Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра ПОИТ

Дисциплина «Компиляторные технологии»

ОТЧЁТ

к лабораторной работе №1

РАСПОЗНАВАНИЕ И ПОИСК ЛЕКСЕМ

Вариант 3

Студент группы №451001

Цурко В.Е.

Руководитель, преподаватель

Болтак С.В.

Минск 2025

**СОДЕРЖАНИЕ**

[1 Задание к лабораторной работе 3](#_Toc211805404)

[2 Классы символов и регулярное выражение 4](#_Toc211805405)

[3 Недетерминированный конечный автомат (НКА) 4](#_Toc211805406)

[4 Детерминированный конечный автомат (ДКА) 4](#_Toc211805407)

[5 Тестирование программы 5](#_Toc211805408)

[6 Текст программы 8](#_Toc211805409)

## 1 Задание к лабораторной работе

В таблице 1 представлен перечень пронумерованных заданий. Номер задания соответствует номеру варианта (третий). Расчет номера задания:

Номер студенческого билета: 45100116

16 % 7 + 1 = 3

Таблица 1 – Варианты заданий к лабораторной работе №1

|  |  |  |
| --- | --- | --- |
| № варианта | Требования | Примеры корректных строк |
| 1 | Строковый литерал языка C с поддержкой следующих экранирующих последовательностей:   * \n * \t * \\ * \" | "Hello, world!"  "This \"string\" is valid."  "Hello\nworld\t!"  "Quotes are written this way: \\\"." |
| 2 | Строковый литерал языка Pascal (апостроф записывается, как два апострофа подряд). | 'Hello, world!'  'I don''t know.' |
| 3 | 16-чное число в синтаксисе C или Pascal. | $c001c0De  0xCa5EFeED |
| 4 | Классический абсолютный путь к файлу в Windows:   * латинская буква от A до Z; * двоеточие; * обратный слеш; * 0 или более блоков из допустимых символов, разделённых обратными слешами.   К допустимым относятся все символы, кроме «\*», «|», «\», «:», «"», «<», «>», «?» и «/».  (Для поиска подстрок недопустимым символом пути следует также считать пробел « ».) | C:\Windows\winmine.exe  D:\WebServer\home\site.by\www\.htaccess  Z:\autoexec.bat  N:\testfile.  X:\testfile2 |
| 5 | Двоичное, восьмеричное или 16-чное число в синтаксисе FASM (постфиксная форма записи). 16-чное число должно начинаться с цифры от 0 до 9. | 01101011B  54212110o  51245H  0B8Ch |
| 6 | Корректный идентификатор: начинается с буквы или нижнего подчёркивания, затем — произвольное количество букв, цифр и/или нижних подчёркиваний. | abCde  x0  Y  z\_1  test\_value\_is\_here  \_  \_\_custom |
| 7 | Адрес электронной почты (упрощённо).  Имя ящика — произвольный набор букв, цифр и символов «.» (точка). Доменное имя — одна или более групп, состоящих из букв, цифр и дефисов, разделённые точками. | test@localhost  vasya@pupkin.ru  ivan.ivanov@mail.bsuir.by  a.b.c@d-e-f.com |

## 2 Классы символов и регулярное выражение

Регулярное выражение: (((0)(x, X)) | ($))( [A..Fa..f] | [0..9])+

К выделенным в ходе выполнения лабораторной работы классам символов относятся:

* *Letters* ::= [A..Fa..f];
* *Digits* ::= [1..9];
* *Zero* ::= (0);
* *StartPascal* ::= ($);
* *ContinueC* ::= (x, X).

Регулярное выражение, соответствующее условию задачи, выглядит следующим образом: *((Zero ContinueC)|(StartPascal))(Letters|Digit| Zero)+.*

## 3 Недетерминированный конечный автомат (НКА)

На рисунке 3.1 представлена схема недетерминированного конечного автомата, обрабатывающего строку на входе и определяющего, является ли она 16-чным числом в синтаксисе С или Pascal.

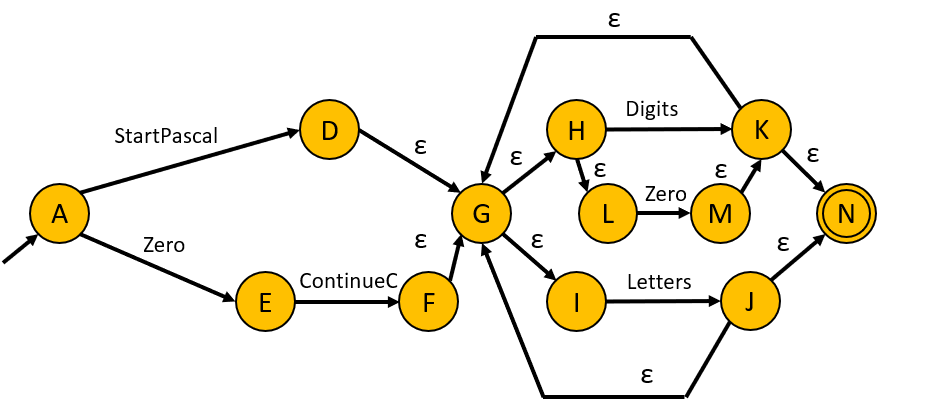


Рисунок 3.1 – Недетерминированный конечный автомат для 16-чного числа в С и Pascal

## 4 Детерминированный конечный автомат (ДКА)

На рисунке 4.1 представлена схема неоптимизированного детерминированного конечного автомата, обрабатывающего 16-чное число в С и Pascal.

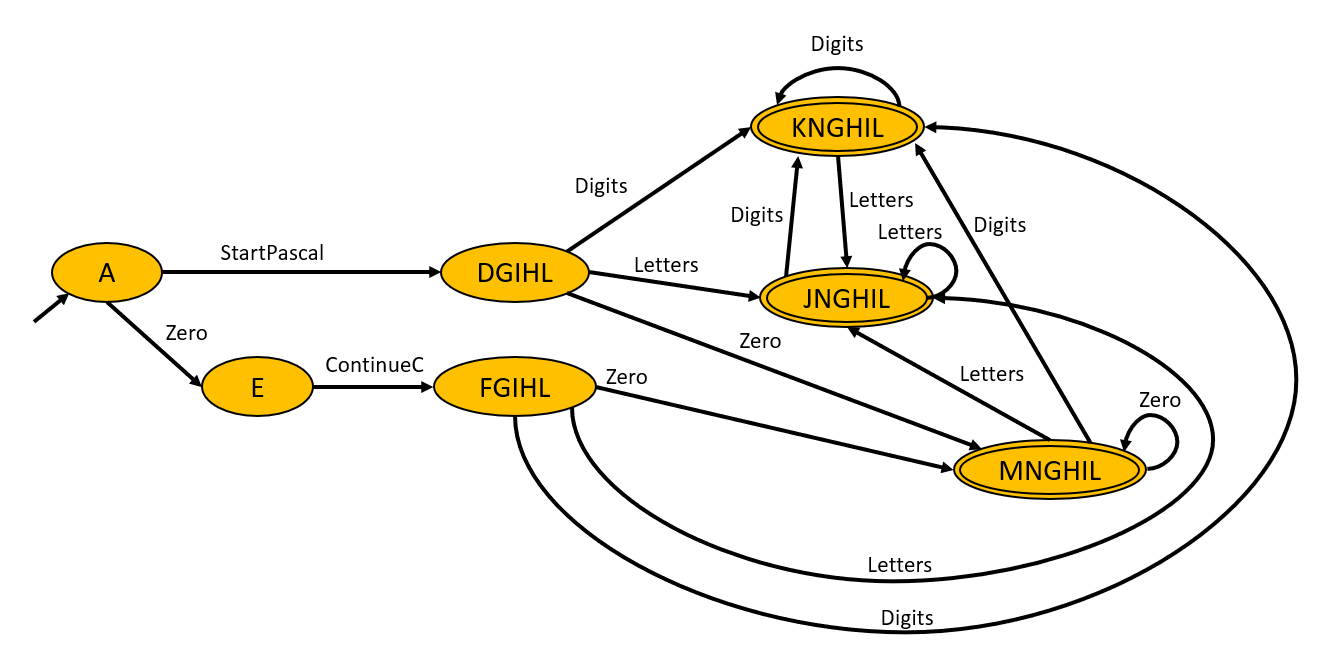


Рисунок 4.1 - Неоптимизированный детерминированный конечный автомат для 16-чного числа в С и Pascal

На рисунке 4.2 представлена схема оптимизированного детерминированного конечного автомата, обрабатывающего 16-чное число в С и Pascal.

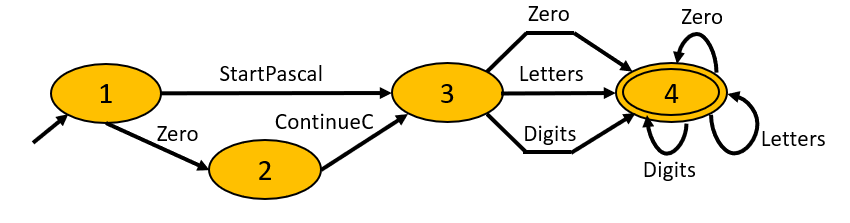


Рисунок 4.2 – Оптимизированный детерминированный конечный автомат для 16-чного числа в С и Pascal

Таблица 2 – Матрица переходов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер состояния | StartN | Digit | Letters |
| 0 | 0 | 0 | 0 |
| 1 | 2 | 0 | 0 |
| 2 | 0 | 3 | 3 |
| 3 | 0 | 3 | 3 |

## 5 Тестирование программы

В таблице 3 представлены ключевые тесты, проверяющие корректность результата работы программы.

Ожидаемый результат: если введённая строка является 16-чным числом в языке Pascal, то программа выводит текст “It is a Pascal hex num”; если введенная строка является 16-чным числом в языке C, то программа выводит текст “It is a C hex num”; если строка не является ни тем, ни тем, то соответствующий вывод “It is not a hex num in C or Pascal”. Помимо этого, программа выводит все подстроки, которые соответствуют 16-чному числу в С или Pascal, по группам: “Founded Pascal substrings:” и “Founded C substrings:”. Если подстроки для какой-то из групп не найдены, то вывод для Pascal – “There arent any Pascal substrings”, для С – “There arent any C substrings”

Таблица 3 – Тесты программы

|  |  |  |  |
| --- | --- | --- | --- |
| Номер и исход теста | Входные  данные | Ожидаемый результат | Результат работы программы |
| 1 (пройден успешно) | Нажатие Enter | It is not a hex num in C or Pascal  There arent any Pascal substrings  There arent any C substrings |  |
| 2 (пройден успешно) | 1234455553 | It is not a hex num in C or Pascal  There arent any Pascal substrings  There arent any C substrings |  |
| 3 (пройден успешно) | 0x0x | It is not a hex num in C or Pascal  There arent any Pascal substrings  Founded C substrings:  0x0 |  |
| 4 (пройден успешно) | $0x0X | It is not a hex num in C or Pascal  Founded Pascal substrings:  $0  Founded C substrings:  0x0 |  |

Таблица 4 – Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| 5 (пройден успешно) | $abCdEf | It is a Pascal hex num  Founded Pascal substrings:  $a  $ab  $abC  $abCd  $abCdE  $abCdEf  There arent any C substrings |  |
| 6 (пройден успешно) | $120xAB | It is not a hex num in C or Pascal  Founded Pascal substrings:  $1  $12  $120  Founded C substrings:  0xA  0xAB |  |
| 7 (пройден успешно) | привет, друг 0xff | It is not a hex num in C or Pascal  There arent any Pascal substrings  Founded C substrings:  0xf  0xff |  |

Таблица 4 – Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| 8 (пройден успешно) | "2320x001111$1" | It is not a hex num in C or Pascal  Founded Pascal substrings:  $1  Founded C substrings:  0x0  0x00  0x001  0x0011  0x00111  0x001111 |  |

## 6 Текст программы

#include <stdio.h>

#include <stdlib.h>

enum TCharType{

Unknown,

StartN,

Digit,

Letters,

TCharTypeCount //количество состояний

};

const int isFinalState[] = {0, 0, 0, 1};

const int values[4][TCharTypeCount] = {{0, 0, 0, 0}, {0, 2, 0, 0}, {0, 0, 3, 3}, {0, 0, 3, 3}};

int getCharType(char \*symbol, int \*addC);

int getInput(char \*\*str);

int checkString(char str[], size\_t length);

void \* checkRealloc(void \*pointer, size\_t newLength);

void findCandPascalNums(char \*str);

void grow(char \*\*str, int \*capacity);

void printSubstrings(char \*pascalStrings, int length\_Pascal, char \*cStrings, int length\_C);

int main(int argc, char const \*argv[])

{

while (1){

size\_t len = 0;

char \*string;

printf("Input your string: ");

len = getInput(&string);

if (isFinalState[checkString(string, len)]){

if (\*string == '$') printf("It is a Pascal hex num \n");

else printf("It is a C hex num \n");

}

else {

printf("It is not a hex num in C or Pascal \n");

}

findCandPascalNums(string);

free(string);

}

return 0;

}

//функция для ввода строки, возвращает длину

int getInput(char \*\*string)

{

char symbol;

char \*str;

size\_t capacity = 16, length = 0;

str = calloc(capacity, sizeof(char));

while ((symbol = getchar()) != '\n')

{

if (length == capacity) //if (length > capacity - 2)

{

capacity = capacity << 1;

str = checkRealloc(str, capacity);

}

str[length] = symbol;

length++; //только для Char 1 байт

};

str = checkRealloc(str, length + 1);

\*string = str;

return length;

}

void \* checkRealloc(void \*pointer, size\_t newLength){

void \*temp;

temp = realloc(pointer, newLength);

if (temp != NULL) return temp;

else{

printf("Stopped because of OutOfMemory");

exit(1);

}

}

int getCharType(char \*symbol, int \*addC){

/\*

Возвращаем номер класса, которому принадлежит текущий элемент:

Letters ::= [A..Fa..f];

Digit ::= [0..9];

StartN ::= ($, 0x, 0X).

\*/

\*addC = 0;

char elem = \*symbol;

if (elem == '$') return StartN;

if (((elem >= 'A') && (elem <= 'F')) || ((elem >= 'a') && (elem <= 'f'))) return Letters;

else if (elem == '0') {

if (\*(symbol + 1) != '\0'){ //проверяем, можем ли мы проверить следующий символ

char nextElem = \*(symbol + 1);

if ((nextElem == 'x') || (nextElem == 'X')){

\*addC = 1;

return StartN;

}

}

return Digit;

}

else if ((elem >= '1') && (elem <= '9')) return Digit;

else return Unknown;

}

int checkString(char str[], size\_t length){

int state = 1, \*addC;

addC = malloc(sizeof(int));

\*addC = 0;

for (size\_t i = 0; i < length; i++){

state = values[state][getCharType(str + i + \*addC, addC)];

}

free(addC);

return state;

}

void grow(char \*\*str, int \*capacity){

\*capacity = (\*capacity) \* 2;

\*str = (char \*) checkRealloc(\*str, \*capacity);

}

void findCandPascalNums(char \*str){

int i = 0, \*addC, state = 1, typeOfChar, indexStartNum, isCstring;

addC = malloc(sizeof(int));

\*addC = 0;

int \*C\_capacity, \*Pascal\_capacity, length\_C, length\_Pascal;

length\_C = 0;

length\_Pascal = 0;

C\_capacity = malloc(sizeof(int));

Pascal\_capacity = malloc(sizeof(int));

\*C\_capacity = 16;

\*Pascal\_capacity = 16;

char \*cStrings, \*pascalStrings; //надо память выделять под них и потом записывать туда соотв строки

cStrings = malloc(\*C\_capacity);

pascalStrings = malloc(\*Pascal\_capacity);

while (str[i] != '\0'){

typeOfChar = getCharType(str + i/\* + \*addC\*/, addC);

state = values[state][typeOfChar];

if (isFinalState[state]){

if (isCstring){

for (int index = indexStartNum; index <= i; index++){

if (length\_C == \*C\_capacity) grow(&cStrings, C\_capacity);

cStrings[length\_C] = str[index];

length\_C++;

}

}

else {

for (int index = indexStartNum; index <= i; index++){

if (length\_Pascal == \*Pascal\_capacity) grow(&pascalStrings, Pascal\_capacity);

pascalStrings[length\_Pascal] = str[index];

length\_Pascal++;

}

}

}

else if ((state == 0) && (typeOfChar != StartN)) state = 1;

else if ((typeOfChar == StartN) && (state == 2)){

indexStartNum = i;

isCstring = \*addC;

}

else if ((state == 0) && (typeOfChar == StartN)){

//isCstring = \*addC;

if (\*addC){

if (isCstring) {

for (int index = indexStartNum; index <= i; index++){

if (length\_C == \*C\_capacity) grow(&cStrings, C\_capacity);

cStrings[length\_C] = str[index];

length\_C++;

}

}

else {

for (int index = indexStartNum; index <= i; index++){

if (length\_Pascal == \*Pascal\_capacity) grow(&pascalStrings, Pascal\_capacity);

pascalStrings[length\_Pascal] = str[index];

length\_Pascal++;

}

}

}

state = 1;

continue;

}

i = i + 1 + \*addC;

}

printSubstrings(pascalStrings, length\_Pascal, cStrings, length\_C);

//избавляемся от утечек памяти

free(addC);

free(C\_capacity);

free(Pascal\_capacity);

free(cStrings);

free(pascalStrings);

}

void printSubstrings(char \*pascalStrings, int length\_Pascal, char \*cStrings, int length\_C){

//Вывод паскаль чисел

if (length\_Pascal != 0){

printf("Founded Pascal substrings:\n");

for (int i = 0; i < length\_Pascal; i++){

if ((pascalStrings[i] == '$') && (i != 0)) printf("\n%c", pascalStrings[i]);

else printf("%c", pascalStrings[i]);

}

printf("\n\n");

}

else printf("There arent any Pascal substrings\n\n");

//Вывод си чисел

if (length\_C != 0){

printf("Founded C substrings:\n");

int \*addC;

addC = malloc(sizeof(int));

\*addC = 0;

for (int i = 0; i < length\_C; i++){

if ((getCharType(cStrings + i, addC) == StartN) && (i != 0)) printf("\n%c", cStrings[i]);

else printf("%c", cStrings[i]);

}

printf("\n\n");

free(addC);

}

else printf("There arent any C substrings\n\n");

}