Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра ТПИ

ЯЗЫКИ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ

Лабораторная работа № 2

Разработка и реализация блока лексического анализа (сканер)

Факультет: ПМИ

Преподаватель: Еланцева И.Л.

Группа: ПМ-81

Студенты: Ефремов А. А.,
Ртищева К. С.

Бригада: 1

Вариант: 1

Новосибирск
2021

1. Цель работы

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

2. Условие задачи

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. Построение детерминированного конечного автомата

	текущий символ	0	1	2	3	4
предыдущее слово		разделитель (" ","\n","\t")	оператор	буква	цифра	не в алфавите
0	пусто	(0)	(1)	(2)	(3)	**ошибка**
1	оператор	новый оператор(0)	если == или != то новый оператор(0), иначе если // то(4) иначе если /* то(5) иначе **ошибка**	новый оператор(2)	новый оператор(3)	**ошибка**
2	слово	если КС то новое КС(0) иначе новый ид(0)	если КС то новое КС(1) иначе новый ид(1)	(2)	(2)	**ошибка**
3	константа	новая константа(0)	новая константа(1)	**ошибка**	(3)	**ошибка**
4	однострочный комментарий	если "\n" то (0) иначе (4)	(4)	(4)	(4)	(4)
5	многострочный комментарий	(5)	если */ то (0) иначе (5)	(5)	(5)	(5)

4. Текст программы

Файл “VarTableRow.h”

```
#pragma once
#include <string>

using namespace std;

class VarTableRow
{
public:
   int value;
   string name;
   bool is_set;

   VarTableRow() {};

   VarTableRow(const int& t_value, const string& t_name, const bool t_is_set) :
      value(t_value), name(t_name), is_set(t_is_set) {};

   bool operator == (VarTableRow lhs)
   {
      return value == lhs.value  && name == lhs.name && is_set == lhs.is_set;
   }
};
```

Файл “VarTable.h”

```
#pragma once
#include <vector>
#include <string>
#include <iomanip>
#include "VarTableRow.h"

using namespace std;

class VarTable
{
public:
   vector<VarTableRow> table;

   // Создание пустой таблицы
   VarTable()
   {
      table = vector<VarTableRow>(0);
   }

   // Функция поиска номера строки таблицы по идентификатору,
   // возращает -1 в случае отсутствия строки с таким идентификатором в таблице
   int GetRowIndex(const VarTableRow& t_row)
   {
      for(size_t i = 0; i < table.size(); i++)
         if(table[i] == t_row)
            return i;

      return -1;
   }
```

```
    // Функция добавления строки в таблицу, если такого вхождения нет,
    // возвращает номер строки
    int AddRow(const VarTableRow& t_row)
    {
       int index = GetRowIndex(t_row);
       if(index == -1)
       {
          table.push_back(t_row);
          return table.size() - 1;
       }
       else
          return index;
    }

    // Функция, возвращающая
    VarTableRow GetRow(const int& t_index)
    {
       if(t_index < table.size())
          return table[t_index];
       else
          printf_s("Error!");
    }

    void Output(const string& OUT_FILE)
    {
       ofstream fout(OUT_FILE);
       fout << "i   value   name  is set" << endl;
       for (size_t i = 0; i < table.size(); i++)
       {
          fout << i << setw(5) << table[i].value;
          fout << setw(8) << table[i].name;
          fout << setw(5) << table[i].is_set;
          fout << endl;
       }
       fout.close();
    }

    // Получение значений атрибутов
    int    GetValue(const int& t_index) { return table[t_index].value; }
    string GetName(const int& t_index)  { return table[t_index].name; }
    bool   GetIsSet(const int& t_index) { return table[t_index].is_set; }

    // Установление значений атрибутов
    void SetValue(const int& t_index, const int& t_value) { table[t_index].value = t_value; }
    void SetName(const int& t_index, const string& t_name)  { table[t_index].name = t_name; }
    void SetIsSet(const int& t_index, const bool t_is_set){table[t_index].is_set = t_is_set;}

};
```

Файл “ConstTableRow.h”

```
#pragma once
#include <string>

using namespace std;

class ConstTableRow
{
public:
   string name;
   ConstTableRow() {};

   ConstTableRow(const string& t_name) :
      name(t_name) {};

   bool operator == (const ConstTableRow& lhs)
   {
      return name == lhs.name;
   }

};
```

Файл “ConstTable.h”

```
#pragma once
#pragma once
#include <vector>
#include <string>
#include <iomanip>
#include "ConstTableRow.h"

using namespace std;

class ConstTable
{
public:
   vector<ConstTableRow> table;

   // Создание пустой таблицы
   ConstTable()
   {
      table = vector<ConstTableRow>(0);
   }

   // Создание таблицы с ключевыми словами
   void FillKeyWords()
   {
      const int k = 5;
      table.resize(k);
      string key_words[k] = {"if", "else", "main", "return", "int"};
      for(size_t i = 0; i < k; i++)
         table[i] = ConstTableRow(key_words[i]);
   }

   // Создание таблицы с операторами
   void FillOperators()
   {
      const int k = 14;
      table.resize(k);
      string operators[k] = { "=", "+", "-", "*" , "/", "==", "!=", "<", "(", ")", "{", "}",
         ",", ";"};

      for(size_t i = 0; i < k; i++)
```

```
      table[i] = ConstTableRow(operators[i]);
}

// Создание таблицы со всеми символами алфавита языка
void FillAplhabet()
{
   const int k = 15;
   table.resize(k + 26 + 26 + 10);
   string operators[k] = { "=", "+", "-", "*", "/", "=", "!", "<", "(", ")", "{", "}",
      ",", ";", "_" };

   for(size_t i = 0; i < k; i++)
      table[i] = ConstTableRow(operators[i]);

   for(int i = 0; i < 26; i++)
      table[i + k] = ConstTableRow(string(1, (char)('a' + i)));

   for(int i = 0; i < 26; i++)
      table[i + k + 26] = ConstTableRow(string(1, (char)('A' + i)));

   for (int i = 0; i < 10; i++)
      table[i + k + 26 + 26] = ConstTableRow(string(1, (char)('0' + i)));
}

// Создание таблицы со всеми символами алфавита языка с которых
// могут начинаться идентификаторы
void FillIdentName()
{
   table.resize(1 + 26 + 26);

   for (int i = 0; i < 26; i++)
      table[i] = ConstTableRow(string(1, (char)('a' + i)));

   for (int i = 0; i < 26; i++)
      table[i + 26] = ConstTableRow(string(1, (char)('A' + i)));

   table[52] = ConstTableRow(string(1, (char)('_')));
}

// Создание таблицы со всеми цифрами алфавита языка
void FillNumbers()
{
   table.resize(10);

   for(int i = 0; i < 10; i++)
      table[i] = ConstTableRow(string(1, (char)('0' + i)));
}
// Функция поиска номера строки таблицы по идентификатору,
// возращает -1 в случае отсутствия строки с таким идентификатором в таблице
int GetRowIndex(const ConstTableRow& t_row)
{
   for(size_t i = 0; i < table.size(); i++)
      if(table[i] == t_row)
         return i;

   return -1;
}

void Output(const string& OUT_FILE)
{
   ofstream fout(OUT_FILE);
   fout << "i    name" << endl;
   for (size_t i = 0; i < table.size(); i++)
   {
```

```
            fout << setw(2) << i;
            fout << setw(10) << table[i].name;
            fout << endl;
        }
        fout.close();
    }
};
```

Файл “LexicalAnalyzer.h”

```
#pragma once
#include <iostream>
#include <string>
#include <fstream>
#include "VarTable.h"
#include "ConstTable.h"

enum class WordType
{
    Blank,
    Operator,
    Word,
    Constant,
    KeyWord
};

enum class SymbolType
{
    Separator,
    Operator,
    Letter,
    Number,
    Error
};

class LexicalAnalyzer
{
public:
    ConstTable alphabet, key_words, operators, numbers, ident_name;
    VarTable var_table, const_table;

    LexicalAnalyzer()
    {
        alphabet.FillAplhabet();
        key_words.FillKeyWords();
        operators.FillOperators();
        numbers.FillNumbers();
        ident_name.FillIdentName();
    }

    // Определение типа символа и получение его индекса
    // в соответствующей таблице
    SymbolType GetSymbolType(const string& s, int& place)
    {
        // Разделитель
        if(s == " " || s == "\n" || s == "\t")
            return SymbolType::Separator;

        // Ошибка
        place = alphabet.GetRowIndex(ConstTableRow(s));
        if(place == -1)
            return SymbolType::Error;
```

```
    // Оператор
    place = operators.GetRowIndex(ConstTableRow(s));
    if(place != -1)
        return SymbolType::Operator;

    // Символ с которого может начинаться имя переменной
    place = ident_name.GetRowIndex(ConstTableRow(s));
    if(place != -1)
        return SymbolType::Letter;

    // Цифра
    place = numbers.GetRowIndex(ConstTableRow(s));
    if(place != -1)
        return SymbolType::Number;
}

// Печать всех таблиц
void PrintAllTables(const string& directory)
{
    alphabet.Output(directory + "/aplhabet.txt");
    key_words.Output(directory + "/keyWords.txt");
    operators.Output(directory + "/operators.txt");
    numbers.Output(directory + "/numbers.txt");
    ident_name.Output(directory + "/ident_name.txt");
    const_table.Output(directory + "/const.txt");
    var_table.Output(directory + "/var.txt");
}

void MakeTokens(const string& in_filename, const string& out_filename)
{
    ifstream fin(in_filename);
    ofstream fout(out_filename);

    int symbol_n = 0, line_n = 1;
    char c;
    string word = "", symbol;

    // Тип передыдущего слова
    WordType word_type = WordType::Blank;

    // Тип символа
    SymbolType symbol_type;

    // Место символа в соответствующей таблице
    int place = 0;

    // Место предыдущего символа в соответствующей таблице
    int prev_place = 0;

    // Если комментирование оператором */
    bool is_op_comment = false;
    string prev_symbol;

    // Если комментирование оператором //
    bool is_line_comment = false;

    while(fin.get(c))
    {
        symbol = c;
        symbol_n++;
        symbol_type = GetSymbolType(symbol, place);
```

```
if(symbol_type == SymbolType::Error)
{
   cout << "Error at line " << line_n << " pos " << symbol_n;
   cout << ": Invalid symbol! ";
   exit(2);
}

if(is_op_comment)
{
   string temp_s = prev_symbol + symbol;
   if(temp_s == "*/")
      is_op_comment = false;
   else
      prev_symbol = symbol;
}
else if(is_line_comment)
{
   if(symbol == "\n")
      is_line_comment = false;
}
else
   switch(word_type)
   {
      // Слово не задано
      case WordType::Blank:
      {
         switch(symbol_type)
         {
            case SymbolType::Separator:
            {
               if(symbol == "\n")
               {
                  symbol_n = 0;
                  line_n++;
                  fout << endl;
               }
               word_type = WordType::Blank;
               break;
            }
            case SymbolType::Operator:
            {
               prev_place = place;
               word = symbol;
               word_type = WordType::Operator;
               break;
            }
            case SymbolType::Letter:
            {
               word = symbol;
               word_type = WordType::Word;
               break;
            }
            case SymbolType::Number:
            {
               word = symbol;
               word_type = WordType::Constant;
               break;
            }
         }
         break;
      }
      // Слово - оператор
      case WordType::Operator:
      {
```

```
switch(symbol_type)
{
   // Символ - разделитель
   case SymbolType::Separator:
   {
      fout << "(20," << place << ")";
      word_type = WordType::Blank;
      word = "";

      if(symbol == "\n")
      {
         symbol_n = 0;
         fout << endl;
         line_n++;
      }
      break;
   }
   // Символ - оператор
   case SymbolType::Operator:
   {
      string temp_op = word + symbol;

      if(temp_op == "/*")
      {
         word_type = WordType::Blank;
         word = "";
         is_op_comment = true;
         break;
      }
      if(temp_op == "//")
      {
         word_type = WordType::Blank;
         word = "";
         is_line_comment = true;
         break;
      }

      int temp_place = operators.GetRowIndex(ConstTableRow(temp_op));

      // Если оператор - "==" или "!="
      if(temp_place != -1)
      {
         fout << "(20," << temp_place << ")";
         word_type = WordType::Blank;
         word = "";
      }
      else if(temp_op == "()")
      {

         fout << "(20," << prev_place << ")";
         word_type = WordType::Operator;
         word = symbol;
         prev_place = place;
      }
      else
      {
         cout << "Error at line " << line_n << " pos " << symbol_n;
         cout << ": Invalid operator! ";
         exit(2);
         break;
      }
      break;
   }
```

```
            // Символ - буква
            case SymbolType::Letter:
            {
                fout << "(20," << place << ")";
                word_type = WordType::Word;
                word = symbol;
                break;
            }
            // Символ - цифра
            case SymbolType::Number:
            {
                fout << "(20," << place << ")";
                word_type = WordType::Constant;
                word = symbol;
                break;
            }
        }
        break;
    }
    // Слово - слово
    case WordType::Word:
    {
        switch(symbol_type)
        {
            // Символ - разделитель
            case SymbolType::Separator:
            {
                int kw_place = key_words.GetRowIndex(ConstTableRow(word));

                // Если слово - ключеваое слово
                if(kw_place != -1)
                    fout << "(10," << kw_place << ")";
                else
                    fout << "(30," << var_table.AddRow(VarTableRow(0, word, false))
                        << ")";

                word_type = WordType::Blank;
                word = "";

                if(symbol == "\n")
                {
                    symbol_n = 0;
                    line_n++;
                    fout << endl;
                }
                break;
            }
            // Символ - оператор
            case SymbolType::Operator:
            {
                int kw_place = key_words.GetRowIndex(ConstTableRow(word));

                // Если слово - ключеваое слово
                if(kw_place != -1)
                    fout << "(10," << kw_place << ")";
                else
                    fout << "(30," << var_table.AddRow(VarTableRow(0, word, false))
                        << ")";

                word_type = WordType::Operator;
                prev_place = place;
                word = symbol;
                break;
            }
```

```
            // Символ - буква
            case SymbolType::Letter:
            {
               word += symbol;
               break;
            }
            // Символ - цифра
            case SymbolType::Number:
            {
               word += symbol;
               break;
            }
         }
         break;
      }
      // Слово - константа
      case WordType::Constant:
      {
         switch(symbol_type)
         {
            // Символ - разделитель
            case SymbolType::Separator:
            {
               fout << "(40," << const_table.AddRow(VarTableRow(0, word, false))
                  << ")";
               word_type = WordType::Blank;
               word = "";

               if(symbol == "\n")
               {
                  symbol_n = 0;
                  line_n++;
                  fout << endl;
               }
               break;
            }
            // Символ - оператор
            case SymbolType::Operator:
            {
               fout << "(40," << const_table.AddRow(VarTableRow(0, word, false))
                  << ")";
               word_type = WordType::Operator;
               word = symbol;
               prev_place = place;
               break;
            }
            // Символ - буква
            case SymbolType::Letter:
            {
               cout << "Error at line " << line_n << " pos " << symbol_n;
               cout << ": Invalid constant (identifier)! ";
               exit(2);
               break;
            }
            // Символ - цифра
            case SymbolType::Number:
            {
               word += symbol;
               break;
            }
         }
         break;
      }
   }
```

```
        }

        if(is_op_comment)
        {
            cout << "Unclosed comment!";
            exit(2);
        }

        fout.close();
        fin.close();
    }
};
```

Файл “main.cpp”

```
#include "LexicalAnalyzer.h"

int main()
{
    LexicalAnalyzer la = LexicalAnalyzer();

    la.MakeTokens("test_1.txt", "tokens.txt");
    la.PrintAllTables("tables");
}
```

5. Тестовые примеры

№	Входные данные	Выходные данные			Назначение

1	int main()
	{
	int a = 0;
	a += 10;
	}
		Error at line 4 pos 7: Invalid operator!			Недопустимый оператор

2	int main()
	{
	int # = 10;
	}
		Error at line 3 pos 8: Invalid symbol!			Недопустимый символ

3	int main()
	{
	int 23a = 10;
	}
		Error at line 3 pos 10: Invalid constant (identifier)!			Недопустимая константа (идентификатор)

4	int main()
	{
	int a = 90;
	/* int asd
	}
		Unclosed comment!			Незакрытый комментарий

5	int main()
	{
	int a = 90;
	/* int asd
	*/
	int b = 1;
	// int asd
	return 0;
	}
		“tokens.txt”
		
		(10,4)(10,2)(20,8)(20,9)
		(20,10)
		(10,4)(30,0)(20,0)(40,0)(20,13)
		
		(10,4)(30,1)(20,0)(40,1)(20,13)
		(10,3)(40,2)(20,13)
		(20,11)
			“const.txt”
			
			i value name is set
			0 0 90 0
			1 0 1 0
			2 0 0 0
				“var.txt”
				
				i value name is set
				0 0 a 0
				1 0 b 0
					Лексически правильных код

Постоянные таблицы:

“alphabet.txt”

i	name	i	name
0	=	51	K
1	+	52	L
2	-	53	M
3	*	54	N
4	/	55	O
5	=	56	P
6	!	57	Q
7	<	58	R
8	(	59	S
9	)	60	T
10	{	61	U
11	}	62	V
12	,	63	W
13	;	64	X
14	_	65	Y
15	a	66	Z
16	b	67	0
17	c	68	1
18	d	69	2
19	e	70	3
20	f	71	4
21	g	72	5
22	h	73	6
23	i	74	7
24	j	75	8
25	k	76	9
26	l		
27	m		
28	n		
29	o		
30	p		
31	q		
32	r		
33	s		
34	t		
35	u		
36	v		
37	w		
38	x		
39	y		
40	z		
41	A		
42	B		
43	C		
44	D		
45	E		
46	F		
47	G		
48	H		
49	I		
50	J		

“ident_name.txt”

i	name
0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	A
27	B
28	C
29	D
30	E
31	F
32	G
33	H
34	I
35	J
36	K
37	L
38	M
39	N
40	O
41	P
42	Q
43	R
44	S
45	T
46	U
47	V
48	W
49	X
50	Y
51	Z
52	_

“keyWords.txt”

i	name
0	if
1	else
2	main
3	return
4	int

“numbers.txt”

i	name
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

“operators.txt”

i	name
0	=
1	+
2	-
3	*
4	/
5	==
6	!=
7	<
8	(
9	)
10	{
11	}
12	,
13	;
14	/*
15	*/