Міністерство освіти і науки України

Національній технічний університет України "Київський політехнічний інститут"

Факультет прикладної математики

Кафедра спеціалізованих комп’ютерних систем

Лабораторна робота № 1

Дисципліна:

"Основи проектування трансляторів"

Тема:

"Побудова лексичного аналізатора"

**Варіант №13**

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

**Київ 2011**

**Завдання**

Варіант 13

<signal-program> --> <program>

<program> --> PROCEDURE <procedure-identifier><<parameters-list>> ; <block> ;

<block> --> <declarations> BEGIN <statements-list> END

<declarations> --> <procedure-declarations>

<procedure-declarations> --> <procedure> <procedure-declarations> |  
<empty>

<procedure> --> PROCEDURE <procedure-identifier><parameters-list> ;

<parameters-list> --> ( <variable-identifier> <identifiers-list> ) |  
<empty>

<identifiers-list> --> , <variable-identifier> <identifiers-list> |  
<empty>

<statements-list> --> <statement> <statements-list> |  
<empty>

<statement> --> <procedure-identifier><actual-arguments> ; |  
RETURN ;

<actual-arguments> --> ( <unsigned-integer> <actual-arguments-list> ) |  
<empty>

<actual-arguments-list> --> , <unsigned-integer> <actual-arguments-list> |  
<empty>

<variable-identifier> --> <identifier>

<procedure-identifier> --> <identifier>

<identifier> --> <letter><string>

<string> --> <letter><string> |  
<digit><string> |  
<empty>

<unsigned-integer> --> <digit><digits-string>

<digits-string> --> <digit><digits-string> |  
<empty>

<digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letter> --> A | B | C | D | ... | Z

Код програми

## Файл заголовків

#ifndef ANALYZER\_H

#define ANALYZER\_H

#include <iostream>

#include <fstream>

#include <string>

#include <map>

#include <QString>

using namespace std;

typedef map <int, int> myLineTable;

typedef map <string, int> myTable;//таблиця ідентифікаторів і роздільників

// стани цифрового автомату

#define S 10001

#define CNS 10003

#define IDN 10004

#define BCOM 10005

#define COM 10006

#define ECOM 10007

#define ERR 10008

#define OUT 10009

#define EXIT 10010

#define HSTN 10011

#define IDENT\_LST\_COUNT 800

#define CONST\_LST\_COUNT 400

#define KEYWORD\_LST\_COUNT 700

// можвливі варіанти вхідних символів

enum inputSign { DG, LT, DM, WS, PRNS, OTH };//PRNS - круглі дужки

class LexAn{

//myTable іdentList,keyWordList,sepList,constList;

myTable identList, //800-1000

keyWordList, //700-703

sepList, //0-255

constList; //400-500

myLineTable lineTable;

string inputFile,//в цю змінну дані записуються із файлу з текстом програми

keyWordsListFileName,//Імя файлу ключових слів

inputFileName,//імя вхідного файлу

constListFileName,//імя вихідного файлу констант

identListFileName,//імя вихідного файлу ідентифікаторів

sepListFileName,//імя вихідного файлу роздільників

outputFile,//вихідний файл.

outputFileName,//імя вихідного файлу із лексемами

myFileName,//імя мого файл

currentToken,//поточний токен

myFile,//мій файл

errorList;

int state,//Поточний стан ЦА

lastState,//Попередінй стан ЦА

currentPos,//поточна позиція від початку файлу

lineCounter,

possitionCounter,

errorCounter,//лічильник помилок

lineTableCounter;

bool exit,

DGFlag,

LTFlag,

DMFlag,

WSFlag,

PRNSFlag,

OTHFlag;

char determinedArray[256];//масив для визначення типу поточного вхідного символа

inputSign inpSignType;//тип поточного вхідного символу

char currentSign;//поточний вхідний символ

//\*\*\*\*\*\*\*\*\*МЕТОДИ\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void show();

void run();

inputSign determineInputSign();

void loadKeyWordList();

void readNextSign();

void proccesState();

void proccesStateS();

void proccesStateCNS();

void proccesStateIDN();

void proccesStateBCOM();

void proccesStateCOM();

void proccesStateECOM();

void proccesStateERR();

void proccesStateOUT();

void proccesStateEXIT();

void proccesStateHSTN();

void proccesStateMSDM();

public:

LexAn();

~LexAn();

void shareData(string &tokenString,//вихідний рядок лексем

myTable &identifiers,//таблиця ідентифікаторів

myTable &constants,//таблиця констант

myTable &separators,//таблиця роздільників

string &myErrorList,

myTable &myKeyWordList,

bool &error);

void setData(QString inFile, QString keyWordListFile);

void getLineTable(myLineTable &myTable);

};

#endif // ANALYZER\_H

## Файл визначень

#include "LexicalAnalyzer.h"

#include <stdlib.h>

LexAn::LexAn()

{

currentSign = 0;

currentPos = 0;

lineCounter = 1;

errorCounter = 0;

possitionCounter = 0;

exit = false;

DGFlag = false;

LTFlag = false;

DMFlag = false;

WSFlag = false;

PRNSFlag = false;

OTHFlag = false;

lineTableCounter = 0;

for (int i = 0; i < 256; ++i)

{

int tmp = i;

if ((tmp > 64) && (tmp < 91)){

determinedArray[i] = LT;

} else

if ((tmp > 47) && (tmp < 58)){

determinedArray[i] = DG;

} else

if ((tmp == '(') || (tmp == ')') || (tmp == '\*') || (tmp == ',')|| (tmp == ';')){

determinedArray[i] = DM;

} else

if((tmp == 32) || (tmp == 9) || (tmp == 10)){

determinedArray[i] = WS;

} else

if (tmp == 40){

determinedArray[i] = PRNS;

} else

determinedArray[i] = OTH;

}

}

LexAn::~LexAn()

{

inputFileName.erase();

inputFile.erase();

}

void LexAn::loadKeyWordList()

{

string ident, id;

ifstream inFile(keyWordsListFileName.c\_str());

if (!inFile.is\_open())

{

errorCounter = 1;

return;

}

while (!inFile.eof())

{

inFile >> ident;

inFile >> id;

keyWordList[ident] = atoi(id.c\_str());

}

}

void LexAn::readNextSign()

{

currentSign = inputFile[currentPos++];

if (currentSign == '\0'){

exit = true;

lastState = state;

if (currentToken.size() > 0){

proccesStateOUT();

}

}

}

inputSign LexAn::determineInputSign()

{

return (inputSign)determinedArray[(int)currentSign];

}

void LexAn::run()

{

state = S;

lastState = S;

while (!exit)

{

switch ( state ) {

case S:

proccesStateS();

break;

case CNS:

proccesStateCNS();

break;

case IDN:

proccesStateIDN();

break;

case BCOM:

proccesStateBCOM();

break;

case COM:

proccesStateCOM();

break;

case ECOM:

proccesStateECOM();

break;

case ERR:

proccesStateERR();

break;

case OUT:

proccesStateOUT();

break;

case EXIT:

proccesStateEXIT();

break;

case HSTN:

proccesStateHSTN();

break;

}

}

}

void LexAn::proccesStateS()

{

readNextSign();

inpSignType = determineInputSign();

switch (inpSignType)

{

//1.

case DG:

DGFlag = true;

state = ERR;

break;

//2.

case LT:

state = IDN;

currentToken += currentSign;

break;

//3.

case DM:

DMFlag = true;

state = ERR;

break;

//4.

case WS:

//nothing

break;

//5.

case PRNS:

state = BCOM;

break;

//6.

case OTH:

OTHFlag = true;

state = ERR;

break;

}

}

void LexAn::proccesStateCNS()

{

readNextSign();

inpSignType = determineInputSign();

switch (inpSignType)

{

//1.

case DG:

currentToken += currentSign;

break;

//2.

case LT:

lastState = state;

LTFlag = true;

state = ERR;

break;

//3.

case DM:

lastState = state;

--currentPos;

state = OUT;

break;

//4.

case WS:

lastState = state;

--currentPos;

state = OUT;

break;

//5.

case PRNS:

lastState = state;

--currentPos;

state = OUT;

break;

//6.

case OTH:

lastState = state;

OTHFlag = true;

state = ERR;

break;

}

}

void LexAn::proccesStateIDN()

{

readNextSign();

inpSignType = determineInputSign();

switch (inpSignType)

{

//1.

case DG:

currentToken += currentSign;

break;

//2.

case LT:

currentToken += currentSign;

break;

//3.

case DM:

lastState = state;

--currentPos;

state = OUT;

break;

//4.

case WS:

lastState = state;

--currentPos;

state = OUT;

break;

//5.

case PRNS:

lastState = state;

--currentPos;

state = OUT;

break;

//6.

case OTH:

lastState = state;

OTHFlag = true;

state = ERR;

break;

}

}

void LexAn::proccesStateBCOM()

{

char prevSign = currentSign;

readNextSign();

inpSignType = determineInputSign();

switch (inpSignType)

{

//1.

case DG:

currentToken += prevSign;

currentPos--;

lastState = state;

state = OUT;

break;

//2.

case LT:

currentToken += prevSign;

currentSign = prevSign;

currentPos--;

lastState = state;

state = OUT;

break;

//3.

case DM:

if (currentSign == '\*')

{

state = COM;

} else

if ((currentSign == '+') || (currentSign == '-'))

{

lastState = state;

currentToken = prevSign;

currentSign = prevSign;

--currentPos;

state = OUT;

} else

{

lastState = state;

DMFlag = true;

state = ERR;

}

break;

//4.

case WS:

currentToken += prevSign;

currentPos--;

lastState = state;

state = OUT;

break;

//5.

case PRNS:

currentToken += prevSign;

currentPos--;

lastState = state;

state = OUT;

break;

//6.

case OTH:

lastState = state;

OTHFlag = true;

state = ERR;

break;

}

}

void LexAn::proccesStateCOM()

{

readNextSign();

if (currentSign == 10)

lineTable[lineTableCounter++] = outputFile.size();

if (currentSign == '\*')

state = ECOM;

}

void LexAn::proccesStateECOM()

{

readNextSign();

if (currentSign == ')')

state = HSTN;

else if (currentSign == '\*')

state = ECOM;

else state = COM;

}

void LexAn::show()

{

int counter = 0;

char ch[5];

errorList += "Line - ";

errorList += \_itoa(lineCounter, ch, 10);

errorList += ", position - ";

errorList += \_itoa(currentPos - possitionCounter, ch, 10);

errorList += (char)10;

errorList += '\"';

counter = possitionCounter;

while (true)

{

if ((inputFile[counter] == 10) || (inputFile[counter] == 0))

{

errorList += '\"';

errorList += (char)10;

errorList += (char)10;

break;

}

errorList += inputFile[counter++];

}

}

void LexAn::proccesStateERR()

{

errorCounter++;

if (lastState == S)

{

if (DGFlag)

{

errorList += "Error No 2 - first token cann't be a constant! ";

DGFlag = false;

}

if (DMFlag)

{

errorList += "Error No 5 - firs token cann't be a delimiter!";

DMFlag = false;

}

if (OTHFlag)

{

errorList += "Error No 1 - extra-character on line!";

OTHFlag = false;

}

show();

} else

if (lastState == CNS)

{

if (LTFlag)

{

errorList += "Error No 3 - constants cann't contain a letters!";

LTFlag = false;

}

if (OTHFlag)

{

errorList += "Error No 1 - extra-character on line!";

OTHFlag = false;

}

show();

} else

if (lastState == IDN)

{

if (OTHFlag)

{

errorList += "Error No 1 - extra-character on line!";

OTHFlag = false;

}

show();

} else

if (lastState == BCOM)

{

if (DMFlag)

{

errorList += "Error No 4 - unexpected character on line!";

DMFlag = false;

}

if (OTHFlag)

{

errorList += "Error No 1 - extra-character on line!";

OTHFlag = false;

}

show();

} else

if (lastState == HSTN)

{

if (OTHFlag)

{

errorList += "Error No 1 - extra-character on line!";

OTHFlag = false;

}

show();

} else

currentToken.erase();

lastState = state;

state = HSTN;

}

void LexAn::proccesStateOUT()

{

char ch[5];

myFile += currentToken;

myFile += ' ';

if (lastState == CNS)

{

if (constList[currentToken])

{

\_itoa(constList[currentToken], ch, 10);

outputFile += ch;

outputFile += " ";

currentToken.erase();

}else

{

constList[currentToken] = CONST\_LST\_COUNT + constList.size();

\_itoa(constList[currentToken], ch, 10);

outputFile += ch;

outputFile += " ";

currentToken.erase();

}

}else

if (lastState == IDN)

{

bool presents = false;

for (myTable::iterator it = keyWordList.begin(); it != keyWordList.end(); ++it)

{

if (it->first == currentToken)

{

presents = true;

break;

}

}

if (presents)//Якщо знайдено

{

\_itoa(keyWordList[currentToken], ch, 10);

outputFile += ch;

outputFile += " ";

currentToken.erase();

}

else{

if (identList[currentToken])

{

\_itoa(identList[currentToken], ch, 10);

outputFile += ch;

outputFile += " ";

currentToken.erase();

}else

{

identList[currentToken] = IDENT\_LST\_COUNT + identList.size();

\_itoa(identList[currentToken], ch, 10);

outputFile += ch;

outputFile += " ";

currentToken.erase();

}

}

} else

if ((lastState == HSTN)|| (lastState == BCOM))

{

if (sepList[currentToken])

{

\_itoa(sepList[currentToken], ch, 10);

outputFile += ch;

outputFile += " ";

currentToken.erase();

} else

{

sepList[currentToken] = (int)currentSign;

\_itoa(sepList[currentToken], ch, 10);

outputFile += ch;

outputFile += " ";

currentToken.erase();

}

}else

lastState = state;

state = HSTN;

if (exit)

lineTable[lineTableCounter++] = outputFile.size();

}

void LexAn::proccesStateEXIT()

{

//writeFile();

exit = true;

}

void LexAn::proccesStateHSTN()

{

readNextSign();

if (currentSign == 0)

{

lineTable[lineTableCounter++] = outputFile.size();

proccesStateEXIT();

return;

}

if (currentSign == 10)

{

lineTable[lineTableCounter++] = outputFile.size();

++lineCounter;//кількість рядків у програмі.

possitionCounter = currentPos;//запам'ятовується позиція кінця рядка.

}

inpSignType = determineInputSign();

switch (inpSignType)

{

//1.

case DG:

currentToken += currentSign;

lastState = state;

state = CNS;

break;

//2.

case LT:

lastState = state;

state = IDN;

currentToken += currentSign;

break;

//3.

case DM:

currentToken = currentSign;

lastState = state;

state = OUT;

break;

//4.

case WS:

//nothing

break;

//5.

case PRNS:

state = BCOM;

break;

//6.

case OTH:

OTHFlag = true;

lastState = state;

state = ERR;

break;

}

}

void LexAn::shareData(std::string &tokenString,

myTable &identifiers,

myTable &constants,

myTable &separators,

string &myErrorList,

myTable &myKeyWordList,

bool &error)

{

run();

if (errorCounter == 0)

{

tokenString = outputFile;

identifiers.operator = (identList);

constants.operator = (constList);

separators.operator = (sepList);

myKeyWordList.operator = (keyWordList);

error = false;

}else{

error = true;

myErrorList = errorList;

}

}

void LexAn::setData(QString inFile, QString keyWordListFile)

{

for (int i = 0; i < inFile.size(); i++)

{

inputFile+=inFile[i].toAscii();

}

string id, ident;

for (unsigned int i = 0; i <= keyWordListFile.size(); i++)

{

if (determinedArray[(int)keyWordListFile[i].toAscii()] == LT){

ident += keyWordListFile[i].toAscii();

}else

if (determinedArray[(int)keyWordListFile[i].toAscii()] == DG){

id += keyWordListFile[i].toAscii();

} else

if (((int)keyWordListFile[i].toAscii() == 13)||((int)keyWordListFile[i].toAscii() == 10)||((int)keyWordListFile[i].toAscii() == 0))

{

keyWordList[ident] = atoi(id.c\_str());

ident.erase();

id.erase();

}

}

}

void LexAn::getLineTable(myLineTable &myTable)

{

myTable = lineTable;

}

## Файл основного блоку

#include <QtGui/QApplication>

#include "optgui.h"

typedef vector <int> myVector;

int main(int argc, char \*argv[])

{

QApplication app(argc, argv);

OptGUI w;

w.show();

return app.exec();

}

Приклад роботи

PROCEDURE MAINPROC( A, B );

PROCEDURE PROC1( C, D );

BEGIN

PROC1( 1, 2 );

RETURN;

END;

**Constant table: Identifiers table**:

1 401 A 802

2 402 B 803

C 805

D 806

MAINPROC 801

PROC1 804

**Separators table:**

( 40

) 41

, 44

; 59

**Згенерована числова послідовність лексем:**

701 801 40 802 44 803 41 59 701 804 40 805 44 806 41 59 702 804 40 401 44 402 41 59 704 59 703 59