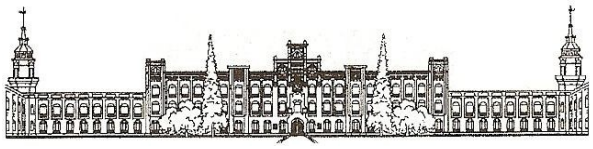

Міністерство освіти і науки України Національний технічний університет України

«Київський політехнічний інститут»

Лабораторна робота №1

з дисципліни «Основи проектування трансляторів»

Виконав студент групи: КВ-22

ПІБ: Крутогуз Максим Ігорович

Перевірив:

_____

Київ 2025

Постановка задачі

1. . Розробити програму лексичного аналізатора (ЛА) 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>*);

   - формування рядка лексем зінформацією про позиції лексем;

   - заповнення таблиць ідентифікаторів та констант інформацією, отриманою підчас згортки лексем;

   - виведення повідомлень про помилки.

○

Варіант 12

```
1.    <signal-program> --> <program>
2.    <program> --> PROCEDURE <procedure-identifier>
                    <parameters-list> ; <block> ;
3.    <block> --> <declarations> BEGIN <statements-list>
                    END
4.    <declarations> --> <label-declarations>
5.    <label-declarations> --> LABEL <unsigned-integer>
                    <labels-list>; |
                    <empty>
6.    <labels-list> --> , <unsigned-integer> <labels-list>
                    |
                    <empty>
7.    <parameters-list> --> ( <variable-identifier>
                    <identifiers-list> ) |
                    <empty>
8.    <identifiers-list> --> , <variable-identifier>
                    <identifiers-list> |
                    <empty>
9.    <statements-list> --> <statement> <statements-list>
                    |
                    <empty>
10.   <statement> --> <unsigned-integer> : <statement> |
                    GOTO <unsigned-integer> ; |
                    RETURN ; |
                    ;  |
                    ($ <assembly-insert-file-identifier> $)
11.   <variable-identifier> --> <identifier>
12.   <procedure-identifier> --> <identifier>
13.   <assembly-insert-file-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
```

Рисунок 1 — Варіант завдання

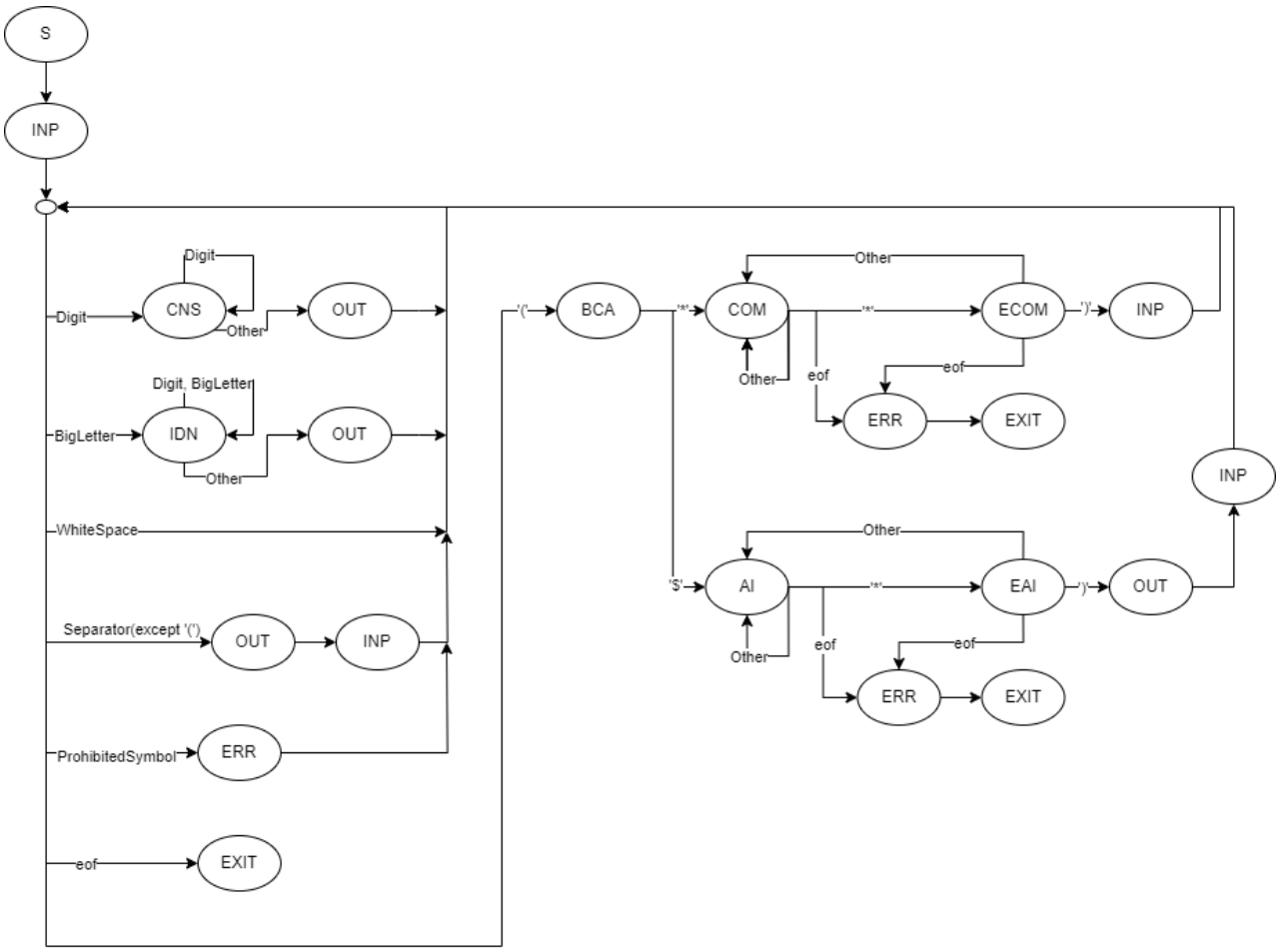

Рисунок 2 — Граф станів лексичного аналізатора

Стан	Назва
CNS	Виділення числової константи
IDN	Виділення ідентифікатора
BCAI	Коментар або вставлення коду мови асемблеру
COM	Виділення коментаря
AI	Виділення асемблевої вставки
ECOM	Визначення кінця коментаря
EAI	Визначення кінця асемблевої вставки

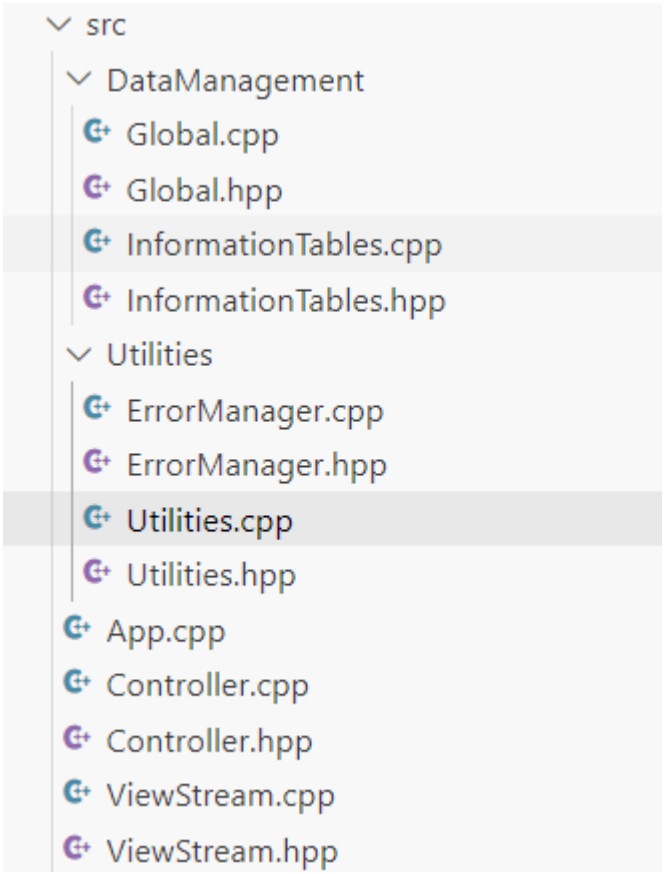

Рисунок 3 — Структура файлів проекту

Код програми:

//Global.hpp

```
#include <map>
#include <string>

#ifndef GLOBAL_HPP
#define GLOBAL_HPP

using namespace std;

extern int SYMBOL_CATEGORIES[];

extern map<short, string> TOKEN_MAP;

enum tokensType {
    NUMBER,
    KEYWORD,
    IDENTIFIER,
    SEPARATOR,
    ASSEMBLY_INSERTION,
    KEYWORD_OR_IDENDIFIER
};
```

```
enum category {
    WHITESPACE,
    CONSTANT_START,
    IDENTIFIER_OR_KEYWORD_START,
    UNIQUE_SEPARATORS,
    AMBIGUES_SEPARATORS,
    PROHIBITED_CHARACTER
};

enum state {
    CNS,
    IDN,
    BCAI,
    COM,
    AI,
    ECOM,
    EAI,
    OUT
};

#endif
```

//Global.cpp

```
#include "Global.hpp"

int SYMBOL_CATEGORIES[] = {
    PROHIBITED_CHARACTER, PROHIBITED_CHARACTER, PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER, PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    WHITESPACE,
    WHITESPACE,
    WHITESPACE,
    WHITESPACE,
    WHITESPACE,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
```

```
PROHIBITED_CHARACTER,
WHITESPACE,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
AMBIGUES_SEPARATORS,
UNIQUE_SEPARATORS,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
UNIQUE_SEPARATORS,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
CONSTANT_START,
CONSTANT_START,
CONSTANT_START,
CONSTANT_START,
CONSTANT_START,
CONSTANT_START,
CONSTANT_START,
CONSTANT_START,
CONSTANT_START,
CONSTANT_START,
UNIQUE_SEPARATORS,
UNIQUE_SEPARATORS,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
PROHIBITED_CHARACTER,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
IDENTIFIER_OR_KEYWORD_START,
PROHIBITED_CHARACTER,
```

```
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
    PROHIBITED_CHARACTER,
};

map<short, string> TOKEN_MAP = {
    {NUMBER, "Number"},
    {KEYWORD, "Keyword"},
    // {CONSTANT, "Constant"},
    {IDENTIFIER, "Identifier"}
};
```

//InformationTables.hpp

```
#include <string>

#include <map>
#include <vector>
#include <iomanip>
#include <iostream>
#include "../ViewStream.hpp"
#include "../Utilities/Utilities.hpp"
#include "Global.hpp"

#ifndef INFORMATAIONTABLES_HPP
#define INFORMATAIONTABLES_HPP
```

```
using namespace std;

typedef struct {
    short type;
    int code;
    int row;
    int col;
} Token;


class InformationTables {
    public:
    InformationTables(ViewStream& vs);
    void initializeTables();

    void outputOneCharacterSeparatorTable();
    void outputMultiCharecterSeparatorTable();
    void outputKeywordTable();
    void outputConstantTable();
    void outputIdentifierTable();
    void outputTokenTable();
    void outputAssemblyInsertionTable();
    void outputAllTables();

    bool isKeyword(string token);

    void addConstant(string constantValue);
    void addIdentifier(string identifierValue);

    void prosessToken(short tokenType, string tokenName, int row, int col);

    private:
    map<string, int> _oneCharacterSeparatorTable;
    map<string, int> _multiCharecterSeparatorTable;
    map<string, int> _keywordTable;
    map<string, int> _constantTable;
    map<string, int> _identifierTable;
    map<string, int> _asssemblyInsertionTable;

    int _nextNumberInOneCharacterSeperatorTable;
    int _nextNumberInMultiCharacterSeperatorTable;
    int _nextNumberInKeywordTable;
    int _nextNumberInConstantTable;
    int _nextNumberInIdentifierTable;

    ViewStream _vs;

    vector<Token> _tokenTable;

    void initializeOneCharacterSeparatorTable();
    void initializeMultiCharecterSeparatorTable();
    void initializeKeywordTable();

    void addToken(Token token);
};

#endif
```

//InformationTables.cpp

```
#include "InformationTables.hpp"

InformationTables::InformationTables(ViewStream& vs) :
_nextNumberInOneCharacterSeperatorTable(0),
_nextNumberInMultiCharacterSeperatorTable(301),
_nextNumberInKeywordTable(601),
_nextNumberInConstantTable(1001),
_nextNumberInIdentifierTable(1000001),
_nextNumberInAssemblyInsertionTable(2000001),
_vs(vs)
{
    initializeTables();
    // We can estimate size of file and reserve size for optimization
    _tokenTable.reserve(100000);
}

void InformationTables::initializeTables() {
    initializeOneCharacterSeparatorTable();
    initializeMultiCharecterSeparatorTable();
    initializeKeywordTable();
}

void InformationTables::initializeOneCharacterSeparatorTable() {
    _oneCharacterSeparatorTable[";"] = 0;
    _oneCharacterSeparatorTable[","] = 1;
    _oneCharacterSeparatorTable["("] = 2;
    _oneCharacterSeparatorTable[")"] = 3;
    _oneCharacterSeparatorTable[":"] = 4;
    _nextNumberInOneCharacterSeperatorTable = 5;
}

void InformationTables::initializeMultiCharecterSeparatorTable() {
    _multiCharecterSeparatorTable["(*"] =
_nextNumberInMultiCharacterSeperatorTable++;
    _multiCharecterSeparatorTable["*)"] =
_nextNumberInMultiCharacterSeperatorTable++;
    _multiCharecterSeparatorTable["($"] =
_nextNumberInMultiCharacterSeperatorTable++;
    _multiCharecterSeparatorTable["$)"] =
_nextNumberInMultiCharacterSeperatorTable++;
}

void InformationTables::initializeKeywordTable() {
    _keywordTable["PROCEDURE"] = _nextNumberInKeywordTable++;
    _keywordTable["BEGIN"] = _nextNumberInKeywordTable++;
    _keywordTable["END"] = _nextNumberInKeywordTable++;
    _keywordTable["LABEL"] = _nextNumberInKeywordTable++;
    _keywordTable["GOTO"] = _nextNumberInKeywordTable++;
    _keywordTable["RETURN"] = _nextNumberInKeywordTable++;
}

void InformationTables::outputAllTables() {
    if (_oneCharacterSeparatorTable.size() > 0) {
        outputOneCharacterSeparatorTable();
        _vs << '\n';
    }
    if (_multiCharecterSeparatorTable.size() > 0) {
```

```
        outputMultiCharecterSeparatorTable();
        _vs << '\n';
    }
    if (_keywordTable.size() > 0) {
        outputKeywordTable();
        _vs << '\n';
    }
    if (_constantTable.size() > 0) {
        outputConstantTable();
        _vs << '\n';
    }
    if (_identifierTable.size() > 0) {
        outputIdentifierTable();
        _vs << '\n';
    }
    if (_asssemblyInsertionTable.size() > 0) {
        outputAssemblyInsertionTable();
        _vs << '\n';
    }
    if (_tokenTable.size() > 0) {
        outputTokenTable();
        _vs << '\n';
    }
}

void InformationTables::outputTokenTable() {
    _vs << "Token table\n";
    _vs << "Type      Code      Row       Col\n";
    for (const auto& pair : _tokenTable) {
        // string str = this->token_map[pair.code];
        _vs << Utilities::getLeftString(pair.type, 10);
        _vs << Utilities::getLeftString(pair.code, 10);
        _vs << Utilities::getLeftString(pair.row, 10);
        _vs << Utilities::getLeftString(pair.col, 10) << '\n';
    }
}

void InformationTables::outputOneCharacterSeparatorTable() {
    _vs << "Onecharacter separator table:\n";
    for (const auto& pair : _oneCharacterSeparatorTable) {
        _vs << pair.first << " -> " << pair.second << '\n';
    }
}

void InformationTables::outputMultiCharecterSeparatorTable() {
    _vs << "Multicharacter separator table:\n";
    for (const auto& pair : _multiCharecterSeparatorTable) {
        _vs << pair.first << " -> " << pair.second << '\n';
    }
}

void InformationTables::outputKeywordTable() {
    _vs << "Keyword table:\n";
    for (const auto& pair : _keywordTable) {
        _vs << pair.first << " -> " << pair.second << '\n';
    }
}

void InformationTables::outputConstantTable() {
    _vs << "Constant table:\n";
    for (const auto& pair : _constantTable) {
```

```
        _vs << pair.first << " -> " << pair.second << '\n';
    }
}

void InformationTables::outputIdentifierTable() {
    _vs << "Identifier table:\n";
    for (const auto& pair : _identifierTable) {
        _vs << pair.first << " -> " << pair.second << '\n';
    }
}

void InformationTables::outputAssemblyInsertionTable() {
    _vs << "Assembly insertion\n";
    for (const auto& pair : _asssemblyInsertionTable) {
        _vs << pair.first << " -> " << pair.second << '\n';
    }
}

void InformationTables::addToken(Token token) {
    _tokenTable.push_back(token);
}


void InformationTables::prosessToken(short tokenType, string tokenName, int row,
int col) {
    int code;
    switch (tokenType) {
        case NUMBER:
            code = _constantTable[tokenName];
            addToken(Token({NUMBER, code, row, col}));
            break;
        case KEYWORD:
            code = _keywordTable[tokenName];
            addToken(Token({KEYWORD, code, row, col}));
            break;
        case IDENTIFIER:
            code = _identifierTable[tokenName];
            addToken(Token({IDENTIFIER, code, row, col}));
            break;
        case SEPARATOR:
            code = _oneCharacterSeparatorTable[tokenName];
            addToken(Token({SEPARATOR, code, row, col}));
            break;
        case ASSEMBLY_INSERTION:
            string trimedTokenName = Utilities::trim(tokenName);

            addAssemblyInsertion(trimedTokenName);

            code = _asssemblyInsertionTable[trimedTokenName];
            addToken(Token({ASSEMBLY_INSERTION, code, row, col}));
            break;
    }
}

void InformationTables::addConstant(string constantValue) {
    if (!_constantTable.count(constantValue)) {
        _constantTable[constantValue] = _nextNumberInConstantTable++;
    }
}

void InformationTables::addIdentifier(string identifierValue) {
```

```
    if (!_identifierTable.count(identifierValue)) {
        _identifierTable[identifierValue] = _nextNumberInIdentifierTable++;
    }
}

void InformationTables::addAssemblyInsertion(string assemblyInsertion) {
    string trimedTokenName = Utilities::trim(assemblyInsertion);

    if (!_asssemblyInsertionTable.count(trimedTokenName)) {
        _asssemblyInsertionTable[assemblyInsertion] =
_nextNumberInAssemblyInsertionTable++;
    }
}

bool InformationTables::isKeyword(string token) {
    auto it = _keywordTable.find(token);

    if (it != _keywordTable.end()) {
        return true;
    } else {
        return false;
    }
}
```

//ErrorManager.hpp

```
#include <string>
#include <vector>
#include "../ViewStream.hpp"

#ifndef ERRORMANAGER_HPP
#define ERRORMANAGER_HPP

using namespace std;

class ErrorManager
{
    public:
    ErrorManager(ViewStream& vs);
    void addProgramError(string errorMessage);
    void addCompilingError(string errorMessage);

    void output();

    private:
    ViewStream _vs;
    vector<string> _programErrorMessages;
    vector<string> _compilingErrorMessages;
};

#endif
```

//ErrorManager.cpp

```
#include <string>
```

```
#include <vector>
#include "../ViewStream.hpp"

#ifndef ERRORMANAGER_HPP
#define ERRORMANAGER_HPP

using namespace std;

class ErrorManager
{
    public:
    ErrorManager(ViewStream& vs);
    void addProgramError(string errorMessage);
    void addCompilingError(string errorMessage);

    void output();

    private:
    ViewStream _vs;
    vector<string> _programErrorMessages;
    vector<string> _compilingErrorMessages;
};

#endif
```

//Utilities.hpp

```
#include <string>

#include <sstream>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include "../DataManagement/Global.hpp"

#ifndef UTILITIES_HPP
#define UTILITIES_HPP

using namespace std;

class Utilities {
  public:
  static string getLeftString(int value, int width);
  static string getLeftString(string value, int width);
  static string getErrorMessage(string filename, int row, int col, string
errorMessage, string problemPart);
  static string trim(const string& str);
};

#endif
```

//Utilities.cpp

```
#include <string>
#include <vector>
#include "../ViewStream.hpp"

#ifndef ERRORMANAGER_HPP
#define ERRORMANAGER_HPP

using namespace std;

class ErrorManager
{
    public:
    ErrorManager(ViewStream& vs);
    void addProgramError(string errorMessage);
    void addCompilingError(string errorMessage);

    void output();

    private:
    ViewStream _vs;
    vector<string> _programErrorMessages;
    vector<string> _compilingErrorMessages;
};

#endif
```

//App.cpp

```
#include "DataManagement/Global.hpp"
#include "Controller.hpp"
#include "ViewStream.hpp"
#include <string>

int main() {
    ViewStream vs;
    Controller("J:/Repositories/University/Translators/Lab1/bin/program2.sig",
vs);
    return 0;
}
```

//Controller.hpp

```
#include <fstream>
#include <iostream>
#include <string>
#include <sstream>
#include "DataManagement/Global.hpp"
#include "ViewStream.hpp"
#include "DataManagement/InformationTables.hpp"
#include "Utilities/ErrorManager.hpp"

#ifndef CONTROLLER_HPP
#define CONTROLLER_HPP
```

```
using namespace std;

class Controller {
    public:
    Controller(string filepath, ViewStream vs);
    void run(string);

    private:
    ViewStream _vs;
    InformationTables _it;
    ErrorManager _em;
};

#endif
```

//Controller.cpp

```
#include "Controller.hpp"

Controller::Controller(string filepath, ViewStream vs) :
_vs(vs),
_it(InformationTables(vs)),
_em(ErrorManager(vs))
{
    this->run(filepath);
}

void Controller::run(string filepath) {
    ifstream inputFile(filepath);
    ostringstream buffer;

    try {
        if (inputFile.is_open()) {
            char symbol;
            int currentRow = 1;
            int currentCol = 1;
            bool isReadingAllowed = true;
            bool isLoopActive;
            while (!isReadingAllowed || inputFile.get(symbol)) {
                if (inputFile.eof()) {
                    break;
                }
                isReadingAllowed = true;
                switch (SYMBOL_CATEGORIES[(short)symbol]) {
                    case WHITESPACE:
                        switch (symbol)
                        {
                            case '\n':
                                currentCol = 1;
                                currentRow++;
                                break;
                            case '\t':
                                currentCol += 4;
                                break;
                            default:
```

```
                                currentCol++;
                                break;
                        }
                        break;
                    case CONSTANT_START:
                        do {
                        buffer << symbol;
                        } while (inputFile.get(symbol) &&
SYMBOL_CATEGORIES[(short)symbol] == CONSTANT_START);
                        _it.addConstant(buffer.str());
                        _it.prosessToken(NUMBER, buffer.str(), currentRow,
currentCol);
                        currentCol += buffer.str().length();
                        buffer.str("");
                        isReadingAllowed = false;
                        break;
                    case IDENTIFIER_OR_KEYWORD_START:
                        do {
                            buffer << symbol;

                        } while (inputFile.get(symbol) &&
(SYMBOL_CATEGORIES[(short)symbol] == CONSTANT_START ||
SYMBOL_CATEGORIES[(short)symbol] == IDENTIFIER_OR_KEYWORD_START));
                        if (_it.isKeyword(buffer.str())) {
                            _it.prosessToken(KEYWORD, buffer.str(), currentRow,
currentCol);
                        } else {
                            _it.addIdentifier(buffer.str());
                            _it.prosessToken(IDENTIFIER, buffer.str(),
currentRow, currentCol);
                        }
                        currentCol += buffer.str().length();
                        buffer.str("");
                        isReadingAllowed = false;
                        break;
                    case UNIQUE_SEPARATORS:
                        buffer << symbol;
                        _it.prosessToken(SEPARATOR, buffer.str(), currentRow,
currentCol);
                        buffer.str("");
                        currentCol++;
                        break;
                    case AMBIGUES_SEPARATORS:
                        buffer << symbol;
                        inputFile.get(symbol);
                        currentCol++;
                        short state;
                        isLoopActive = true;
                        int startRow;
                        int startCol;
                        switch (symbol) {
                            case '*':
                                buffer.str("");
                                state = COM;
                                while (isLoopActive) {
                                    if (!inputFile.get(symbol)) {
                                        throw
Utilities::getErrorMessage(filepath, currentRow, currentCol, "Not closed
commentary", "");
                                    }
```

```
                                switch (symbol) {
                                    case '\t':
                                        currentCol += 4;
                                        break;
                                    case '\n':
                                        currentCol = 1;
                                        currentRow++;
                                        break;
                                    default:
                                        currentCol++;
                                }
                                switch (state) {
                                    case COM:
                                        if (symbol == '*') {
                                            state = ECOM;
                                        }
                                        break;
                                    case ECOM:
                                        if (symbol == ')') {
                                            isLoopActive = false;
                                        } else {
                                            state = COM;
                                        }
                                }
                            }
                            break;
                        case '$':
                            buffer.str("");
                            state = AI;
                            startRow = currentRow;
                            startCol = currentCol + 1;
                            while (isLoopActive) {
                                if (!inputFile.get(symbol)) {
                                    throw
Utilities::getErrorMessage(filepath, currentRow, currentCol, "Not closed
assembly insertion", "");
                                }

                                switch (symbol) {
                                    case '\t':
                                        currentCol += 4;
                                        break;
                                    case '\n':
                                        currentCol = 1;
                                        currentRow++;
                                        break;
                                    default:
                                        currentCol++;
                                }
                                bool isDollarMissed = false;
                                switch (state) {
                                    case AI:
                                        if (symbol == '$') {
                                            state = EAI;
                                            isDollarMissed = true;
                                        } else {
                                            buffer << symbol;
                                        }
                                        break;
                                    case EAI:
                                        if (symbol == ')') {
```

```
                                                isLoopActive = false;

_it.prosessToken(ASSEMBLY_INSERTION, buffer.str(), startRow, startCol);
                                                buffer.str("");

                                            } else {
                                                state = AI;
                                                if (isDollarMissed) {
                                                    buffer << '$';
                                                    isDollarMissed = false;
                                                }
                                                if (symbol == '$') {
                                                    isDollarMissed = true;
                                                    state = EAI;
                                                } else {
                                                    buffer << symbol;
                                                }
                                            }
                                        }
                                    }
                                    break;
                                default:
                                    _it.prosessToken(SEPARATOR, buffer.str(),
currentRow, currentCol - 1);
                                    buffer.str("");
                                    // currentCol++;
                                    isReadingAllowed = false;
                                    break;
                            }
                            break;

                        case PROHIBITED_CHARACTER:
                        default:

_em.addCompilingError(Utilities::getErrorMessage(filepath, currentRow,
currentCol, "Used prohibited character:", string(1, (char)symbol)));
                            currentCol++;
                            break;
                    }
                }
            } else {
                throw "Reading file error";
            }
        } catch (string erorrMessage) {
            _em.addProgramError(erorrMessage);
        }

        _it.outputAllTables();
        _em.output();
    }
```

//ViewStream.hpp

```
#include <sstream>

#include <iostream>
#include <string>

#ifndef VIEWSTEAM_HPP
```

```
#define VIEWSTEAM_HPP

using namespace std;

class ViewStream {
    public:
    // std::ostringstream buffer;

    ViewStream& operator<<(string data);
    ViewStream& operator<<(const char data);
    ViewStream& operator<<(int data);

    // template <typename T>
    // ViewStream& operator<<(T data);
};

#endif
```

//ViewStream.cpp

```
#include <sstream>
#include <iostream>
#include <string>

#ifndef VIEWSTEAM_HPP
#define VIEWSTEAM_HPP

using namespace std;

class ViewStream {
    public:
    // std::ostringstream buffer;

    ViewStream& operator<<(string data);
    ViewStream& operator<<(const char data);
    ViewStream& operator<<(int data);

    // template <typename T>
    // ViewStream& operator<<(T data);
};

#endif
```

Тестування:

program1t.sig | program1f.sig | program2t.sig | program2f.sig | App.cpp | Controller.hpp | Controller.cpp | Keyboard Shortcuts

bin > program1t.sig

```
1  34 HE1  RETURN
2  LABEL
```

PROBLEMS OUTPUT TERMINAL PORTS GITLENS COMMENTS

TERMINAL powershell

```
; -> 0

Multicharacter separator table:
$) -> 304
($ -> 303
(* -> 301
*) -> 302

Keyword table:
BEGIN -> 602
END -> 603
GOTO -> 605
LABEL -> 604
PROCEDURE -> 601
RETURN -> 606

Constant table:
34 -> 1001

Identifier table:
HE1 -> 1000001

Token table
Type      Code      Row       Col
0         1001      1         2
2         1000001   1         5
1         606       1         10
1         604       2         1

There are no errors in a program.
PS J:\Repositories\University\Translators\Lab1>
```

Рисунок 4 — Перший True тест

program1t.sig | program1f.sig | program2t.sig | program2f.sig | App.cpp | Controller.hpp | Controller.cpp | Keyboard Shortcuts

bin > program1f.sig

```
1  34 1D f  RETURN
2  LABEL
```

PROBLEMS OUTPUT TERMINAL PORTS GITLENS COMMENTS

TERMINAL powershell

```
$) -> 304
($ -> 303
(* -> 301
*) -> 302

Keyword table:
BEGIN -> 602
END -> 603
GOTO -> 605
LABEL -> 604
PROCEDURE -> 601
RETURN -> 606

Constant table:
1 -> 1002
34 -> 1001

Identifier table:
D -> 1000001

Token table
Type      Code      Row       Col
0         1001      1         2
0         1002      1         5
2         1000001   1         6
1         606       1         11
1         604       2         1

There are 1 in the program:
J:/Repositories/University/Translators/Lab1/bin/program1f.sig:1:8: error: Used prohibited character: f
PS J:\Repositories\University\Translators\Lab1>
```

Рисунок 5 — Перший False тест

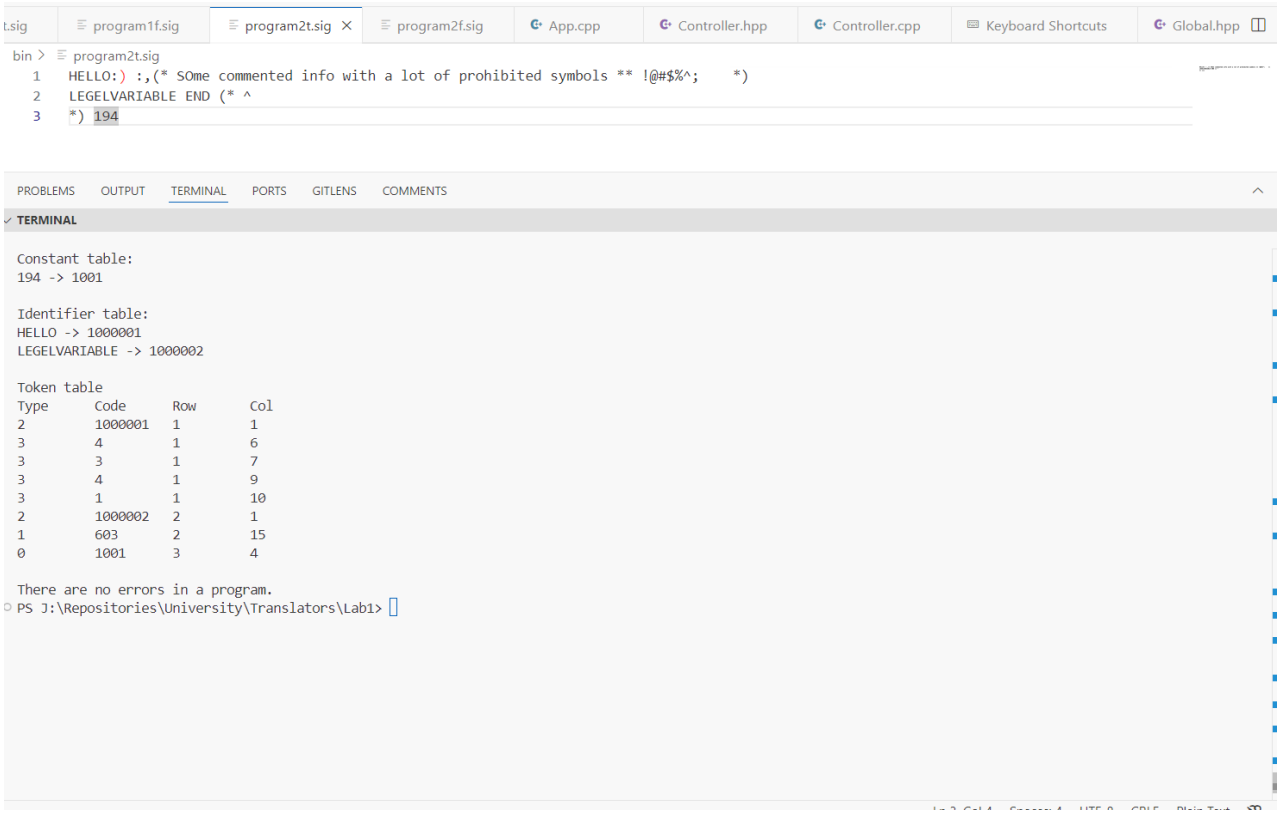

Рисунок 6 — Другий True тест

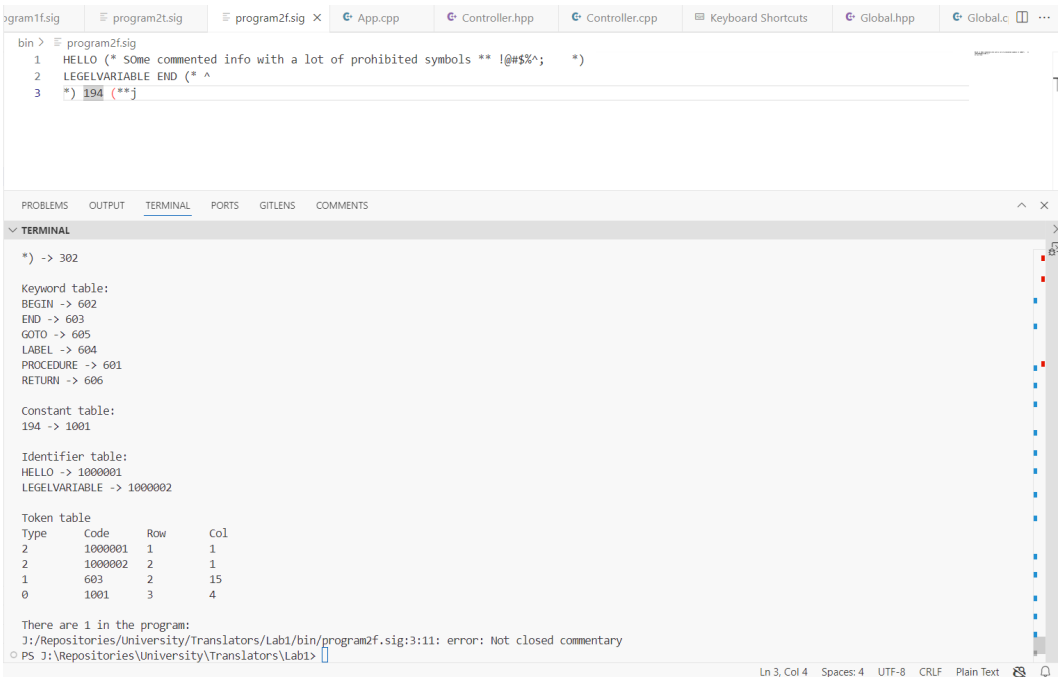

Рисунок 7 — Другий False тест

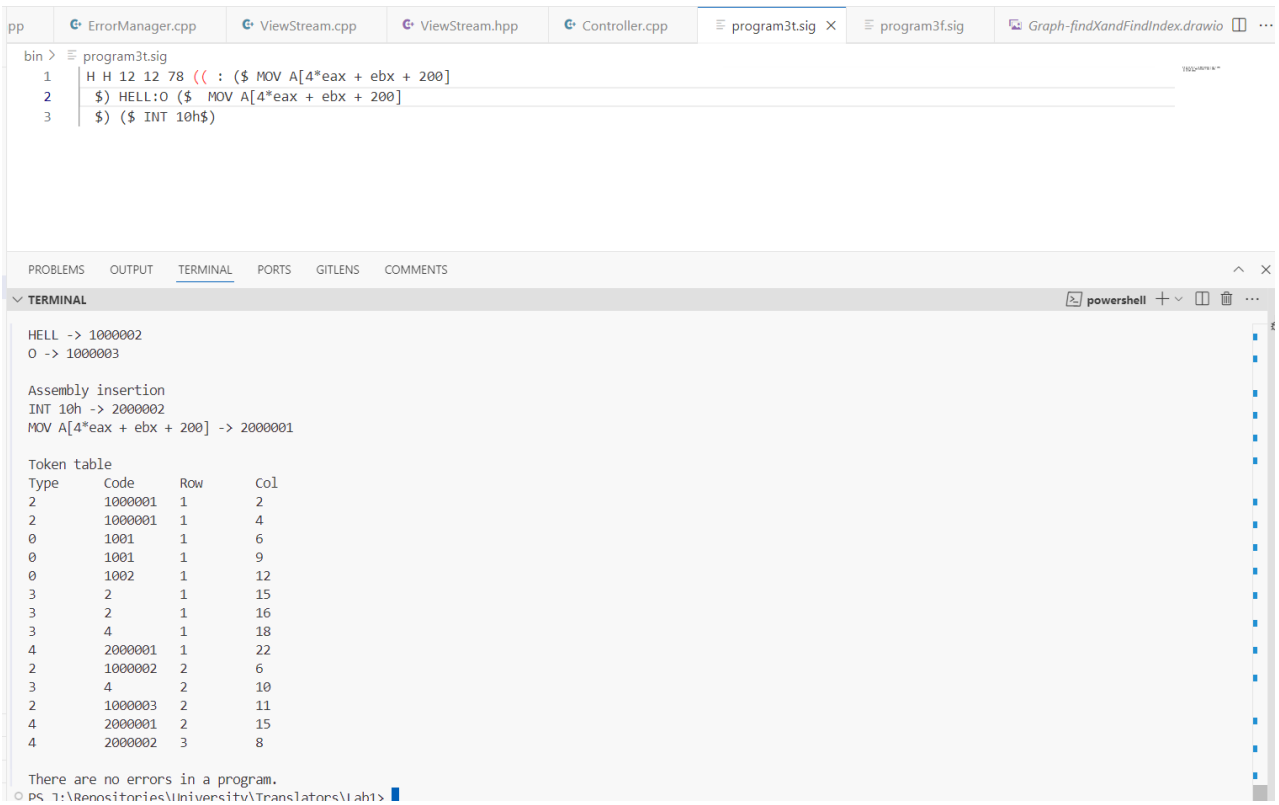

Рисунок 8 — Третій True тест

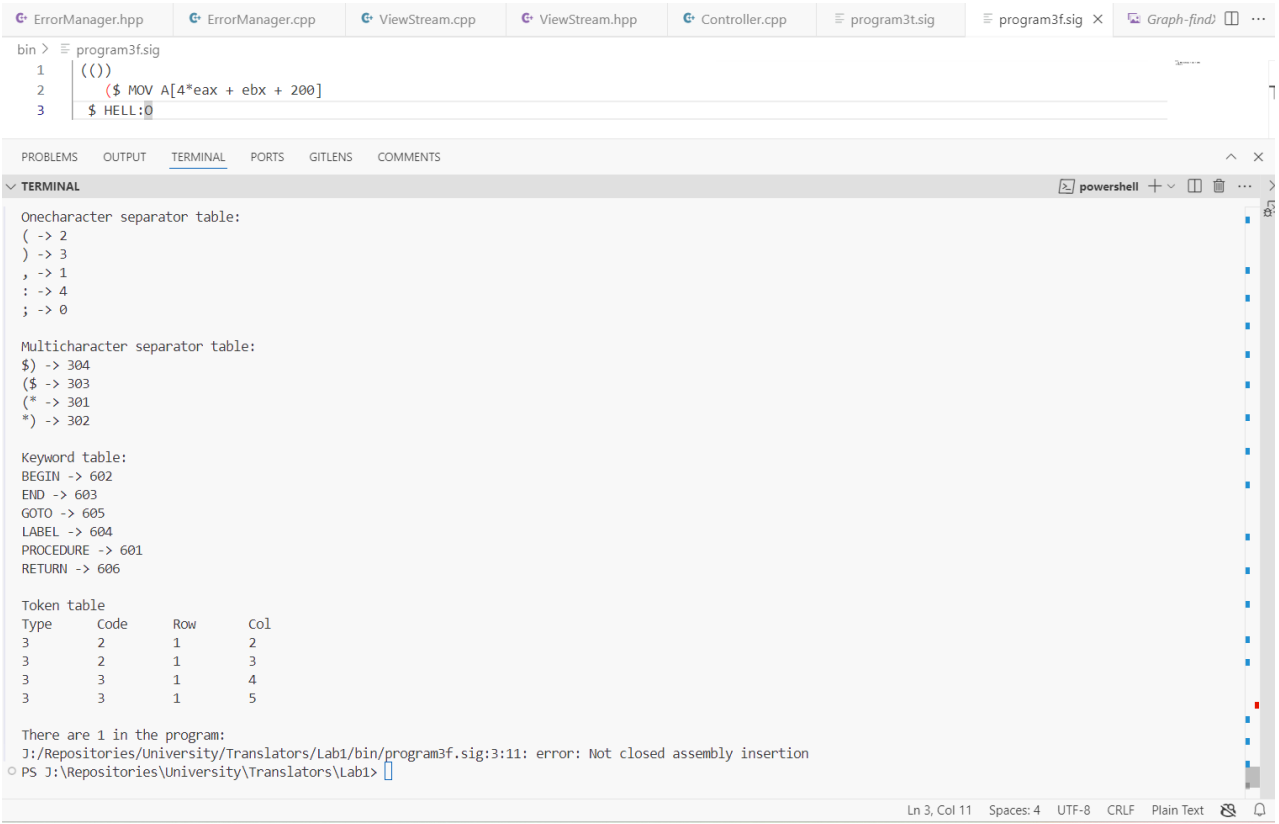

Рисунок9 — Третій False тест