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.

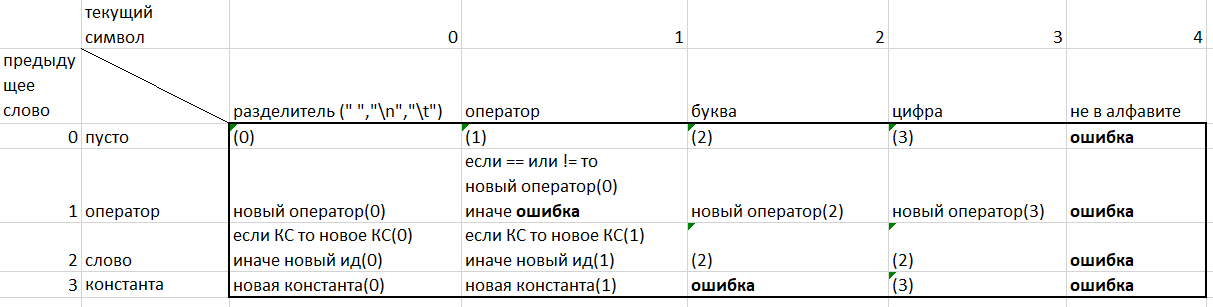
1. **Условие задачи**

Подмножество языка С++ 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 С++ 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.

1. **Построение детерминированного конечного автомата**



1. **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");

}

1. **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” “ident\_name.txt”**

|  |  |  |  |
| --- | --- | --- | --- |
| **i name**  **0 =**  **1 +**  **2 -**  **3 \***  **4 /**  **5 =**  **6 !**  **7 <**  **8 (**  **9 )**  **10 {**  **11 }**  **12 ,**  **13 ;**  **14 \_**  **15 a**  **16 b**  **17 c**  **18 d**  **19 e**  **20 f**  **21 g**  **22 h**  **23 i**  **24 j**  **25 k**  **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** | **i name**  **51 K**  **52 L**  **53 M**  **54 N**  **55 O**  **56 P**  **57 Q**  **58 R**  **59 S**  **60 T**  **61 U**  **62 V**  **63 W**  **64 X**  **65 Y**  **66 Z**  **67 0**  **68 1**  **69 2**  **70 3**  **71 4**  **72 5**  **73 6**  **74 7**  **75 8**  **76 9** | **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 \*/** |