Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧЕТ**

по лабораторной работе №2

по дисциплине «Трансляторы и компиляторы»

## на тему «Разработка конечного автомата-распознавателя языка констант Си»

Выполнили студенты группы 22ВВП1

Демин М.С.

Беляев Д. И.

Приняли:

Дубинин В.Н.

Карамышева Н.С.

Пенза 2024

**Цель работы**

Изучение принципов функционирования лексического анализатора. Изучение алгоритма преобразования регулярных грамматик в конечный автомат.

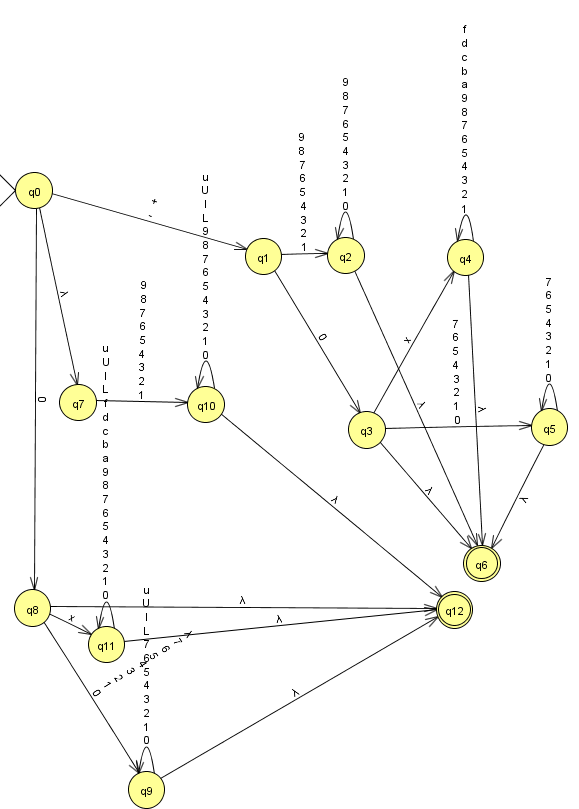
**Задание**

1. На основе грамматики, полученной в результате выполнения лабораторной работы №1 построить конечный автомат, реализующий распознавание целочисленных констант языка C. Необходимо учесть особое состояние ошибки разбора входной последовательности терминальных символов. Результатом выполнения лабораторной работы должен явиться детерминированный конечный автомат, представленный в любой форме.
2. Используя конечный автомат, построенный в результате выполнения лабораторной работы №2 написать программу на языке C, реализующую распознавание целочисленных констант языка C.Результатом выполнения лабораторной работы должен явиться исполняемый файл программы.

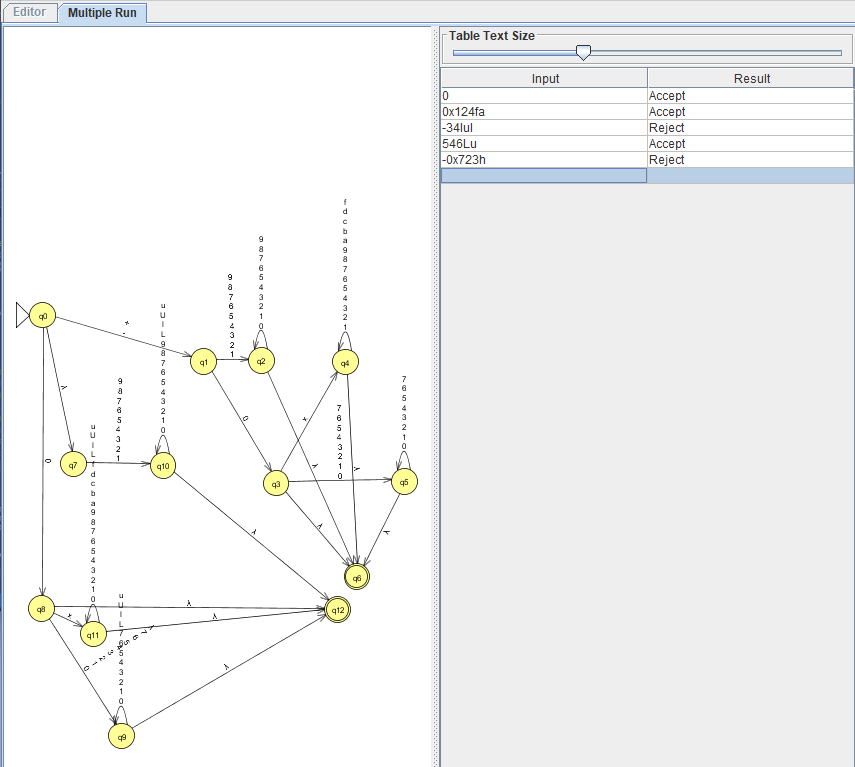
**Ход работы**

**ЧАСТЬ 1**

1. Реализовали детерминированный конечный автомат, который реализует распознавание целочисленных констант языка C.

****

1. Выполнили проверку автомата на различных константах.



**ЧАСТЬ 2**

Листинг

**main.cpp**

#include <stdio.h>

#include <ctype.h>

#include <stdbool.h>

const char\* testCases[] = {

"123;",

"+456;",

"-789;",

"0123;",

"0x1a2b;",

"-0xff;",

"123u;",

"0127;",

"0xFFUL;",

"12llu;",

"12ull;",

"1ull;",

"+0x101luL;",

"0x7bH;",

"078;",

"0;",

"+0x10x1lL;",

"+0x101LxL;",

"+0x101lUL;",

"+0x101l+L;",

"+0x101llu2;",

"+0x101llL;",

"+0l;",

"+++----1;",

"++++\_\_\_;",

"-0;",

"1UU;",

NULL

};

typedef enum

{

START,

SIGN,

ZERO,

DEC,

OCT,

HEX,

SUF\_U,

SUF\_L,

SUF\_UL,

SUF\_LU,

SUF\_LL,

SUF\_ULL,

SUF\_LLU,

END,

ERROR

} STATE;

bool isDecDigit(char c)

{

return c >= '0' && c <= '9';

}

bool isOctDigit(char c)

{

return c >= '0' && c <= '7' ;

}

bool isHexDigit(char c)

{

return (c >= '0' && c <= '9') || (c >= 'a' && c <= 'f');

}

STATE nextState(STATE curState, char c)

{

char lowC = tolower(c);

switch (curState)

{

case START:

if (lowC == '+' || c == '-') return SIGN;

if (lowC == '0') return ZERO;

if (isDecDigit(lowC)) return DEC;

break;

case SIGN:

if (c == '0') return ZERO;

if (isDecDigit(c)) return DEC;

break;

case ZERO:

if (lowC == ';') return END;

if (lowC == 'x') return HEX;

if (isOctDigit(lowC)) return OCT;

if (lowC == 'u') return SUF\_U;

if (lowC == 'l') return SUF\_L;

break;

case DEC:

if (isDecDigit(lowC)) return DEC;

if (lowC == 'u') return SUF\_U;

if (lowC == 'l') return SUF\_L;

if (lowC == ';') return END;

break;

case OCT:

if (isOctDigit(lowC)) return OCT;

if (lowC == 'u') return SUF\_U;

if (lowC == 'l') return SUF\_L;

if (lowC == ';') return END;

break;

case HEX:

if (isHexDigit(lowC)) return HEX;

if (lowC == 'u') return SUF\_U;

if (lowC == 'l') return SUF\_L;

if (lowC == ';') return END;

case SUF\_U:

if (lowC == 'l') return SUF\_UL;

if (lowC == ';') return END;

break;

case SUF\_L:

if (lowC == 'l') return SUF\_LL;

if (lowC == 'u') return SUF\_LU;

if (lowC == ';') return END;

break;

case SUF\_LL:

if (lowC == 'u') return SUF\_LLU;

if (lowC == ';') return END;

break;

case SUF\_UL:

if (lowC == 'l') return SUF\_ULL;

if (lowC == ';') return END;

break;

case SUF\_LU:

if (lowC == ';') return END;

break;

case SUF\_LLU:

if (lowC == ';') return END;

break;

case SUF\_ULL:

if (lowC == ';') return END;

break;

case END:

if (lowC == ';') return END;

return ERROR;

}

return ERROR;

}

bool isValidConst(const char\* input)

{

STATE state = START;

for (int i = 0; i < strlen(input); i++) {

state = nextState(state, input[i]);

if (state == ERROR) {

return false;

}

}

return state == END;

}

int main()

{

for (int i = 0; testCases[i] != NULL; i++) {

if (isValidConst(testCases[i])) {

printf("'%s' is a valid constant.\n", testCases[i]);

}

else {

printf("'%s' is not a valid constant.\n", testCases[i]);

}

}

return 0;

}

**Вывод**

В ходе выполнения лабораторной работы были изучены принципы функционирования лексического анализатора, алгоритмы преобразования регулярных грамматик в конечный автомат.