Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра вычислительной техники

Отчет по лабораторной работе № 2

по дисциплине «Теория формальных языков и компиляторов»

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

Студент: Смирнов А.М.

Вариант: 14431332

Группа: АВТ-709

Преподаватель: Малявко А.А.

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.

# Постановка задачи

1. Используя пакет ВебТрансЛаб, освоить:

- создание лексических правил на языке регулярных выражений (РВ);

- использование операций «+, \*, ?, конкатенации и выбора» языка РВ для построения сложных регулярных выражений;

- преобразование простой системы РВ в одноавтоматный лексический акцептор;

- добавление правил и действий в систему РВ для построения мультиавтоматного лексического акцептора;

2. Разработать (доработать разработанный при выполнении работы №1) 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, оценить их затраты времени в шагах на распознавание слов;

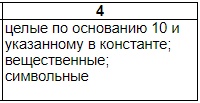
4. Изучить по тексту программного модуля способ реализации вызова действий, определенных в лексических правилах и алгоритм работы формирователя лексем.

5. Проверить функционирование конечных автоматов, построенных ВебТрансЛабом (подготовить тестовый пример программы на языке, заданном на курсовую работу, запустить каждый автомат на выполнение, протрассировать вручную работу лексического акцептора в графовой и табличной реализации, убедиться в работоспособности автоматов, в противном случае – доработать систему РВ и добиться правильного функционирования лексического акцептора).

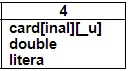
Вариант: 14431332



*Рис. 1 – Идентификаторы*



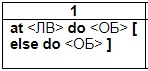
*Рис. 2 – Константы*

**

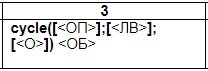
*Рисунок 3 – Типы*

**

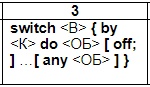
*Рисунок 4 – Оператор присваивания*

**

*Рисунок 5 – Условный оператор*

**

*Рисунок 6 – Оператор цикла*

**

*Рисунок 7 – Оператор переключателя*

# Ход работы

**Тестоваяпрограмма*:***

*// Сортировка заданного массива выбором.*

*#include <stdio.h>*

*#include <math.h>*

*void S\_electionsort(card A\_rr[], card N\_)*

*{*

*card I\_;*

*card J\_;*

*card M\_inidx;*

*cycle(I\_ := 0; I\_ < N\_ - 1; I\_++)*

*{*

*M\_inidx := I\_;*

*cycle(J\_ : = I\_ + 1; J\_ < N\_; J\_++) {*

*at(A\_rr[J\_] < A\_rr[M\_inidx]) do{*

*M\_inidx:= J\_;*

*}*

*}*

*card T\_emp := A\_rr[M\_inidx];*

*A\_rr[M\_inidx] := A\_rr[I\_];*

*A\_rr[I\_] := T\_emp;*

*}*

*}*

*card S\_tarter(){*

*const card N\_ := 10;*

*card A\_rr[N\_] := {5, 12, 6, 9, 1, 3, 23, 67, 99, 2};*

*card M\_;*

*S\_canf("%d", M\_);*

*switch(M\_){*

*by(1) do {*

*S\_electionsort(A\_rr, N\_);*

*}*

*off;*

*by(2) do {*

*printf("Hello, World!\n");*

*}*

*off;*

*any {*

*printf("Default output!\n");*

*}*

*}*

*return 0;*

*}*

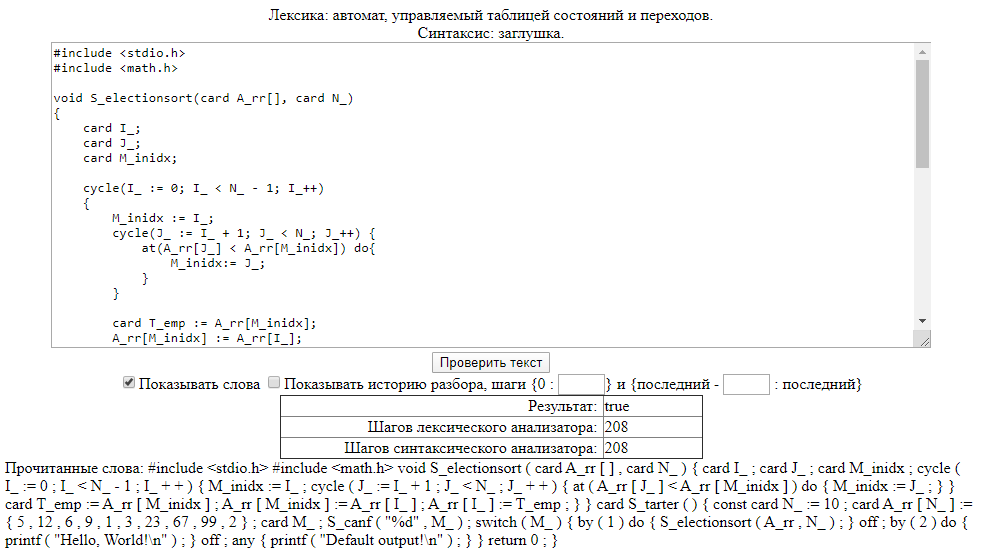


Рис.1. – Результат выполнения лексического анализатора

**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.

**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 *S\_tarter* 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 *S\_tarter*. Они начинаются с символа ‘#’, такие как *#define* или *#include.*

Пример:

*#include <stdio.h>*

*#define P\_i 3.14;*

**Точка входа**

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

Точка входа определяется ключевым словом *S\_tarter.*

То есть программа всегда начинает свою работу с выполнения функции *S\_tarter* и заканчивает свое выполнение с окончанием операторов в ней.

Начать выполнение программы без этой функции невозможно.

В программе допускается только одна функция *S\_tarter*, 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.

Объявление: *card Z\_;*

Инициализация: *Z\_ := 23;*

Возможно инициализировать переменную при объявлении:

*card Z\_ := 23;*

**Типы переменных**

*card* – целочисленный тип данных.

*//cardinal* – ???.

*card\_u* – без знаковое целое число.

*double* – число с плавающей запятой.  
*litera* – символьный тип данных.

Существует тип *void* (аналогично языку С), но используется только для создания функций.

**Константы**

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

**Целые константы**

Целые константы могут быть десятичными, двоичными или восьмеричными:  
*6* – Константа по основанию 10

**Вещественные константы**

Вещественные константы могут быть только десятичными:  
*10.2  
100.0  
5.999*

**Символьные константы**

Символьные константы состоят из одного любого символа, заключенного в одинарные кавычки:

*‘a’  
‘0’  
‘<’*

**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.

Если функция имеет возвращаемое значение, то тогда в ней должно быть указано ключевое слово *return* которое передаст управление оператору, следующему за вызовом этой функции. Оно должно быть указано во всех местах возможного выхода из функции. В функциях с типом *void*, не требуется указания ключевого слово *return*.

Пример:

*card F\_aktorial(card N\_umber)*

*{*

*card R\_ezult = 1;*

*card I\_ = 1*

*cycle(;I\_ <= N\_umber;){*

*R\_ezult = R\_ezult \* I\_;*

*I\_++;*

*}*

*return R\_ezult;*

*}*

Тип аргументов, переданных в функцию, должен совпадать с типом аргументов, указанных при объявлении функции.

Пример вызова вышеописанной функции:

*card N\_ := 10;*

*F\_aktorial(N\_);*

**Выполняемые операторы**

Присваивание:

*<Идентификатор>:=<Произвольное\_выражение>*

Пример:

*N\_ := 10*

Условный оператор:

*at (<Логическое\_выражение>) do <Оператор\_или\_блок> [else do <Оператор\_или\_блок>]*

Пример:

*at ((M\_ > N\_)  && (M\_ < (N\_ + 5))) do M\_--;*

Оператор цикла:

*cycle([<Оператор\_присваивания>];[<Логическое\_выражение>];[<Одиночный\_оператор>])<Оператор\_или\_блок>*

Выражение создает цикл, который выполняет указанное выражение до тех пор, пока условие не станет ложным. Условие проверяется после выполнения выражения, то есть выражение выполнится как минимум один раз.

Пример:

*card I\_;*

*card M\_ := 5;*

*cycle(I\_ := 0; I\_ < 10; I\_++)*

*{*

*M\_ := M\_ \* I\_;*

*}*

Оператор переключателя:

*switch<Произвольное\_выражение>*

*{*

*by<Константа> do<Оператор\_или\_блок>*

*[off;]…[any <Оператор\_или\_блок >]*

*}*

Оператор *break* передает управление оператору end, который заканчивает выполнение оператора переключателя и передает управление следующему оператору.

Пример:

*card N\_;*

*S\_canf("%d", &N\_);*

*switch (N\_)*

*{*

*by(1) do {*

*D\_osomething(N\_ + 5);*

*}*

*off;*

*by(2) do {*

*D\_osomething(N\_ + 10);*

*}*

*off;*

*any{*

*D\_osomething(N\_);*

*}*

*}*

**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:

\r – возврат каретки  
\n – перенос строки  
\t – строчная табуляция

**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

“ 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

+ 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.

Существуют следующие ключевые слова:  
*card*

*card\_u*

*cardinal*

*double*

*litera*

*void*

*return*

*cycle*

*switch*

*by*

*do*

*off*

*any*

*at*

*do*

*else*

Переопределить ключевые слова нельзя.

# Вывод

В ходе лабораторной работы были описаны основные свойства языка и разобраны основные операторы рассматриваемого языка. В результате лабораторной работы была разработана тестовая программа.

# Приложение

Текст xml-файла с системой правил:

<?xml version="1.0" encoding="windows-1251"?>

<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='realConstants'>

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

<comment>Вещественные константы</comment>

</rule>

<rule groupWordsName='StringConst'>

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

</rule>

</automat>

</lexic>

</transLab>