Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Языки программирования и методы трансляции

Лабораторная работа №2

Факультет прикладной математики и информатики

Группа ПМ-01

Студенты Александров М.Е.

Жигалов П.С.

Преподаватели Еланцева И.Л.

Полетаева И.А.

Вариант 7

Новосибирск

2013

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 С++ включает:

* данные типа **int, float, массивы** из элементов указанных типов;
* инструкции описания переменных;
* операторы присваивания 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.

3. Структура входных и выходных данных

Входные данные представляют собой файлы: содержащего исходный код, а также файлы, содержащие допустимые буквы, числа, операции, ключевые слова и разделители. Результатом работы программы являются список токенов (номер таблицы и именной адрес) и ошибок

4. Детерминированный конечный автомат

S

W

DER

KW

OP

OP2

ID

CT

ERR

Z

DEC

слово

Описание состояний

**S** – начальное состояние

**W** – Слово

**KW** – Ключевое слово

**DER** – разделитель

**OP** – операция

OP2– оператор

**CT** – константа

**ID** – Идентификатор

**DEC** – разделяемый идентификатор

**ERR** – ошибка

**Z** – конечное состояние

5. Алгоритм разбора

1) считать весь текст

2) Очистить от комментариев

3) Разделить текст на элементы по пробелам

4) обработать очередное выделенное слово, если ключевое слово – перехожу к шагу 5, если операция – к шагу 5, если разделитель к шагу 5, если оператор – то шаг 5, если идентификатор то шаг 6. Если слова закончились – к шагу 10

5) сформировать токен, удалить слово из строки, вернуться к шагу 4.

6) проверить, есть ли вхождения оператора, или знака операции, если есть – окружить пробелами, вернуться к шагу 4, иначе 7

7) проверяем, является ли слово константой – шаг 8, если нет – шаг 9

8) формируем токен, формируем лексему и заносим в таблицу констант, возвращаемся к шагу 4

9) проверяем правильность идентификатора , если удовлетворительно – формируем токен, проверяем есть ли идентификатор в таблице идентификаторов, если нет – то формируем лексему и заносим в таблицу идентификаторов. В случаи некорректного имени идентификатора – выводится ошибкаВозвращаемся к шагу 4

10) конец

6. Тесты

6.1 Верный исходный код с различными видами структур:

Исходный код

int main()  
{  
int a = - 1;  
 int a = 0;  
 int b = a;  
 //c=wg  
 /\*sdf  
 qwe\*/  
 float k = {4,  
 7,8  
 }  
 a=b++-4;  
 float b[4];  
 int a [4]   
 return 0;  
}

Файл токенов

0 int

0 main

2 (

2 )

2 {

0 int

10 a

1 =

1 -

11 1

2 ;

0 int

10 a

1 =

11 0

2 ;

0 int

10 b

1 =

10 a

2 ;

0 float

10 k

1 =

2 {

11 4

2 ,

11 7

2 ,

11 8

2 }

10 a

1 =

10 b

3 ++

1 -

11 4

2 ;

0 float

10 b

2 [

11 4

2 ]

2 ;

0 int

10 a

2 [

11 4

2 ]

0 return

11 0

2 ;

2 }

6.3 проблемный код

int mai^n()  
{  
int a = - 1;  
 int a = 0;  
 int b = a;  
 int c = 12.3.4;  
 //c=wg  
 /\*sdf\*/  
 a=b++-4;  
 float% b[4];  
 #  
 int a [4[  
 return 0;  
}

Токены:

0 int

2 (

2 )

2 {

0 int

10 a

1 =

1 -

11 1

2 ;

0 int

10 a

1 =

11 0

2 ;

0 int

10 b

1 =

10 a

2 ;

0 int

10 c

1 =

2 ;

10 a

1 =

10 b

3 ++

1 -

11 4

2 ;

10 b

2 [

Ошибки

Error, incorrect identifier mai^n in line 1

Error, incorrect identifier 12.3.4 in line 6

Error, incorrect identifier float% in line 10

Error, incorrect identifier 4 in line 6

***7. Код программы***

Table.cs

using System.Collections.Generic;  
using System.IO;  
using System;  
namespace YP2  
{  
 class Table  
 {  
 public Table()  
 {  
 Create();  
 }  
 public VariableTable ind = new VariableTable(10);  
 public VariableTable constant = new VariableTable(11);  
 public ConstTable keywords = new ConstTable(0);  
 public ConstTable operators = new ConstTable(1);  
 public ConstTable delimiters = new ConstTable(2);  
 public ConstTable latters = new ConstTable(4);  
 public ConstTable nummber = new ConstTable(5);  
 public ConstTable oper = new ConstTable(3);  
 private void Create()  
 {  
 string workingDirectory = Environment.CurrentDirectory;  
 string projectDirectory = Directory.GetParent(workingDirectory).Parent.FullName;  
 projectDirectory = projectDirectory.Substring(0, projectDirectory.Length - 4);  
 keywords.ReadFrom(projectDirectory+"\\keywords.txt");  
 operators.ReadFrom(projectDirectory+"\\operators.txt");  
 oper.ReadFrom(projectDirectory+"\\oper.txt");  
 delimiters.ReadFrom(projectDirectory+"\\delimiters.txt");  
 latters.ReadFrom(projectDirectory+"\\letters.txt");  
 nummber.ReadFrom(projectDirectory+"\\num.txt");  
 }  
 public int GetNum(string word)  
 {  
 if (keywords.Contains(word))  
 return 0;  
 else if (operators.Contains(word))  
 return 1;  
 else if (delimiters.Contains(word))  
 return 2;  
 else if (oper.Contains(word))  
 return 3;  
 else if (constant.ContainsKey(word))  
 return 11;  
 else  
 return 10;  
 }  
 public List<string> get(string s)  
 {  
 int num = GetNum(s);  
 List<string> l = new List<string>();  
 l.Add(num.ToString());  
 l.Add(s);  
 return l;  
 }  
  
 public void set(string word)  
 **{** Lexem first = new Lexem(word);  
 ind.Add(word, first);  
 **}** public void PrinVTables()  
 {  
 Console.WriteLine("Const table:");  
 foreach (var var in constant)  
 {  
 Console.WriteLine(var.ToString());  
 }  
   
 Console.WriteLine("Identifiers table:");  
 foreach (var var in ind)  
 {  
 Console.WriteLine(var.ToString());  
 }  
   
   
 }  
  
 }  
}

Lex.cs

using System.Collections.Generic;  
using System.IO;  
using System;  
using System.Linq;  
using System.Text.RegularExpressions;  
  
namespace YP2  
{  
 class Lex  
 {  
 Table t;  
 private string[] literals;  
 public List<string> errors = new List<string>();  
  
 public Lex(Table table)  
 {  
 t = table;  
 }  
 public string[] Read()  
 {  
 string workingDirectory = Environment.CurrentDirectory;  
 string projectDirectory = Directory.GetParent(workingDirectory).Parent.FullName;  
 projectDirectory = projectDirectory.Substring(0, projectDirectory.Length - 4);  
  
   
 literals = File.ReadAllLines(projectDirectory+"\\input.txt");  
 return literals;  
 }  
 public List<string> decoment(string[] s) //удаление пробелов  
 {  
 string str, str\_1 = "";  
 int index, index\_1, index\_2;  
 str = String.Join("\n", s);  
 List<string> new\_s = new List<string>();  
 string[] str\_old = str.Split('\n');  
 foreach (string line in str\_old)  
 {  
 index = line.IndexOf("//");  
 index\_1 = line.IndexOf("/\*");  
 if ((index < index\_1 && index\_1 != -1 && index != -1) || index != -1)  
 str\_1 += line.Remove(index);  
 else  
 str\_1 += line + "\n";  
 }  
 str = str\_1;  
 index\_1 = str.IndexOf("/\*");  
 index\_2 = str.IndexOf("\*/");  
 while (index\_1 != -1 && index\_2 != -1)  
 {  
 if (index\_2 > index\_1)  
 str = str.Remove(index\_1, index\_2 - index\_1 + 2);  
 //else  
 // Console.WriteLine("error, incorrect comment");  
 index\_1 = str.IndexOf("/\*");  
 index\_2 = str.IndexOf("\*/");  
 }  
 new\_s.Add(str);  
 return new\_s;  
 }   
 public bool string\_analize(string str\_old)  
 {  
 string ex = @"\s+", str = str\_old;  
 string[] s = Regex.Split(str, ex);  
 string[] q = { "" };  
 s = s.Except(q).ToArray();  
 if (s.Length > 0)  
 {  
 string word;  
 word = s[0];  
 int num = t.GetNum(word);  
 switch (num)  
 {  
 case 0: //Ключевое слово  
 {  
 List<string> l = t.get(word);  
 Console.WriteLine(l[0] + " " + l[1]);  
 str = str.Remove(0, str.IndexOf(word) + word.Length);  
 return string\_analize(str);  
 }  
 break;  
 case 1: //Операция  
 {  
 List<string> l = t.get(word);  
 Console.WriteLine(l[0] + " " + l[1]);  
 str = str.Remove(0, str.IndexOf(word) + 1);  
 return string\_analize(str);  
 }  
 break;  
 case 2: //Разделитель  
 {  
 List<string> l = t.get(word);  
 Console.WriteLine(l[0] + " " + l[1]);  
 str = str.Remove(0, str.IndexOf(word) + word.Length);  
 return string\_analize(str);  
 }  
 break;  
 case 3: //Оператор  
 {  
 {  
 List<string> l = t.get(word);  
 Console.WriteLine(l[0] + " " + l[1]);  
 str = str.Remove(0, str.IndexOf(word) + word.Length);  
 return string\_analize(str);  
 }  
 }  
 break;  
 case 10: //Индентификатор  
 {  
 for (int i = 0; i < word.Length - 1; i++)  
 {  
 string a = word[i].ToString(), b = word[i + 1].ToString();  
 if (t.oper.Contains(a + b))  
 {  
 str = str.Insert(str.IndexOf(word[i]), " ");  
 str = str.Insert(str.IndexOf(word[i + 2]), " ");  
 return string\_analize(str);  
 }  
 }  
 for (int i = 0; i < word.Length; i++)  
 {  
 string a = word[i].ToString();  
 if (t.operators.Contains(a) || t.delimiters.Contains(a))  
 {  
 str = str.Insert(str.IndexOf(word[i]), " ");  
 str = str.Insert(str.IndexOf(word[i])+1, " ");  
 return string\_analize(str);  
 }  
 }  
 int n;  
 if(Int32.TryParse(word, out n))  
 {  
 if(!t.constant.ContainsKey(word))  
 {  
 Lexem first = new Lexem(word);  
 t.constant.Add(word, first);  
 }  
 List<string> l = t.get(word);  
 Console.WriteLine(l[0] + " " + l[1]);  
 str = str.Remove(0, str.IndexOf(word) + word.Length);  
 return string\_analize(str);  
 }  
 bool f = true;  
 foreach (char c in word)  
 f = f && (t.latters.Contains(c.ToString())) || t.nummber.Contains(c.ToString()) && t.latters.Contains(word[0].ToString());  
 if (!f)  
 {  
 int i = 0;  
 for (i = 0; i < literals.Length; i++)  
 **{** if (literals[i].IndexOf(word) != -1)  
 break;  
 **}** errors.Add(String.Format("Error, incorrect identifier {0} in line {1}", word, i+1));  
 str = str.Remove(0, str.IndexOf(word) + word.Length);  
 return string\_analize(str);  
 }  
 else  
 {  
 if (!t.ind.ContainsKey(word))  
 t.set(word);  
 List<string> l = t.get(word);  
 Console.WriteLine(l[0] + " " + l[1]);  
 str = str.Remove(0, str.IndexOf(word) + word.Length);  
 return string\_analize(str);  
 }  
 }  
 break;  
 default: //Что-то плохое  
 {  
 int i = 0;  
 for (i = 0; i < literals.Length; i++)  
 {  
 if (literals[i].IndexOf(word) != -1)  
 break;  
 }  
 errors.Add(String.Format("Error, incorrect identifier {0} in line {1}", word, i+1));  
 return false;  
 }  
 break;  
 }  
 }  
 return true;  
 }  
  
 public void Out\_errors()  
 {  
 foreach(string w in errors)  
 Console.WriteLine(w);  
 }  
 }  
  
}

program.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Data;  
using System.Data.SqlTypes;  
using System.IO;  
using System.Linq;  
  
namespace YP2  
{  
 enum LexType  
 **{** undef,  
 integer,  
 flt  
 **}** internal class Program  
 {  
 public static void Main(string[] args)  
 {  
 Table table = new Table();  
 Lex l = new Lex(table);  
 //foreach (string s in l.decoment(l.Read()))  
 // l.string\_analize(s);  
 string s = l.decoment(l.Read())[0];  
 l.string\_analize(s);  
 l.Out\_errors();  
 table.PrinVTables();  
 }  
 }  
}

Lexem.cs

using System.Collections.Generic;  
using System.Reflection.Metadata.Ecma335;  
  
namespace YP2  
{  
 enum ValueType  
 {  
 undef,  
 integer,  
 flt  
 }  
 class Lexem  
 {  
 private string name;  
 private int dimension; //размерность: 1 - для переменных и констант  
 private List<bool> isInit = new List<bool>(); //определено ли значение  
  
 public Lexem(string name)  
 {  
 int k;  
 float t;  
 this.name = name;  
 if (int.TryParse(name, out k))  
 {  
 isInit.Add(true);  
 return;  
 }  
   
 if (float.TryParse(name,out t))  
 {  
 isInit.Add(true);  
 }  
   
 }  
  
 public string Name  
 {  
 get => name;  
 }  
  
 public int Dimension  
 {  
 get => isInit.Count;  
 }  
  
 public List<bool> IsInit  
 {  
 get => isInit;  
 set => isInit = value;  
 }  
  
 public ValueType Type1  
 {  
 get => Type;  
 set => Type = value;  
 }  
 public ValueType Type = (ValueType)0;  
  
 public override string ToString()  
 {  
 string ToReturn = $"name = {name}, type = {Type}, dimension = {Dimension}| initialized:";  
 for (int i = 0; i < isInit.Count; i++)  
 {  
 if (isInit[i])  
 {  
 ToReturn += $" {isInit[i]} ,";  
 }  
 }  
  
 if (Dimension == 0)  
 {  
 ToReturn+= false;  
 }  
  
 return ToReturn;  
 }  
  
 public override bool Equals(object obj)  
 {  
 Lexem p = (Lexem)obj;  
 return this.name == p.name;  
 }  
 };  
}

ConstTable.cs

using System.Collections.Generic;  
using System.IO;  
  
namespace YP2  
{  
 class ConstTable : SortedSet<string>  
 {  
 private int num;  
 public ConstTable(int num) { this.num = num; }  
 public bool ReadFrom(string fileName)  
 {  
 string[] literals = File.ReadAllLines(fileName);  
 if (literals.Length == 0)  
 {  
 return false;  
 }  
 foreach (var item in literals)  
 {  
 Add(item);  
 }  
 return true;  
 }  
  
 }  
}

VariableTable.cs

using System.Collections.Generic;  
using System.ComponentModel;  
  
namespace YP2  
{  
  
 class VariableTable : Dictionary**<**string, Lexem**>** {  
 private int num;  
 public VariableTable(int num) { this.num = num; }  
 }  
}