НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування і спеціалізованих комп'ютерних систем

Розрахунково-графічна робота

з дисципліни

«Основи проектування трансляторів»

Тема: «Розробка лексичного аналізатора»

Виконав: студент III курсу
ФПМ групи КВ-01
Шкільнюк В. О.
Перевірив(ла):

Київ 2023

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.

Метод синтаксичного розбору: метод рекурсивного спуску:

Задана граматика відповідає умовам даного методу (не містить лівобічної рекурсії і відсутні правила з однаковою лівою частиною), тому граматика залишається без змін.

Завдання за варіантом 18:

Граматика підмножини мови програмування SIGNAL:

Варіант 18

```
1.    <signal-program> --> <program>
2.    <program> --> PROGRAM <procedure-identifier> ;
                    <block>.
3.    <block> --> BEGIN <statements-list> END
4.    <statements-list> --> <statement> <statements-list>
                    |
                    <empty>
5.    <statement> --> LOOP <statements-list> ENDLOOP ; |
                    CASE <expression> OF <alternatives-list>
                    ENDCASE ;
6.    <alternatives-list> --> <alternative> <alternatives-
                    list> |
                    <empty>
7.    <alternative> --> <expression> : / <statements-list>
                    \
8.    <expression> --> <multiplier><multipliers-list>
9.    <multipliers-list> --> <multiplication-instruction>
                    <multiplier><multipliers-list>  |
                    <empty>
10.   <multiplication-instruction> -->     *    |
                    /    |
                    MOD
11.   <multiplier> --> <variable-identifier> |
                    <unsigned-integer>
12.   <variable-identifier> --> <identifier>
13.   <procedure-identifier> --> <identifier>
14.   <identifier> --> <letter><string>
15.   <string> --> <letter><string> |
                    <digit><string> |
                    <empty>
16.   <unsigned-integer> --> <digit><digits-string>
17.   <digits-string> --> <digit><digits-string> |
                    <empty>
18.   <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
19.   <letter> --> A | B | C | D | ... | Z
```

Parser.h

```
#ifndef __PARSER_H__
#define __PARSER_H__
//program
#include <string>
#include <sstream>
#include <iostream>
#include <queue>
#include <memory>
#include "analyser.h"

struct tokenInfo;
class Parser {
public:
    Parser() = default;
    ~Parser() = default;
    void program();
    void statementsList();
    void statement();
    void procedureIdentifier();
    void blockIdentifier();
    void expression();
    void multiplier();
    void multipliersList();
    void multiplicationInstruction();
    void altlist();
    void alternative();
    void parse();
public:
    int getToken();
    void setTokens(std::string tokens_);
    void setTokens(std::queue<int> tokens_);
    void setKTokens(std::map<std::string, keywordTokenValues> kTokens_) {
        kTokens = kTokens_;
    }
    void setIdnTokens(std::map<std::string, int> idnTokens_) {
        idnTokens = idnTokens_;
    }
    void setConstTokens(std::map<std::string, int> constTokens_) {
        constTokens = constTokens_;
    }
private:
    std::queue<int> tokens;
    std::istringstream tokenString;
    std::map<std::string, keywordTokenValues> kTokens;
    std::map<std::string, int> idnTokens;
    std::map<std::string, int> constTokens;
    int ts;
};
#endif
```

Parser.cpp

```
#include "parser.h"

void Parser::setTokens(std::queue<int> tokens_) {
    tokens = tokens_;
    ts = -1;
}
int Parser::getToken() {
    if(tokens.empty()) {
        return -1;
    }
    int token = tokens.front();
    tokens.pop();
    return token;
}


void Parser::parse() {
    ts = getToken();
    program();
}

void Parser::program() {
    if(ts == keywordTokenValues::PROGRAM) {
        ts = getToken();
        procedureIdentifier();
    }
    else {
        std::cerr << "The program should start with program word¥n";
        exit(-1);
    }
    ts = getToken();
    if(ts == ';') {
        ts = getToken();
        blockIdentifier();
    }
    else {
        std::cerr << "Block identifier is expected¥n";
        exit(-1);
    }
    if(ts != '.') {
        std::cerr << "Dot is expected as the end of the program";
        exit(-1);
    }
}

void Parser::blockIdentifier() {
    std::cout << "block identifier" << std::endl;
    if(ts == keywordTokenValues::BEGIN) {
        ts = getToken();
        statementsList();
    }
```

```
    else {
        std::cerr << "Wrong block identifier¥n";
        exit(-1);
    }
    if(ts != keywordTokenValues::END) {
        std::cerr << "Wrong end identifier¥n";
    }
    ts = getToken();
}

void Parser::procedureIdentifier() {
    std::cout << "Procedure identirier" << std::endl;
    bool found = false;
    for(auto& pair : idnTokens) {
        if(pair.second == ts) {
            found = true;
        }
    }
    if(!found) {
        std::cerr << "wrong procedure identifier¥n";
        exit(-1);
    }
}

void Parser::statementsList() {
    std::cout << "statements-list" << std::endl;
    statement();
    while(ts == ';') {
        ts = getToken();
        statement();
    }
}

void Parser::statement() {
    if(ts == keywordTokenValues::LOOP) {
        std::cout << "statement" << std::endl;
        ts = getToken();
        statementsList();
        if(ts != keywordTokenValues::ENDLOOP) {
            std::cerr << "Wrong endloop identified¥n";
        }
        ts = getToken();
        if(ts != ';') {
            std::cerr << "Wrong final delimiter¥n";
        }
    }
    else if(ts == keywordTokenValues::CASE) {
        std::cout << "statement" << std::endl;
        ts = getToken();
        expression();
```

```
        if(ts != keywordTokenValues::OF) {
            std::cerr << "Wrong of identifier after OF" << ts << "¥n";

            exit(-1);
        }
        ts = getToken();
        altlist();
        if(ts != keywordTokenValues::ENDCASE) {
            std::cerr << "Wrong endcase identifier" << ts << "¥n";
            exit(-1);
        }
        ts = getToken();
        if(ts != ';') {
            std::cerr << "Wrong endline delimiter¥n";
            exit(-1);
        }
    }
}

void Parser::expression() {

    std::cout << "Expression" << std::endl;
    multiplier();
    multipliersList();
}

void Parser::multiplier() {
    std::cout << "Mutiplier" << std::endl;
    bool found = false;
    // int token = getToken();
    for(auto& pair : idnTokens) {
        if(pair.second == ts) {
            found = true;
        }
    }

    for(auto& pair : constTokens) {
        if(pair.second == ts) {
            found = true;
        }
    }
    if(!found) {
        std::cerr << "wrong multiplier" << ts << "¥n";
        exit(-1);
    }
    ts = getToken();
}

void Parser::multipliersList() {
    std::cout << "multipliers-list" << std::endl;
    // if(ts == keywordTokenValues::MOD || ts == '*') {
```

```
    //     ts = getToken();
    //     multiplier();
    // }
    multiplicationInstruction();
}

void Parser::altlist() {
    std::cout << "alt-list" << std::endl;
    bool found = false;
    for(auto& pair : idnTokens) {
        if(pair.second == ts) {
            found = true;
        }
    }
    for(auto& pair : constTokens) {
        if(pair.second == ts) {
            found = true;
        }
    }
    if(found) {
        alternative();
        altlist();
    }

}

void Parser::alternative() {
    std::cout << "alternative" << std::endl;
    expression();
    if(ts == ':') {
        ts = getToken();
        statementsList();
    }
}
            // VARIABLE3 MOD 51: CASE VARIABLE * 2 OF
            // VARIABLE5: LOOP ENDLOOP;

void Parser::multiplicationInstruction() {
    while(ts == keywordTokenValues::MOD || ts == '*') {
       ts = getToken();
       multiplier();
    }

}

void Parser::setTokens(std::string tokens_) {
    tokenString = std::istringstream(tokens_);
}
```

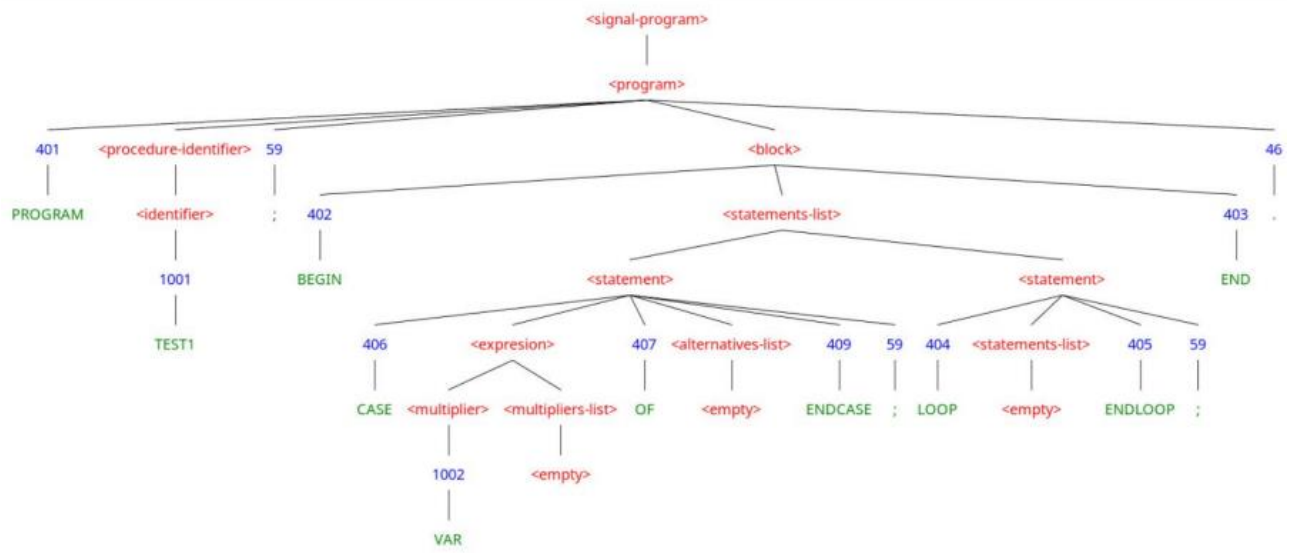
Тест 1

program.txt

```
1  PROGRAM TEST1;
2  BEGIN
3      CASE VAR  OF
4      ENDCASE;
5      LOOP
6      ENDLOOP;
7  END.
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\opt_lab1\build> .\lab1.exe
Procedure identirier
block identifier
statements-list
statement
Expression
Mutiplier
multipliers-list
alt-list
statement
statements-list
PS E:\opt_lab1\build>
```



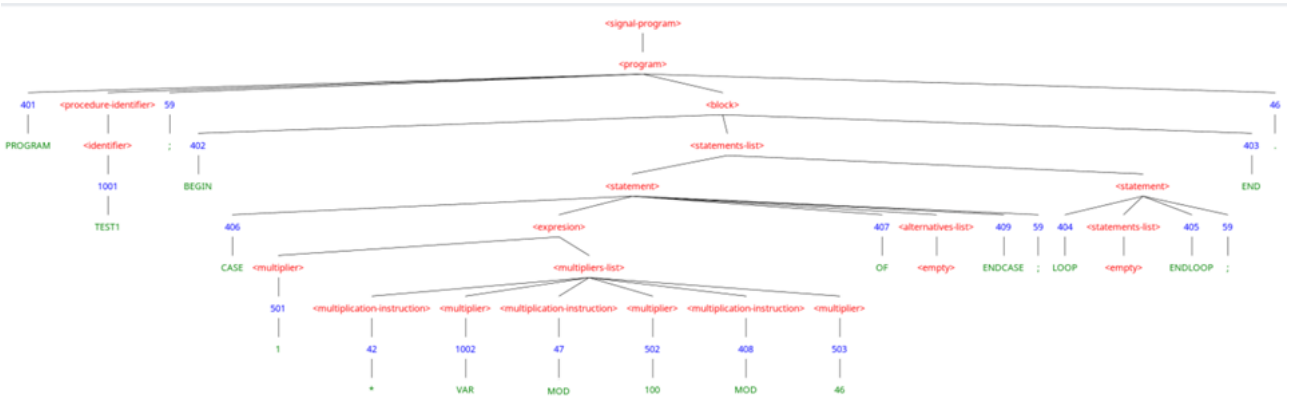
Тест 2

program.txt

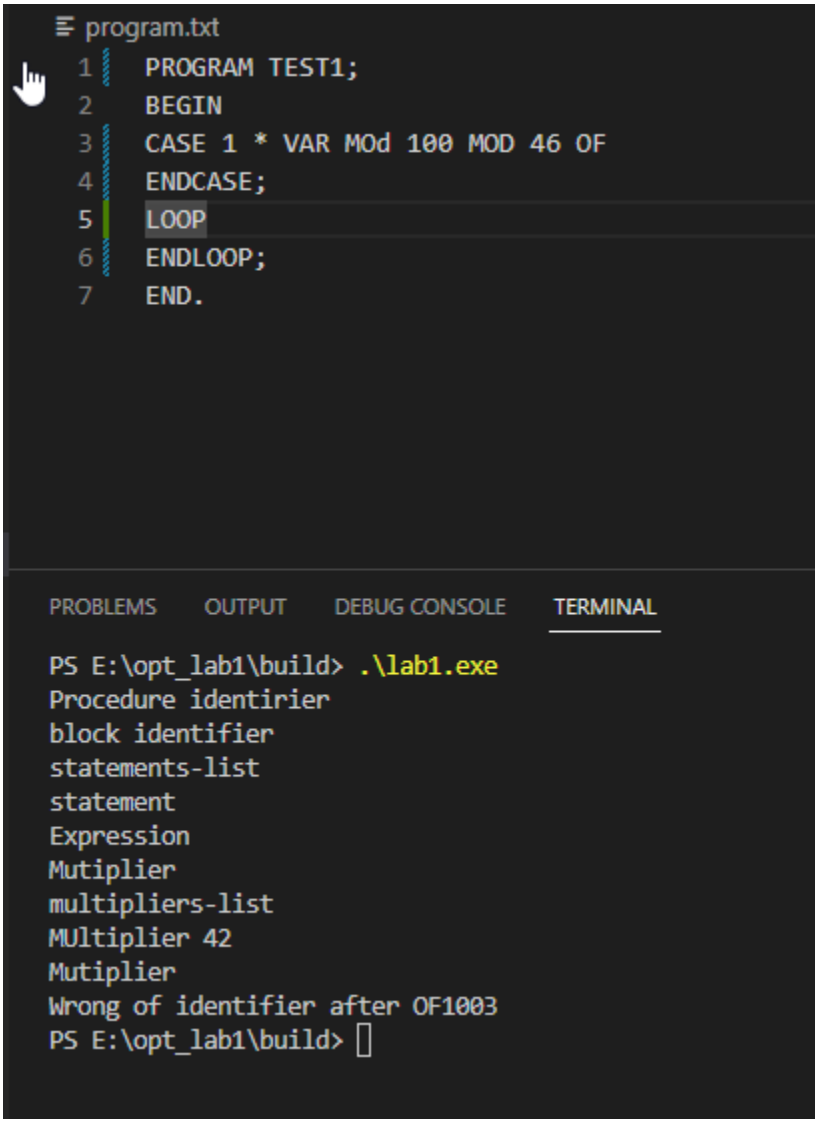
```
1  PROGRAM TEST1;
2  BEGIN
3  CASE 1 * VAR MOD 100 MOD 46 OF
4  ENDCASE;
5  LOOP
6  ENDLOOP;
7  END.
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\opt_lab1\build> .\lab1.exe
Procedure identirier
block identifier
statements-list
statement
Expression
Mutiplier
multipliers-list
MUltiplier 42
Mutiplier
Mutiplier
Mutiplier
alt-list
statement
statements-list
PS E:\opt_lab1\build>
```



Тест 3


program.txt
1 PROGRAM TEST1;
2 BEGIN
3 CASE 1 * VAR MOd 100 MOD 46 OF
4 ENDCASE;
5 LOOP
6 ENDLOOP;
7 END.
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PS E:\opt_lab1\build> .\lab1.exe
Procedure identirier
block identifier
statements-list
statement
Expression
Mutiplier
multipliers-list
MUltiplier 42
Mutiplier
Wrong of identifier after OF1003
PS E:\opt_lab1\build>