МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ | Факультет прикладной
математики и информатики

Кафедра прикладной математики

Практическое задание № 2

по дисциплине «ЯПМТ»

РАЗРАБОТКА И РЕАЛИЗАЦИЯ БЛОКА ЛЕКСИЧЕСКОГО АНАЛИЗА

Бригада 1	БАРАНОВ ЯРОСЛАВ
Группа ПМ-92	МАКАРЫЧЕВ СЕРГЕЙ
Вариант 1	ЮЗЯК МАРИНА

Преподаватели ЕЛАНЦЕВА И.Л.

Новосибирск, 2022

1. **Цель работы**

Изучить методы лексического анализа. Получить представление о методах обработки лексических ошибок. Научиться проектировать сканер на основе детерминированных конечных автоматов.

2. **Задание**

Подмножество языка C++ включает:

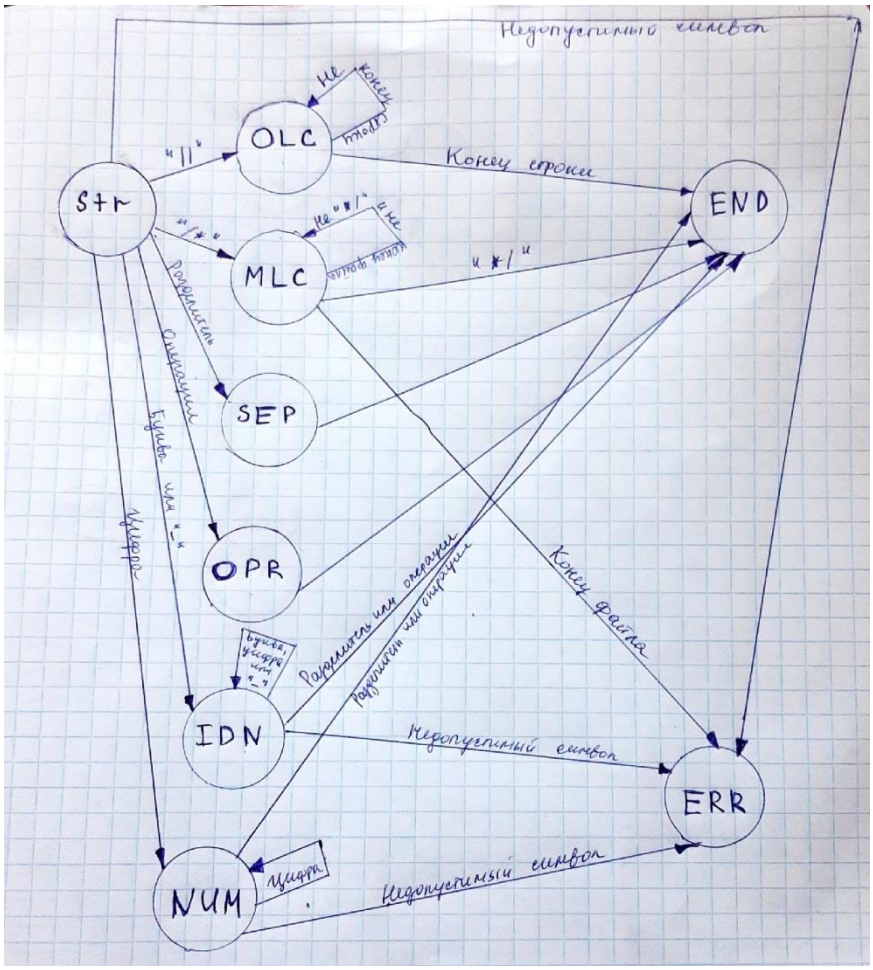
- данные типа **int**;
- инструкции описания переменных;
- операторы присваивания, **if**, **if- else** любой вложенности и в любой последовательности;
- операции **+, – , *, = =, != , <** .

В соответствии с выбранным вариантом задания к лабораторным работам разработать и реализовать лексический анализатор на основе детерминированных конечных автоматов. Исходными данными для сканера является программа на языке C++ и постоянные таблицы, реализованные в лабораторной работе №1. Результатом работы сканера является создание файла токенов, переменных таблиц (таблицы символов и таблицы констант) и файла сообщений об ошибках.

3. **Структура входных и выходных данных**

Сканер работает на основе таблиц зарезервированных слов. Входные данные – текст программы на языке C++. Выходные данные – файл с ошибками и файл с токенами.

4. **Детерминированный конечный автомат**



Описание состояний:

STR – начальное состояние

OLC – однострочный комментарий

MLC – многострочный комментарий

SEP – разделитель

OPR – операция

IDN – идентификатор или ключевое слово

NUM – цифра

END – конечное состояние

ERR – ошибка

5. Алгоритм разбора

1. Считать строку. Если конец файла – перейти к шагу 12.
2. Считать первое слово строки до пробела.
3. Считать символ слова. Если незакрытый многострочный комментарий – перейти к шагу 8, иначе перейти к шагу 4.
4. Если первый символ буква или «_», перейти к шагу 5; цифра – перейти к шагу 6; слэш – перейти к шагу 7; разделитель – перейти к шагу 9; операция – перейти к шагу 10; если конец строки – на шаг 1. Иначе на шаг 11.
5. Выделить идентификатор путем добавления к первому символу всех последующих букв и цифр. Если идентификатор – ключевое слово, сформировать и вывести соответствующий токен, иначе добавить идентификатор в таблицу идентификаторов и вывести соответствующий токен. За слово считать слово после идентификатора и перейти к шагу 4.
6. Выделить константу путем добавления к первому символу всех последующих цифр. Если следующий символ после выделенной константы некорректен – перейти на шаг 11. Сформировать и вывести соответствующий токен для константы. За слово считать слово после константы и перейти к шагу 4.
7. Если следующий символ «/» - переходим на шаг 1. Если следующий символ «*» - многострочный комментарий открыт, переходим на шаг 8.
8. Ищем конец комментария в текущем слове, если нет – переходим на шаг 2. Если конца комментария нет и конец файла – переходим на шаг 11. Если комментарий закрыт, переходим на шаг 4.
9. Выделить разделитель, сформировать и вывести соответствующий токен. За слово считать слово после разделителя и перейти к шагу 4.
10. Выделить одно- или двухсимвольную операцию, сформировать и вывести соответствующий токен. За строку считать строку после операции и перейти к шагу 4.
11. Ошибка.
12. Конец разбора.

6. Тесты

1. Верный исходный код с разнообразным форматированием и стилем комментариев:

```
int main {
/*jjj */

   int abc = 1; // ~7rhhff
int a;
if (abc == 0)
        a *=abc;
else
abc -= a;

/*fjf
*ff
*/ return 0;
}
```

Constant table

Num in table	Value
0	0
1	1

Identifier table

Num in table	Name	Type	Value
1	a	non	non
3	abc	non	non

Файл токенов:

```
(2,2) (2,3) (4,4)
(2,2) (0,3) (3,6) (1,1) (4,3)
(2,2) (0,1) (4,3)
(2,1) (4,0) (0,3) (3,5) (1,0) (4,1)
(0,1) (3,7) (0,3) (4,3)
(2,0)
(0,3) (3,9) (0,1) (4,3)
(2,4) (1,0) (4,3)
(4,5)
```

Файл ошибок: пуст.

2. Незакрытый многострочный комментарий:

```
int main {
/*jjj

   abc = 1; // juygg
}
```

Constant table

Num in table	Value

Identifier table

Num in table	Name	Type	Value

Файл токенов:

```
(2,2) (2,3) (4,4)
```

Файл ошибок:

```
Error in line 2: unclosed multiline comment
```

3. *Незакрытый однострочный комментарий:*

```
int main {
/jjj

   abc = 1; // juygg
}
```

Constant table

Num in table	Value
1	1

Identifier table

Num in table	Name	Type	Value
3	abc	non	non

Файл токенов:

```
(2,2) (2,3) (4,4)
(0,3) (3,6) (1,1) (4,3)
(4,5)
```

Файл ошибок:

```
Error in line 2: /jjj
```

4. *Недопустимые символы в коде:*

```
int main {
/*jjj */

@@
   abc = 1; // kjjjj
  int ggg;
  int kkk;
}
```

Constant table

Num in table	Value
1	1

Identifier table

Num in table	Name	Type	Value
3	abc	non	non
7	ggg	non	non
11	kkk	non	non

Файл токенов:

```
(2,2) (2,3) (4,4)
(0,3) (3,6) (1,1) (4,3)
(2,2) (0,7) (4,3)
(2,2) (0,11) (4,3)
(4,5)
```

Файл ошибок:

```
Error in line 4: @@
```

5. *Некорректный идентификатор:*

```
int main {
/*jjj */

    a@bc = 1; //khhjkkj
}
```

```
Constant table
|---------------|---------------|
|  Num in table |     Value     |
|---------------|---------------|
|1              |1              |
|---------------|---------------|

Identifier table
|---------------|---------------|---------------|---------------|
|  Num in table |     Name      |      Type     |     Value     |
|---------------|---------------|---------------|---------------|
|1              |a              |non            |non            |
|---------------|---------------|---------------|---------------|
```

Файл токенов:

```
(2,2) (2,3) (4,4)
(0,1) (3,6) (1,1) (4,3)
(4,5)
```

Файл ошибок:

```
Error in line 4: a@bc
```

6. *Некорректная константа:*

```
int main {
int a, b, c;

a = 123b;
}
```

```
Constant table
|---------------|---------------|
|  Num in table |     Value     |
|---------------|---------------|

Identifier table
|---------------|---------------|---------------|---------------|
|  Num in table |     Name      |      Type     |     Value     |
|---------------|---------------|---------------|---------------|
|1              |a              |non            |non            |
|---------------|---------------|---------------|---------------|
|2              |b              |non            |non            |
|---------------|---------------|---------------|---------------|
|3              |c              |non            |non            |
|---------------|---------------|---------------|---------------|
```

Файл токенов:

```
(2,2) (2,3) (4,4)
(2,2) (0,1) (4,2) (0,2) (4,2) (0,3) (4,3)
(0,1) (3,6) (4,3)
(4,5)
```

Файл ошибок:

```
Error in line 4: 123b;
```

7. **Код программы**

main.cpp

```
#include "Translator.h"

int main()
{

	using ::std::filesystem::path;
```

```
        path dir_to_file = "files/test-6";
        Translator tarnslator(dir_to_file);

        return 0;
}
```

Tables.h

```
#pragma once
#include <vector>
#include <iostream>
#include <vector>
#include "data.h"

int Find(const std::vector<std::string>& arr, std::string);

class HashTableVar
{
private:
    static const int default_size = 16;
    std::vector<Variable> arr;
    size_t size;
    int HashFunction(const Variable& v, int table_size);

public:
    HashTableVar();
    ~HashTableVar();
    bool Add(const Variable& value);
    int Find(const Variable& value);
    void SetType(int n, const std::string& type) { arr[n].type = type; }
    void SetValue(int n, const std::string& value) { arr[n].value = value; }
    const Variable& Get(int n) { return arr[n]; }
    void printTable();
};

class HashTableConst
{
private:
    static const int default_size = 16;
    std::vector<std::string> arr;
    size_t size;
    int HashFunction(const std::string& v, int table_size);

public:
    HashTableConst();
    ~HashTableConst();
    bool Add(const std::string& value);
    int Find(const std::string& value);
    const std::string& Get(int n) { return arr[n]; }
    void printTable();
};
```

Tables.cpp

```
#include "Tables.h"

int HashTableConst::HashFunction(const std::string& s, int table_size)
{
    int hash_result = 0, a = 31415, b = 27183;
    for (size_t i = 0; i < s.size(); i++, a = a * b % (table_size - 1))
        hash_result = (a * hash_result + s[i]) % table_size;
    return hash_result;
}
```

```
HashTableConst::HashTableConst()
{
    arr.resize(default_size);
    size = 0;
}

HashTableConst::~HashTableConst() {}

bool HashTableConst::Add(const std::string& value)
{
    if (size + 1 > arr.size())
        arr.resize(2 * arr.size());

    int h1 = HashFunction(value, arr.size());
    int h2 = (h1 * 2 + 1) % arr.size();

    for (size_t i = 0; arr[h1] != "" && i < arr.size(); i++, h1 = (h1 + h2) %
arr.size())
        if (arr[h1] == value)
            return false;

    arr[h1] = value;
    size++;
    return true;
}

int HashTableConst::Find(const std::string& value)
{
    int h1 = HashFunction(value, arr.size());
    int h2 = (h1 * 2 + 1) % arr.size();

    for (size_t i = 0; arr[h1] != "" && i < arr.size(); i++, h1 = (h1 + h2) %
arr.size())
        if (arr[h1] == value)
            return h1;
    return -1;
}

int HashTableVar::HashFunction(const Variable& v, int table_size)
{
    int hash_result = 0, a = 31415, b = 27183;
    for (size_t i = 0; i < v.name.size(); i++, a = a * b % (table_size - 1))
        hash_result = (a * hash_result + v.name[i]) % table_size;
    return hash_result;
}

HashTableVar::HashTableVar()
{
    arr.resize(default_size);
    size = 0;
}

HashTableVar::~HashTableVar() {}

bool HashTableVar::Add(const Variable& value)
{
    if (size + 1 > arr.size())
        arr.resize(2 * arr.size());

    int h1 = HashFunction(value, arr.size());
    int h2 = (h1 * 2 + 1) % arr.size();

    for (size_t i = 0; arr[h1].name != "" && i < arr.size(); i++, h1 = (h1 + h2) %
arr.size())
        if (arr[h1].name == value.name)
```

```
            return false;

    arr[h1] = value;
    size++;
    return true;
}

int HashTableVar::Find(const Variable& value)
{
    int h1 = HashFunction(value, arr.size());
    int h2 = (h1 * 2 + 1) % arr.size();

    for (size_t i = 0; arr[h1].name != "" && i < arr.size(); i++, h1 = (h1 + h2) %
arr.size())
        if (arr[h1].name == value.name)
            return h1;
    return -1;
}

int Find(const std::vector<std::string>& arr, std::string s)
{
    for (size_t i = 0; i < arr.size(); i++)
        if (arr[i] == s)
            return i;

    return -1;

}

void HashTableConst::printTable()
{
    std::cout << "\nConstant table\n";
    std::cout << "|---------------|---------------|\n";
    std::cout << "|  Num in table |     Value     |\n";
    std::cout << "|---------------|---------------|\n";

    int index = 0;
    for (size_t i = 0; i < arr.size(); i++)
        if ((index = Find(arr[i])) != -1)
        {
            std::cout << "|" << index  << "\t\t|" << arr[index] << "\t\t|" <<
std::endl;
            std::cout << "|---------------|---------------|\n";
        }
}

void HashTableVar::printTable()
{
    std::cout << "\nIdentifier table\n";
    std::cout << "|---------------|---------------|---------------|---------------
|\n";
    std::cout << "|  Num in table |     Name      |      Type     |     Value
|\n";
    std::cout << "|---------------|---------------|---------------|---------------
|\n";

    int index = 0;
    for (size_t i = 0; i < arr.size(); i++)
        if ((index = Find(arr[i])) != -1)
        {
            std::cout << "|" << index << "\t\t|" << arr[index].name << "\t\t|"
                << arr[index].type << "\t\t|" << arr[index].value << "\t\t|" <<
std::endl;
            std::cout << "|---------------|---------------|---------------|--------
------|\n";
```

```
        }
}
```

Translator.h

```
#pragma once
#include "data.h"
#include "Tables.h"

#include <filesystem>
#include <array>
#include <vector>

#define _PRINT_DEBUG

using std::filesystem::path;

class Translator
{
public:
	Translator(const path& _path)
	{
		lexicalAnalysis(_path);
		#ifdef _PRINT_DEBUG
		constant_tabel.printTable();
		identifier_tabel.printTable();
		#endif
	}

private:
	void lexicalAnalysis(const path&);
	size_t errorHandler(std::string str); // возвращает номер символа в слове, на
котором остановился

private:
	const std::vector<std::string> keywords{ "else", "if", "int", "main",
"return","void" };
	const std::vector<std::string> separators{ "(", ")",  ",", ";","{", "}" };
	const std::vector<std::string> operations{ "!=", "*", "+", "-", "<", "==",
"=", "*=", "+=", "-=" };

	HashTableVar identifier_tabel;
	HashTableConst constant_tabel;
};
```

Translator.cpp

```
#include "Translator.h"

#include <fstream>
#include <iostream>

void Translator::lexicalAnalysis(const path& _path)
{
	std::ofstream fout_errors(_path / "errors.txt");
	std::ofstream fout_tokens(_path / "tokens.txt");

	std::ifstream fin(_path / "program.txt");
	if (fin.is_open())
	{
		std::string str;
		std::string first_symbol;
		std::string second_symbol;
```

```
		std::string current_symbol;
		std::string identifier;
		std::vector<std::string> source_line;
		size_t line = 0;
		size_t line_of_multiline_comment;
		int number_in_table;
		bool out_of_multiline_comment = true;

		while (std::getline(fin, str))
		{
			line++;
			std::stringstream sstr;		sstr << str;
			source_line.clear();
			while (sstr >> str)
				source_line.push_back(str);

			first_symbol = " ";
			for (size_t i = 0; i < source_line.size(); i++)
			{
				str = source_line[i];
				for (size_t j = 0; j < str.size(); j++)
				{
					first_symbol = str[j];
					if (!out_of_multiline_comment)
					{
						str.size() == j + 1 ? second_symbol = " " :
second_symbol = str[j + 1];
						if (first_symbol == "*" && second_symbol ==
"/")
						{
							out_of_multiline_comment = true;
							str.size() == ++j + 1 ? first_symbol =
" " : first_symbol = str[j + 1];
						}
					}

					if (out_of_multiline_comment && first_symbol != " ")
					{
						SymbolType first_symbol_type =
SymbolType::DEFAULT;

						if (("a" <= first_symbol && first_symbol <=
"z") || ("A" <= first_symbol && first_symbol <= "Z") || first_symbol == "_")
							first_symbol_type =
SymbolType::LETTER_OR_UNDERSCORE;
						else if ("0" <= first_symbol && first_symbol
<= "9")
							first_symbol_type = SymbolType::NUMBER;

						else if (first_symbol == "/")
							first_symbol_type =
SymbolType::FORWARD_SLASH;
						else if ((number_in_table = Find(separators,
first_symbol)) != -1)
							first_symbol_type =
SymbolType::SEPARATOR;
						else
						{
							str.size() == j + 1 ? second_symbol = "
" : second_symbol = str[j + 1];

							if ((number_in_table = Find(operations,
first_symbol + second_symbol)) != -1)
							{
```

```
                                                first_symbol_type =
SymbolType::OPERATION;
                                                j++;
                                          }
                                          else if ((number_in_table =
Find(operations, first_symbol)) != -1)
                                                first_symbol_type =
SymbolType::OPERATION;
                                    }

                                    switch (first_symbol_type)
                                    {
                                    case SymbolType::LETTER_OR_UNDERSCORE:
                                    {
                                          identifier = first_symbol;
                                          for (size_t i = j + 1; i < str.size();
i++, j++)
                                          {
                                                current_symbol = str[i];
                                                if ("a" <= current_symbol &&
current_symbol <= "z" || "A" <= current_symbol && current_symbol <= "Z"
                                                      || "0" <= current_symbol
&& current_symbol <= "9" || current_symbol == "_")
                                                      identifier +=
current_symbol;
                                                else
                                                      break;
                                          }

                                          Variable variable{ identifier, "non",
"non" };
                                          int in_keywords_tabel = Find(keywords,
identifier);
                                          int in_identifier_tabel =
identifier_tabel.Find(variable);

                                          if (in_keywords_tabel == -1)
                                          {
                                                if (in_identifier_tabel == -1)
                                                {

      identifier_tabel.Add(variable);
                                                      in_identifier_tabel =
identifier_tabel.Find(variable);
                                                }
                                                fout_tokens << "(" <<
(int)TypesOfTables::VARIABLE_TABEL << ","
                                                      << in_identifier_tabel <<
") ";
                                          }
                                          else
                                                fout_tokens << "(" <<
(int)TypesOfTables::KEYWORDS_TABLE << ","
                                                << in_keywords_tabel << ") ";
                                    }
                                    break;
                                    case SymbolType::NUMBER:
                                    {
                                          identifier = first_symbol;
                                          for (size_t i = j + 1; i < str.size();
i++, j++)
                                          {
                                                current_symbol = str[i];
```

```
                                                        if ("0" <= current_symbol &&
current_symbol <= "9")
                                                              identifier +=
current_symbol;
                                                        else
                                                              break;
                                                  }
                                                  if (!("0" <= current_symbol &&
current_symbol <= "9")
                                                        && Find(operations,
current_symbol) == -1 && Find(separators, current_symbol) == -1)
                                                  {
                                                        j = errorHandler(str.substr(j));
                                                        fout_errors << "Error in line "
<< line << ": " << str << std::endl;
                                                  }
                                                  else
                                                  {
                                                        int in_constant_tabel =
constant_tabel.Find(identifier);

                                                        if (in_constant_tabel == -1)
                                                        {

      constant_tabel.Add(identifier);
                                                              in_constant_tabel =
constant_tabel.Find(identifier);
                                                        }
                                                        fout_tokens << "(" <<
(int)TypesOfTables::CONSTATN_TABEL << ","
                                                              << in_constant_tabel << ")
";
                                                  }
                                            }
                                            break;
                                            case SymbolType::FORWARD_SLASH:
                                            {
                                                  str.size() == j + 1 ? second_symbol = "
" : second_symbol = str[++j];
                                                  if (second_symbol == "*")
                                                  {
                                                        out_of_multiline_comment = false;
                                                        line_of_multiline_comment = line;
                                                        for (size_t i = j + 1; i <
str.size(); i++, j++)
                                                        {
                                                              current_symbol = str[i];
                                                              if (current_symbol == "*")
                                                              {
                                                                    str.size() == i + 1
? second_symbol = " " : second_symbol = str[i + 1];
                                                                    if (second_symbol ==
"/")
                                                                    {

      out_of_multiline_comment = true;
                                                                          break;
                                                                    }
                                                              }
                                                        }
                                                  }
                                                  else if (second_symbol == "/")
                                                  {
                                                        j = str.size();
                                                        i = source_line.size();
```

```
                                                }
                                                else
                                                {
                                                        fout_errors << "Error in line "
<< line << ": " << str << std::endl;
                                                        j = str.size();
                                                        i = source_line.size();
                                                }
                                        }
                                        break;
                                        case SymbolType::SEPARATOR:
                                                fout_tokens << "(" <<
(int)TypesOfTables::SEPARATORS_TABEL << ","
                                                        << number_in_table << ") ";
                                                break;
                                        case SymbolType::OPERATION:
                                                fout_tokens << "(" <<
(int)TypesOfTables::OPERATIONS_TABEL << ","
                                                        << number_in_table << ") ";
                                                break;
                                        default:
                                                j = errorHandler(str.substr(j));
                                                fout_errors << "Error in line " << line
<< ": " << str << std::endl;
                                                break;
                                        }
                                }
                        }
                        if (out_of_multiline_comment && first_symbol != " ")
                                fout_tokens << std::endl;
                }
                if (!out_of_multiline_comment)
                        fout_errors << "Error in line " << line_of_multiline_comment <<
": unclosed multiline comment" << std::endl;
        }
        else
        {
                std::cerr << "File `program.txt` not open..." << std::endl;
                std::exit(2);
        }
        fin.close();
        fout_errors.close();
        fout_tokens.close();
}

size_t Translator::errorHandler(std::string str)
{
        std::string begin_of_operation;
        for (size_t i = 0; i < str.size(); i++)
        {
                begin_of_operation = str[i];
                if (Find(operations, begin_of_operation) != -1)
                        return i - 1;
        }
        return str.size();
}
```

data.h

```
#pragma once
#include <string>

struct Variable
```

```
{
    std::string name;
    std::string type;
    std::string value;
};

enum class TypesOfTables
{
    VARIABLE_TABEL,
    CONSTATN_TABEL,
    KEYWORDS_TABLE,
    OPERATIONS_TABEL,
    SEPARATORS_TABEL
};

enum class SymbolType
{
    FORWARD_SLASH,
    LETTER_OR_UNDERSCORE, // буква или нижнее подчеркивание
    NUMBER,
    OPERATION,
    SEPARATOR,
    DEFAULT
};
```