tree:TTreeView);

var MNode, Node, Node1, Node2:TTreeNode;

i,k:integer;

B:boolean;

begin

case lexstr[sind] of

401..500: begin

MNode:=tree.Items.AddChild(TNode,'<statement>.5');

node:=tree.Items.AddChild(MNode,'<unsigned-integer>');

parserConstSearch(IntToStr(lexstr[sind]),k,B);

node1:=tree.Items.AddChild(node,ConstTabl[k].key);

inc(sind);

if lexstr[sind]=58 then

begin

node2:=tree.Items.AddChild(MNode,':');

inc(sind);

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

statements(TNode,sind,tree);

end

else

begin

errflag:=true;

err\_log.Lines.Add('Error, : expected in line:'+IntToStr(lexpos[sind]));

tree.Enabled := false;

statements(TNode,sind,tree);

end

end;

704:begin

MNode:=tree.Items.AddChild(TNode,'<statement>.5');

node:=tree.Items.AddChild(MNode,'GOTO');

inc(sind);

parserConstSearch(IntToStr(lexstr[sind]),k,B);

if B=true then

begin

node1:=tree.Items.AddChild(node,ConstTabl[k].key);

inc(sind);

if lexstr[sind]=59 then

begin

node2:=tree.Items.AddChild(MNode,';');

inc(sind);

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

statements(TNode,sind,tree);

end

else

begin

errflag:=true;

err\_log.Lines.Add('Error, ; expected in line:'+IntToStr(lexpos[sind]));

tree.Enabled := false;

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

statements(TNode,sind,tree);

end;

end

else

begin

errflag:=true;

err\_log.Lines.Add('Error GOTO label expected in line'+IntToStr(lexpos[sind]));

tree.Enabled := false;

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

statements(TNode,sind,tree);

end

end;

59:begin

node:=tree.Items.AddChild(TNode,';');

inc(sind);

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

statements(TNode,sind,tree);

end;

706:begin

ifflag:=true;

MNode:=tree.Items.AddChild(TNode,'<statement>.5');

node:=tree.Items.AddChild(MNode,'<conditional-statement>');

node1:=tree.Items.AddChild(node,'<incomplete-condition-statement>');

cond(node1, sind,tree);

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

statements(TNode,sind,tree);

end;

708:begin

if ifflag=true then

begin

MNode:=tree.Items.AddChild(TNode,'<statement>.5');

node:=tree.Items.AddChild(MNode,'<alternative-part>');

node1:=tree.Items.AddChild(node,'ELSE');

inc(sind);

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

elseflag:=true;

statements(node1,sind,tree);

exit;

end

else

begin

errflag:=true;

err\_log.Lines.Add('Error ELSE finded but IF expected');

tree.Enabled := false;

inc(sind);

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

elseflag:=true;

statements(TNode,sind,tree);

exit;

end;

exit;

end;

705:begin

elseflag:=false;

if ifflag=true then

begin//1

node:=tree.Items.AddChild(TNode,'ENDIF.5');

inc(sind);

if lexstr[sind]=59 then

begin

node1:=tree.Items.AddChild(TNode,';');

inc(sind);

thenflag:=false;

elseflag:=false;

exit;

end

else

begin

errflag:=true;

err\_log.Lines.Add('Error, ; expected in line:'+IntToStr(lexpos[sind-1]));

tree.Enabled := false;

inc(sind);

if lexstr[sind]=59 then inc(sind);

end;

end//1

else

begin//2

errflag:=true;

err\_log.Lines.Add('Error, ENDIF founded, but IF expected');

tree.Enabled := false;

inc(sind);

if lexstr[sind]=59 then inc(sind);

end;//2

ifflag:=false;

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

exit;

statements(TNode,sind,tree);

end;

709:begin

lexstr[sind+1]:=10001;

exit;

end;

10001:exit;

else begin

errflag:=true;

tree.Enabled := false;

err\_log.Lines.Add('Error, unknown statement in line:'+IntToStr(lexpos[sind]));

inc(sind);

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

statements(TNode,sind,tree);

end;

end;

//inc(sind);

if TNode.Text='ELSE' then exit;

if thenflag=true or elseflag=true then

begin

thenflag:=false;

elseflag:=false;

exit;

end;

statements(TNode,sind,tree);

end;

Procedure TForm1.block(var TNode:TTreeNode;var sind:integer;var tree:TTreeView);

var Node, Node1, Node2:TTreeNode;

B:boolean;

i,j,k:integer;

begin

Node := tree.Items.Addchild(TNode,'<block>.3');

declarations(Node, sind, tree);

inc(sind);

KeyTabSearch('BEGIN',k,B);

if lexstr[sind] = k

then begin

Node1 := tree.Items.addchild(Node, 'BEGIN');

inc(sind);

end

else

begin

err\_log.Lines.Add('Error, BEGIN expected in line: '+inttostr(lexpos[sind]));

errflag:=true;

tree.Enabled := false;

end;

Node1 := tree.Items.addchild(Node,'<statements-list>');

statements(node1,sind,tree);

if lexstr[sind] = 709 then Node1 := tree.Items.addchild(Node, 'END.6')

else

begin

err\_log.Lines.Add('Error, END expected in line: '+inttostr(lexpos[sind]));

errflag:=true;

tree.Enabled := false;

end;

end;

Procedure TForm1.procedure\_identifier(var node:TTreeNode; var sind:integer; var tree:TTreeView);

var s:string;

j,i,k:integer;

tnode,node1:TTreeNode;

B:boolean;

begin

ParserIdnSearch(s, lexstr[sind], B);

if B = true then

begin

tNode := tree.Items.AddChild(Node, '<procedure-identifier>');

idnt(tree, s, tNode);

inc(sind);

if lexstr[sind] <> 59 then

begin

err\_log.Lines.Add('Error, ; expected after identifier in line: '+inttostr(lexpos[sind]));

errflag:=true;

tree.Enabled := false;

end

else

begin

Node1 := tree.Items.addchild(Node, ';');

inc(sind);

end;

end

else

begin

errflag:=true;

tree.Enabled := false;

err\_log.lines.add('Error, PROCEDURE indentifier expected in line: '+inttostr(lexpos[sind]));

if lexstr[sind] = 59 then

begin

Node1 := tree.Items.addchild(Node, ';');

inc(sind);

end

else

begin

err\_log.Lines.Add('Error, ; expected after identifier in line: '+inttostr(lexpos[sind]));

errflag:=true;

tree.Enabled := false;

end;

end;

end;

Procedure TForm1.signal\_program(var tree:TTreeView; var sind:integer);

var j,k,i:integer;

begin

node := tree.Items.Add(nil, '<program>.2');

node.HasChildren := True;

ifflag:=false;

thenflag:=false;

elseflag:=false;

KeyTabSearch('PROGRAM',k,fl);

if lexstr[sind] = k then begin

tree.Items.AddChildFirst(node, 'PROGRAM');

end

else begin err\_log.lines.add('Error, PROGRAM expected in line: '+inttostr(lexpos[sind]));

tree.Enabled := false; end;

inc(sind);

procedure\_identifier(Node, sind, tree);

//param\_list(TNode, lexrow, tr);

block(node, sind, tree);

end;

Procedure TForm1.signal(var tree:TTreeView);

begin

tree.Items.Add(nil, '<signal-program>.1');

sind:=1;

signal\_program(tree, sind);

end;

//\*parser procedures

//lex analyz

//main procedure

procedure TForm1.main();

Begin

memo1.Clear;

memo2.Clear;

memo3.Clear;

//відкриття файлу початкової програми

AssignFile(FINP,fname);

reset(FINP);

//вивід початкового файлу на екран

readln(FINP, buf1);

while (not EOF(FINP)) do

begin

memo3.Lines.Add(buf1);

readln(finp, buf1);

end;

memo3.Lines.Add(buf1);

reset(FINP);

AssignFile(tablf,'tabl.txt');

AssignFile(FOUT,'Output.txt');

rewrite(FOUT);

rewrite(tablf);

writeln(fout, 'lexem string:');

writeln(fout, ' ');

ind:=1;

CTE:=0;IDN:=0;

CT:=false; KT:=false; IT:=false;

errflag:=false;

//початкове встановлення таблиць ідентифікаторів і констант

FillAttributes;

if eof(FINP) then

begin

memo1.lines.add('Empty file');

Writeln(FOUT,'Empty file');

errflag:=true;

end;

jmpcount:=0;

lin:=1;

symbol:=Gets;

repeat

buf := '';

lexCode := 0;

SuppressOutput := False;

case symbol.attr of

0: (\*whitespace\*)

begin

while not eof(FINP) do

begin

symbol := Gets;

if symbol.attr <> 0 then

Break;

end;

SuppressOutput := True;

end;

1: (\*константа\*)

begin

while not eof(FINP) and (symbol.attr = 1) do

begin

buf := buf + symbol.value;

symbol := Gets;

end;

ConstTabSearch(buf,CTE,char,CT);

if CT=true then

begin

lexCode := char;

lexstr[ind]:=lexCode;

lexpos[ind]:=lin;

inc(ind);

end

else

begin

CTE:=CTE+1;

ConstTabForm(buf,CTE);

lexCode := CTE+400;

lexstr[ind]:=lexCode;

lexpos[ind]:=lin;

inc(ind);

end;

end;

2: (\*ідентифікатор\*)

begin

while not eof(FINP) and ((symbol.attr = 1)or (symbol.attr = 2)) do

begin

buf := buf + symbol.value;

symbol := Gets;

end;

KeyTabSearch(buf,chKT,KT);

if KT=true then

begin

lexCode := chKT;

lexstr[ind]:=lexCode;

lexpos[ind]:=lin;

inc(ind);

end

else begin

IdnTabSearch(buf,IDN,chIDN,IT);

if IT=true then

lexCode :=chIDN

else

begin

IDN:=IDN+1;

IdnTabForm(buf,IDN);

lexCode := IDN+709;

lexstr[ind]:=lexCode;

lexpos[ind]:=lin;

inc(ind);

end;

end;

end;

3: (\*можливий коментар, тобто зустрінута '(' \*)

begin

if eof(FINP) then

begin

memo1.Lines.add('Illegal symbol in line:'+inttostr(lin));

Write(fout,' 10000 '+inttostr(lin));

errflag:=true;

SuppressOutput :=true;

end

else

begin

symbol := Gets;

if symbol.value = '\*' then

begin

if eof(FINP) then

begin

memo1.Lines.add('\*) expected but end of file found');

Writeln(fout,' 10000');

errflag:=true;

SuppressOutput :=true;

end

else

begin

symbol := Gets;

repeat

while not eof(FINP) and (symbol.value <> '\*') do

symbol := Gets;

if eof(FINP) then //якщо кінець файла

begin

memo1.Lines.add('\*) expected! Error!');

Writeln(fout, ' 10000');

errflag:=true;

symbol.value := '+'; // все що завгодно, але не ')'

SuppressOutput :=true;

Break;

end

else //була '\*' і немає кінця файла

symbol := Gets;

until symbol.value = ')';

if symbol.value = ')' then

SuppressOutput := True;

if not eof(FINP) then

symbol := Gets;

end;

end

else

begin

memo1.Lines.add('Illegal symbol in line:'+inttostr(lin));

Write(fout,' 10000 '+inttostr(lin));

errflag:=true;

SuppressOutput :=true;

end;

end;

end;

4: (\*роздільник\*)

begin

lexCode := Ord(symbol.value);

lexstr[ind]:=lexCode;

lexpos[ind]:=lin;

inc(ind);

symbol := Gets;

end;

5: (\*помилка\*)

begin

memo1.lines.add('Illegal symbol in line:'+inttostr(lin));

//inc(ind);

Write(fout,' 10000 '+inttostr(lin));

errflag:=true;

symbol := Gets;

SuppressOutput:= true;

end;

end; (\*case\*)

if not SuppressOutput then

begin memo1.Lines.add('Output: '+IntToStr(lexCode));

write(FOUT,' ',lexCode);

end;

until eof(FINP);

//Вивід таблиць

Writeln(tablf,'');

ind:=0;

memo2.Lines.Strings[ind]:= 'Keywords Table:';

Writeln (tablf,'Keywords Table:');

Writeln(tablf,'');

for j:=1 to 9 do

begin

memo2.Lines.add( inttostr(KeyTabl[j].number)+' '+ KeyTabl[j].key);

Writeln(tablf, KeyTabl[j].number,' ', KeyTabl[j].key);

end;

if CTE<>0 then

begin

Writeln(tablf,'');

memo2.Lines.Add('Constant Table:');

Writeln (tablf,'Constant Table:');

Writeln(tablf,'');

for j:=1 to CTE do

begin

memo2.Lines.Add(inttostr(ConstTabl[j].number)+' '+ ConstTabl[j].key);

Writeln(tablf, ConstTabl[j].number,' ', ConstTabl[j].key);

end;

end;

if IDN<>0 then

begin

Writeln(FOUT,'');

memo2.Lines.Add('Identifier Table:');

Writeln (tablf,'Identifier Table:');

Writeln(tablf,'');

for j:=1 to IDN do

begin

memo2.Lines.Add(inttostr(IdnTabl[j].number)+' '+ IdnTabl[j].key);

Writeln(tablf, IdnTabl[j].number,' ', IdnTabl[j].key);

end;

end;

Writeln(tablf,'');

memo2.Lines.Add('Separators Table:');

Writeln (tablf,'Separators Table:');

Writeln(tablf,'');

memo2.Lines.Add('44 ,');

Writeln(tablf,'44',' ',',' );

memo2.Lines.Add('46 .');

Writeln(tablf,'46',' ','.');

memo2.Lines.Add('58 :');

Writeln(tablf,'58',' ',':');

memo2.Lines.Add('59 ;');

Writeln(tablf,'59',' ',';');

memo2.Lines.Add('61 =');

Writeln(tablf,'61',' ','=');

//\* lex analyz

CloseFile(fout);

CloseFile(finp);

CloseFile(tablf);

//parser

signal(tree);

tree.SaveToFile('TREE.TXT');

end;

procedure TForm1.Button1Click(Sender: TObject);

begin

main();

end;

procedure TForm1.Exit1Click(Sender: TObject);

begin

Close;

end;

end.