**ЛАБОРАТОРНА РОБОТА #1**

З дисципліни «Інженерія програмного забезпечення»

На тему: «Розробка лексичного аналізатора»

Варіант #8

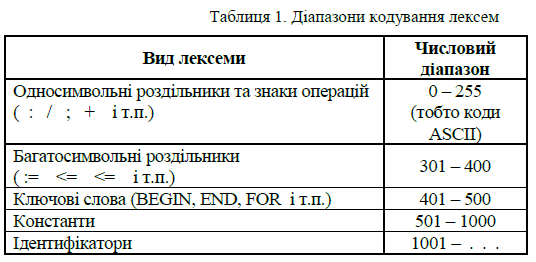
|  |  |
| --- | --- |
| **Виконав**  Студент групи КВ-21  Комарницький О.Б.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Перевірила**  Доцент кафедри СП і СКС  Бояринова Ю. Є.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Варіант #8

**Мета лабораторної роботи**

Метою лабораторної роботи “Розробка лексичного аналізатора” є засвоєння теоретичного матеріалу та набуття практичного досвіду і практичних навичок розробки лексичних аналізаторів (сканерів)

**Постановка задачі**

1. Розробити програму лексичного аналізатора для підмножини мови програмування SIGNAL.
2. Лексичний аналізатор має забезпечувати наступні дії:
   * + - Видалення (пропускання) пробільних символів: пробіл (код ASCII 32), повернення каретки (код ASCII 13); перехід на новий рядок (код ASCII 10), горизонтальна та вертикальна табуляція (коди ASCII 9 та 11), перехід на нову сторінку (код ASCII 12);
       - Згортання ключових слів;
       - Згортання багато-символьних роздільників (якщо передбачаються граматикою варіанту);
       - згортання констант із занесенням до таблиці значення та типу константи (якщо передбачаються граматикою варіанту);
       - Згортання ідентифікаторів;
       - Видалення коментарів, заданих у вигляді (\*<Текст коментаря>\*)
3. Для програмування може бути використана довільна алгоритмічна мова програмування високого рівня. Якщо обрана мова програмування має конструкції або бібліотеки для роботи з регулярними виразами, то використання цих конструкцій та/або бібліотек строго заборонено.
4. Для кодування лексем при їх згортанні необхідно використовувати числова діапазони, вказані в Таблиці 1.
5. Входом ЛА має бути наступне:
   * + - Вхідна програма, написана підмножиною мови SIGNAL відповідно до варіанту;
       - Таблиця кодів ASCII з атрибутами для визначення токенів;
       - Таблиця ключових слів;
       - Таблиця багато-символьних роздільників;
       - Таблиця констант, в яку попередньо можуть бути занесені стандартні константи, якщо потрібно;
       - Таблиця ідентифікаторів, в яку попередньо занесені наперед визначені ідентифікатори, якщо потрібно
6. Виходом ЛА має бути наступне:
   * + - Закодований рядок лексем;
       - Таблиця констант, що сформована для конкретної програми і яка містить значення та тип констант;
       - Таблиця ідентифікаторів, що сформована для конкретної програми.

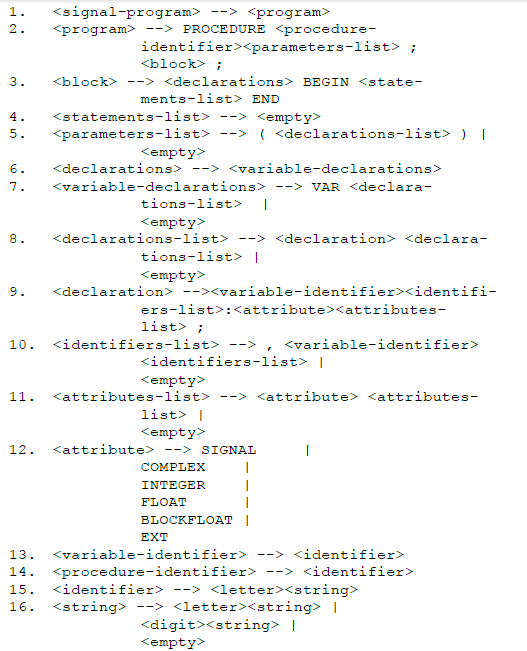
**Зміст звіту програми**

Зміст оформлюється згідно вимог до лабораторних робіт і має мати наступне:

* + - Титульний аркуш;
    - Індивідуальне завдання згідно до варіанту;
    - Таблиці та граф автомату, що задає алгоритм ЛА;
    - Лістинг програми ЛА мовою програмування;
    - Контрольні приклади, необхідні для демонстрації всіх конструкцій заданої граматики, а також всіх можливих помилкових ситуацій (опис кожного контрольного прикладу, який може містити згенерований рядок лексем та заповнені таблиці).

До звіту на довільному носії інформації мають бути додані:

* + - Файл з текстом звіту;
    - Проект працюючої програми мовою програмування;
    - Завантажувальний модуль програми з необхідними файлами даних

**Граматика мови**

Для побудови ЛА із заданої граматики виділимо лексеми (токени), для яких побудуємо регулярні визначення:

**PROCEDURE** 🡪 PROCEDURE

**BEGIN** 🡪 BEGIN

**END** 🡪END

**VAR** 🡪 VAR

**SIGNAL** 🡪 SIGNAL

**COMPLEX** 🡪 COMPLEX

**INTEGER** 🡪 INTEGER

**FLOAT** 🡪 FLOAT

**BLOCKFLOAT** 🡪 BLOCKFLOAT

**EXT** 🡪 EXT

**Identifier** 🡪 letter ( letter | digit ) \*

**Delimiter** 🡪 ; | ( | ) | , | :

**Whitespace 🡪**  space | tab | newline

**Граф автомату, що задає алгоритм ЛА**

**OUT**

**IDN**

other

lit

**S**

lit, dig

**INP**

**OUT**

dm

ws

other

**WS**

ws

**EXIT**

**INP**

**ERR**

other

**INP**

**ERR**

**ERR**

**INP**

)

**ECOM**

\*

**COM**

\*

**FC**

(

eof

eof

other

eof

eof

other

other

eof

**Значення міток:**

lit – літера (A .. Z) ;

dig – цифра (0 .. 9) ;

dm – роздільник (;, (, ), ,, : та ін.) ;

ws – пробільний символ;

eof – символ кінця файлу ;

other – будь-який символ, не вказаний іншими дугами, що виходять із стану S

**Значення станів деталізованих діаграм переходів:**

S – стартовий стан діаграми переходів ЛА ;

EXIT – кінцевий стан ЛА ;

OUT – виведення лексеми (токена) ;

IDN – введення ідентифікатора ;

FC – введення функції чи коментаря;

COM – текст коментаря;

ECOM – закінчення коментаря;

WS – пропуск пробільних символів;

INP – введення чергового символу;

ERR - видача повідомлення про помилку;

**Лістинг програми ЛА мовою програмування С++**

Вміст файлу “Declarations.h”

#include <iostream>

#include <string>

#include <stdio.h>

#include <fstream>

#include <list>

using namespace std;

class Parser

{

public:

Parser();

~Parser();

list<int> processFile();

bool setFile(string);

unsigned short int getLexemType(char );

int searchIdentifier(string );

int searchKeys(string );

short int getLastIdentifierNumber();

void addIdentifier(string, int);

void showIdentifers();

private:

struct \_word\_

{

string Word;

unsigned short int Code;

};

list<\_word\_> IDENTIFIERS;

list<\_word\_> KEYS;

ifstream mFile;

};

Вміст файлу “Scanner.cpp”

#include "Declarations.h"

Parser::Parser()

{

KEYS.push\_back({ "BEGIN", 401 });

KEYS.push\_back({ "END", 402 });

KEYS.push\_back({ "VAR", 403 });

KEYS.push\_back({ "PROCEDURE", 404 });

KEYS.push\_back({ "SIGNAL", 405 });

KEYS.push\_back({ "COMPLEX", 406 });

KEYS.push\_back({ "INTEGER", 407 });

KEYS.push\_back({ "FLOAT", 408 });

KEYS.push\_back({ "BLOCKFLOAT", 409 });

KEYS.push\_back({ "EXT", 410 });

}

Parser::~Parser()

{

}

unsigned short int Parser::getLexemType(char symb)

{

if ((int)symb == 32 || (int)symb == 10 || (int)symb == 9)

return 0;

if ((int)symb >= 48 && (int)symb <= 57)

return 1;

if ((int)symb >= 65 && (int)symb <= 90 || (int)symb >= 97 && (int)symb <= 122 || symb == '\_')

return 2;

if (symb == '(')

return 3;

if (symb == ')' || symb == ',' || symb == ';' || symb == ':')

return 4;

return 5;

}

int Parser::searchKeys(string word)

{

list<\_word\_>::iterator it;

int index = 0;

for (it = KEYS.begin(); it != KEYS.end(); ++it)

{

if ((\*it).Word == word) return (\*it).Code;

index++;

}

return -1;

}

int Parser::searchIdentifier(string word)

{

list<\_word\_>::iterator it;

int index = 0;

for (it = IDENTIFIERS.begin(); it != IDENTIFIERS.end(); ++it)

{

if ((\*it).Word == word) return (\*it).Code;

index++;

}

return -1;

}

short int Parser::getLastIdentifierNumber()

{

list<\_word\_>::iterator it = IDENTIFIERS.end();

if (IDENTIFIERS.size() == 0) return 1000;

return (\*(--it)).Code;

}

void Parser::addIdentifier(string word, int code)

{

IDENTIFIERS.push\_back({ word, code });

}

bool Parser::setFile(string filename)

{

mFile.open(filename);

if (!mFile.is\_open())

return false;

return true;

}

list<int> Parser::processFile()

{

list<int> codedLine;

char symbol;

string buf;

bool suppressOutput;

unsigned short int lexcode;

mFile.get(symbol);

while (!mFile.eof())

{

buf = "";

suppressOutput = false;

switch (getLexemType(symbol))

{

case 0: /\* whitespace \*/

while (!mFile.eof())

{

mFile.get(symbol);

if (getLexemType(symbol) != 0)

break;

}

suppressOutput = true;

break;

case 2: /\* identifier \*/

while (!mFile.eof() && getLexemType(symbol) == 2 || getLexemType(symbol) == 1)

{

buf += symbol;

mFile.get(symbol);

}

if (searchKeys(buf) > 0)

lexcode = searchKeys(buf);

else

{

if (searchIdentifier(buf) > 0)

lexcode = searchIdentifier(buf);

else

{

lexcode = getLastIdentifierNumber() + 1;

addIdentifier(buf, lexcode);

}

}

break;

case 3: /\* Maybe comment, that's '(' \*/

if (mFile.eof())

lexcode = 40; // code of '('

else

{

mFile.get(symbol);

if (symbol == '\*')

{

if (mFile.eof())

cout << "Error: \* expected but end of file found" << endl;

else

{

mFile.get(symbol);

do

{

while (!mFile.eof() && symbol != '\*')

mFile.get(symbol);

if (mFile.eof())

{

cout << "Error: \* expected but end of file found \n";

symbol = '+';

break;

}

else

mFile.get(symbol);

} while (symbol != ')');

if (symbol == ')')

suppressOutput = true;

if (!mFile.eof())

mFile.get(symbol);

}

}

else if (mFile.eof())

cout << "Error: ) expected but end of file found" << endl;

else

lexcode = 40;

}

break;

case 4: /\* Delimiter except '(' \*/

lexcode = (int)symbol;

mFile.get(symbol);

break;

case 5:

case 1:

cout << "Error: Illegal symbol" << endl;

while (getLexemType(symbol) != 0 && !mFile.eof())

mFile.get(symbol);

break;

default:

break;

}

if (!suppressOutput)

codedLine.push\_back(lexcode);

}

mFile.close();

return codedLine;

}

void Parser::showIdentifers()

{

list<\_word\_>::iterator it;

for (it = IDENTIFIERS.begin(); it != IDENTIFIERS.end(); ++it)

cout <<(\*it).Code << ": " << (\*it).Word << endl;

}

Вміст файлу “Main.cpp”

#include "Declarations.h"

int main()

{

Parser parser = Parser();

if (!parser.setFile("Program.txt"))

{

cout << "File can't be opened" << endl;

return 0;

}

list<int> codedLine = parser.processFile();

cout << "All identifiers:" << endl;

parser.showIdentifers();

cout << "Encoded line:" << endl;

for each (int symbol in codedLine)

cout << symbol << " ";

cout << endl;

return 0;

}

**Контрольні приклади**

* Протестуємо випадок коректної реалізації конструкцій

Вміст файлу “Program.txt”

PROCEDURE Proc1(\* \*)();

BEGIN

VAR var5, var6: BLOCKFLOAT, EXT;

END;

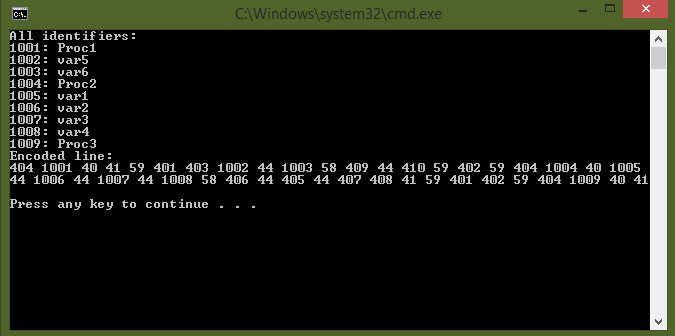
PROCEDURE Proc2(var1, var2, var3, var4: COMPLEX, SIGNAL, INTEGER FLOAT);

BEGIN

END;

PROCEDURE Proc3()

Результат роботи програми: помилок не виявлено.



На скриншоті поданий список ідентифікаторів користувача та закодований рядок лексем.

* Протестуємо випадок некоректної реалізації конструкцій – не закритий коментар

Вміст файлу “Program.txt”

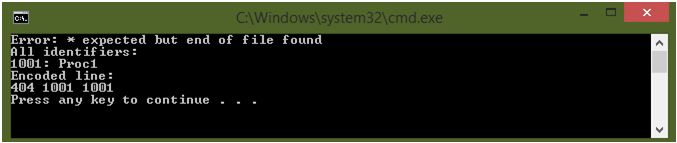
PROCEDURE Proc1(\* \*();

BEGIN

VAR var5, var6: BLOCKFLOAT, EXT;

END;

Результат роботи програми:



* Протестуємо випадок некоректної реалізації конструкцій – наявність невірних односимвольних роздільників

Вміст файлу “Program.txt”

PROCEDURE Proc1(VAR a = INTEGER);

Результат роботи програми

