МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РФ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра вычислительной техники



Лабораторная работа №2

По дисциплине: «Теория формальных языков и компиляторов»

На тему: «Лексика языков программирования. Конечные автоматы без памяти для обнаружения слов в тексте программы»

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| Студент гр. *АВТ-709*, *АВТФ* |  |
| *Андреева А.О.* | *Малявко Александр Антонович* |
| «\_\_\_» \_\_\_\_\_\_ 20\_\_г. | «\_\_\_» \_\_\_\_\_\_ 20\_\_г. |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| (подпись) | (подпись) |

Новосибирск

2020

**Цель работы:**

Изучение конечных автоматов (КА) без памяти, способов определения КА – канонического, графового и табличного, методов построения недетерминированного КА по системе регулярных выражений, методов эквивалентных преобразований недетерминированных КА в оптимальные полностью определенные КА – лексические акцепторы.

**Задание:**

* 1. Используя пакет ВебТрансЛаб, освоить:
     + создание лексических правил на языке регулярных выражений (РВ);
     + использование операций «+, \*, ?, конкатенации и выбора» языка РВ;
     + преобразование системы РВ в одноавтоматный лексический акцептор;
     + добавление правил и действий в систему РВ для построения мультиавтоматного лексического акцептора;
  2. Разработать фрагмент системы регулярных выражений для всех (или выбранной самостоятельно части) групп слов языка, определенного заданием на курсовую работу. Построить по этому фрагменту:
     + программный модуль, управляемый графом состояний и переходов;
     + программный модуль, управляемый таблично;
  3. Изучить структуру программных модулей, построенных ВебТрансЛабом, изучить алгоритмы работы лексического акцептора для графового и табличного способов реализации КА, сравнить реализации конечных автоматов, управляемых различными способами, между собой;
  4. Изучить реализацию вызова действий, определенных в лексических правилах и алгоритм работы формирователя лексем.
  5. Проверить функционирование конечных автоматов, построенных ВебТрансЛабом (подготовить тестовый пример, запустить каждый автомат на выполнение, протрассировать работу лексического акцептора в графовой и табличной реализации, убедиться в работоспособности автоматов, в противном случае – доработать систему РВ и добиться правильного функционирования лексического акцептора).
  6. Подготовить, сдать и защитить отчет к лабораторной работе.

**Ход работы:**

В работе была доработана лексика языка.

Тестовая программа предлагается к рассмотрению в Приложении 1, правила лексики разрабатываемого языка в Приложении 2.

**Основные понятия**

**Произвольное выражение** – это совокупность идентификаторов, знаков математических операций и круглых скобок подчиняемая математическим законам.

**Логическое выражение** – это выражение результат, которого можно интерпретировать как истину или ложь.

**Константа** – это число, символ или строка символов, которая в программе используется, как значение. Значение константы нельзя изменить.

**Оператор** – один выполняемый оператор (присваивания, условный оператор, оператор цикла, переключатель)

**Блок** – это набор идущих подряд операторов, обрамлённый фигурными скобками.

**Функция** – это фрагмент программного кода (подпрограмма), к которому можно обратиться из другого места программы. Пример:

*float p123r(float f104t)*

*{*

*return f104t \* f104t - f104t;*

*}*

**Возвращаемое значение** – это значение, которое вернет функция после выполнения. Тип возвращаемого выражения должен совпадать с типом функции.

**Файл**

В файле с исходным кодом программы располагаются команды препроцессора и функции. Причем только в таком порядке. Функции должны располагаться последовательно, то есть функция должна располагаться над вызвавшей ее функцией. Последней функцией должна быть функция *S\_tarter*.

Переменные могут быть описаны либо в командах препроцессора, либо внутри функций.

Функции *начала работы программы* может и не быть файле, но тогда этот файл для выполнения функций в нем должен быть подключен к файлу, где эта функция есть.

**Расположение**

Программа может располагаться в нескольких файлах.

Подключение дополнительных файлов происходит путем указания их имени специальной командой *#include <имя\_файла>.*

Включаемый файл должен находится в директории включаемых файлов, указанной в настройках проекта.

Подключаемые файлы также могут иметь директивы *#include.*

**Команды препроцессора**

Существуют команды препроцессора, которые будут выполнены до начала работы функции *начала работы порграммы*. Они начинаются с символа ‘#’, такие как *#define* или *#include.*

**Точка входа**

Точка входа — адрес в оперативной памяти, с которого начинается выполнение программы, адрес, по которому хранится первая команда программы.

В программе допускается только одна функция *входа*, иначе возникнет ошибка.

**Идентификаторы**

Идентификаторы всегда начинаются с большой буквы, далее идет символ «\_», далее возможно пустая последовательность английских букв и/или цифр.

Прописные английские буквы: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Строчные английские буквы: a b c d e f g h i j k l m n o p q r s t u v w x y z

Десятичные цифры: 0 1 2 3 4 5 6 7 8

Примеры: A\_, B\_, H\_abc, H\_abc123*;*

Компилятор обрабатывает буквы в разном регистре, как разные символы. То есть идентификаторы *A\_a* и *A\_A* будут восприняты компилятором как разные.

**Переменные**

Переменная може быть объявлена в любом месте функции и будет существовать только в ней.

Если переменная объявлена в функции *I\_n*, то в функции O\_ut ее не будет видно, и в функции *O\_ut* может быть объявлена переменная с таким же именем.

В команде препроцессора переменную можно объявить, как константную и доступную в любом месте программы.

Переменные должны быть объявлены и инициализированы до их использования.

**Константы**

Константы тоже имеют область видимости. В случае определения константы в теле функции/цикла область ее видимости - функция/цикл, в котором она была определена.

Константы:

целые по основаниям 2,8 и 10 (*2x01, 8x77, 123*);

вещественные (*0.123, 123.321*);

символьные (*'g', 'a'*).

**Функции**

Функция определяется следующим образом:  
*<тип> <идентификатор> ([<аргументы>]) {[<Оператор\_или\_блок>]}*

Тело функции – набор операторов между открывающей и закрывающей фигурной скобкой.

Если функция принимает аргументы, то тогда они должны быть объявлены в поле *<аргументы>.* Переданные в функцию аргументы будут существовать только в ней так же, как и созданные в ней переменные.

**Управляющие последовательности**

Символьные константы могут содержать "управляющие последовательности".

Управляющие последовательности — это комбинации символов, состоящие из разделительного символа и не графических символов.

Управляющая последовательность состоит из знака обратного деления (\) за которым следует буква или комбинация цифр.

Управляющие последовательности обычно используются для задания каких-либо действий (например, возврат каретки или сдвиг на табуляцию на экране или на принтере) или для литерального представления неграфических символов и символов, имеющих специальное значение (например, двойные цитатные скобки - ").

Пример управляющих последовательностей:

**Знаки пунктуации и операторы**

Знаки пунктуации и специальные символы из набора символов имеют самое разное предназначение, от организации текста программы до определения задач, которые будут выполнены компилятором или скомпилированной программой.

Существуют следующие знаки пунктуации и операторы:

; точка с запятой

, запятая

“ двойные кавычки

‘ одинарные кавычки

( левая круглая скобка

) правая круглая скобка

[ левая прямоугольная скобка

] правая прямоугольная скобка

{ левая фигурная скобка

} правая фигурная скобка

< меньше

> больше

<= меньше равно

>= больше равно  
!= не равно

/ знак деления  
/:= поделить равно

% знак остатка от деления

%:= остаток равно

+ плюс

+:= плюс равно

\* знак умножения

\*:= умножить равно

- минус

-:= минус равно

== равно

++ инкремент

- - декремент

**Комментарии**

Комментарии — это последовательность символов, которая рассматривается компилятором как разделительный символ, но игнорируется им.

Комментарии могут быть везде, где допустимо появление разделительных символов.

Пример:

*// Какой-либо комментарий*

**Ключевые слова**

Ключевые слова - это заранее определенные идентификаторы, которые имеют для компилятора специальное значение. Их можно использовать только так, как они определены. Имя элемента программы не может совпадать написанию с ключевым словом. Переопределить ключевые слова нельзя.

Существуют следующие ключевые слова:  
*int*

*int \_u*

*float*

*letter*

*void*

*return*

*cycle*

*select*

*case*

*break*

*end*

*at*

*do*

*else*

**Описание языка:**

На данный момент язык способен воспринимать программы, состоящие из одного файла. В таких файлах может быть любое количество функций, последней из которых должна быть функция main, с нее начнется выполнение программы.

Выражения разделяются символом;

1) Имеющиеся типы данных:

1. entire – целочисленный
2. symbol – строковый
3. real – вещественный
4. entire unsigned – беззнаковый тип
5. Массивы – задаются как ТипДанных[кол-во байт]

2) Имеющиеся математические операции:

1. + - сложение
2. - - вычитание
3. \* - умножение
4. / - деление
5. % - остаточное деление

3) Оператор присваивания обозначен как - :=

4) Для написания комментариев используется конструкция: //Comm//

5) Тип цикла: **exec**<оператор/блок>**with**<Переменная> **from** <Const> **to** <Const>**[step**<Const>**]**

6) Тип условного оператора:

**if (**<Логическое выражение>**)** <оператор/блок>**[ ifnot**<оператор/блок> **]** **fi**

7) Тип оператора переключателя:

**??**<Выражение> **{ ?**<Const> **:**<оператор/блок> **[gout;]**…   **[ ?~** **:**<Оператор/блок> **]** **}**

8) Для начала программного блока используются {}

9) Имеющиеся константы:

1) По основанию 10

2) По основанию 16

3) По основанию, указанному в константе через знак #

4) Вещественные

5) Символьные

10) Структура идентификаторов: Большая буква + нижний слеш + любой набор букв и цифр

11) Для того, чтобы объявить переменную используется конструкция название типа данных **имя** присваивание(по желанию);

12) Функции пишутся в виде:

возвращаемый тип данных **имя** (набор аргументов) {Тело}

Для того, чтобы вернуть значение из функции используется ключевое слово return

**например:** int func (int per){return 0;}

13) Для вывода в стандартный поток вывода используется конструкция cout <<

14) Для чтения из стандартного потока ввода используется конструкция cin >>

15) В языке реализованы все общепринятые логические операторы

16) В языке реализованы два типа областей видимости: глобальная, функциональная. Глобальные переменные объявляются перед описанием всех функций, доступны в любом месте программы. Функциональные переменные объявляются внутри функции. Доступны во время выполнения функции. После того как функция закончила свое выполнение, переменные удаляются из памяти и становятся недоступными.

17) Для вызова функции используется конструкция ИмяФункции(Параметры), где в круглых скобках, через запятую передаются аргументы с типом данных, указанных при объявлении функции.

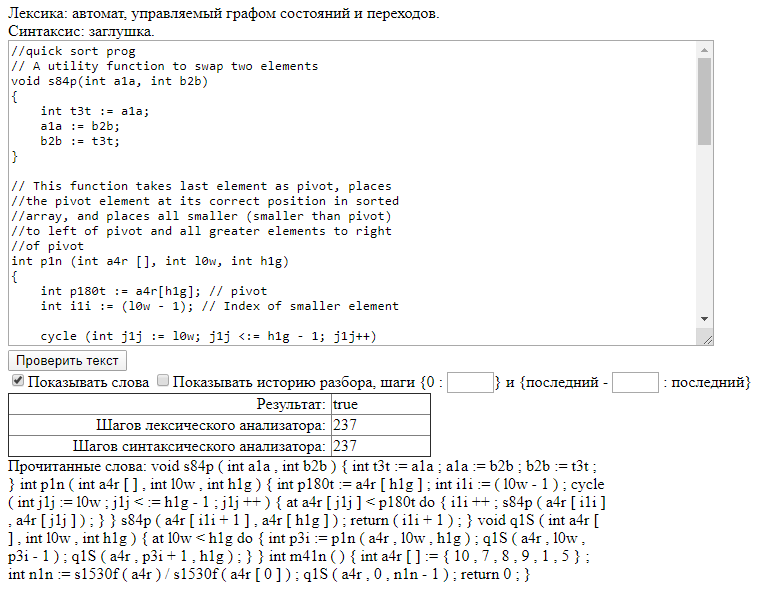


Рис.1 Проверка правильности написания лексики.

**Вывод:**

В данной работе были изучены конечные автоматы (КА) без памяти, способы определения КА – канонического, графового и табличного, методы построения недетерминированного КА по системе регулярных выражений, методы эквивалентных преобразований недетерминированных КА в оптимальные полностью определенные КА – лексические акцепторы.

Был доработан собственный конечный автомат и написана тестовая программа на разработанном языке.

**Приложение:**

**Приложение 1.**

//

// Created by пользователь //

#include <stdio.h>

#include <math.h>

//программа быстрая сортировка

// A utility function to swap two elements

**void** s84p(**int** a1a, **int** b2b)

{

**int** t3t := a1a;

a1a := b2b;

b2b := t3t;

}

// This function takes last element as pivot, places

//the pivot element at its correct position in sorted

//array, and places all smaller (smaller than pivot)

//to left of pivot and all greater elements to right

//of pivot

**int** p1n (**int** a4r [], **int** l0w, **int** h1g)

{

**int** p180t := a4r[h1g]; // pivot

**int** i1i := (l0w - 1); // Index of smaller element

cycle (**int** j1j := l0w; j1j <:= h1g - 1; j1j++)

{

// If current element is smaller than the pivot

at a4r[j1j] < p180t **do**

{

i1i++; // increment index of smaller element

s84p(a4r[i1i], a4r[j1j]);

}

}

s84p(a4r[i1i + 1], a4r[h1g]);

**return** (i1i + 1);

}

// The main function that implements QuickSort

//arr[] --> Array to be sorted,

//low --> Starting index,

//high --> Ending index \*/

**void** q1S(**int** a4r[], **int** l0w, **int** h1g)

{

at l0w < h1g **do**

{

// pi is partitioning index, arr[p] is now

// at right place

**int** p3i := p1n(a4r, l0w, h1g);

// Separately sort elements before

// partition and after partition

q1S(a4r, l0w, p3i - 1);

q1S(a4r, p3i + 1, h1g);

}

}

**int** m41n()

{

**int** a4r[] := {10, 7, 8, 9, 1, 5};

**int** n1n := s1530f(a4r) / s1530f(a4r[0]);

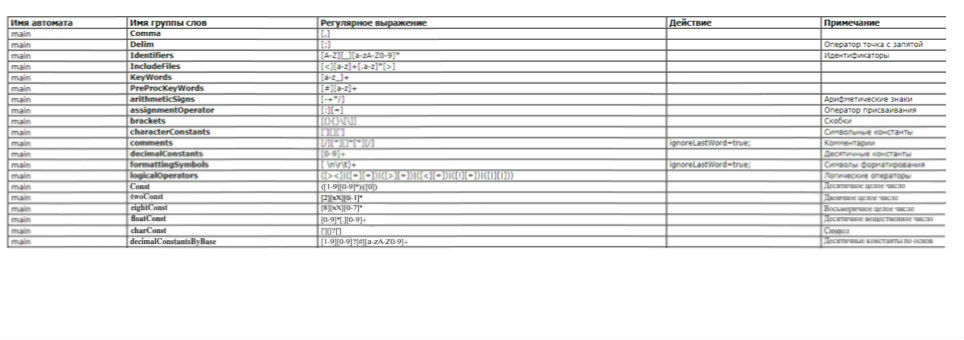
q1S(a4r, 0, n1n - 1);

**return** 0;

}

**Приложение 2**

*. Правила лексики и синтаксиса*

****

Текст xml-файла с системой правил:

<transLab>

<lexic>

<automat name='main'>

<rule groupWordsName='Comma'>

<expression>[,]</expression>

</rule>

<rule groupWordsName='Delim'>

<expression>[;]</expression>

<comment>Оператор точка с запятой</comment>

</rule>

<rule groupWordsName='Identifiers'>

<expression>[A-Z][\_][a-zA-Z0-9]\*</expression>

<comment>Идентификаторы</comment>

</rule>

<rule groupWordsName='IncludeFiles'>

<expression>[<][a-z]+[.a-z]\*[>]</expression>

</rule>

<rule groupWordsName='KeyWords'>

<expression>[a-z\_]+</expression>

</rule>

<rule groupWordsName='PreProcKeyWords'>

<expression>[#]("include")|("define")</expression>

</rule>

<rule groupWordsName='arithmeticSigns'>

<expression>[-+\*/]</expression>

<comment>Арифметические знаки</comment>

</rule>

<rule groupWordsName='assignmentOperator'>

<expression>[:][=]</expression>

<comment>Оператор присваивания</comment>

</rule>

<rule groupWordsName='brackets'>

<expression>[(){}\[\]]</expression>

<comment>Скобки</comment>

</rule>

<rule groupWordsName='characterConstants'>

<expression>['][][']</expression>

<comment>Символьные константы</comment>

</rule>

<rule groupWordsName='comments'>

<expression>[/][\*][]\*[\*][/]</expression>

<action>ignoreLastWord=true;</action>

<comment>Комментарии</comment>

</rule>

<rule groupWordsName='decimalСonstants'>

<expression>[0-9]+</expression>

<comment>Десятичные константы

17#3a6g 6#1234134 61#65fdgREG</comment>

</rule>

<rule groupWordsName='formattingSymbols'>

<expression>[ \n\r\t]+</expression>

<action>ignoreLastWord=true;</action>

<comment>Символы форматирования</comment>

</rule>

<rule groupWordsName='logicalOperators'>

<expression>([><]|([=][=])|([>][=])|([<][=])|([!][=])|([|][|]))</expression>

<comment>Логические операторы</comment>

</rule>

< rule groupWordsName='Const'>

<expression>([1-9][0-9]\*)|([0])</expression>

<comment>Десятичное целое число</comment>

</rule>

<rule groupWordsName='twoConst'>

<expression>[2][xX][0-1]\*</expression>

<comment>Двоичное целое число</comment>

</rule>

<rule groupWordsName='eightConst'>

<expression>[8][xX][0-7]\*</expression>

<comment>Восьмеричное целое число</comment>

</rule>

<rule groupWordsName='floatConst'>

<expression>[0-9]\*[.][0-9]+</expression>

<comment>Десятичное вещественное число</comment>   
 </rule>

</automat>

</lexic>

</transLab>