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

**ОТЧЕТ**

по лабораторной работе №2 часть 3

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. Исследование способов практической реализации элементов лексического анализатора.

**Задание**

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:**

1. **main.cpp**

#include <iostream>

#include <map>

#include "RightFormatter.h"

std::map<std::string, NUMBER\_NS> dict = {

{"1", NUMBER\_NS::DEC},

{"+1", NUMBER\_NS::DEC},

{"-1", NUMBER\_NS::DEC},

{"0x1", NUMBER\_NS::HEX},

{"0xb", NUMBER\_NS::HEX},

{"0012", NUMBER\_NS::OCT},

{"1l", NUMBER\_NS::DEC},

{"+1l", NUMBER\_NS::DEC},

{"-1l", NUMBER\_NS::DEC},

{"0x1l", NUMBER\_NS::HEX},

{"0xbl", NUMBER\_NS::HEX},

{"0012l", NUMBER\_NS::OCT},

{"1u", NUMBER\_NS::DEC},

{"+1u", NUMBER\_NS::DEC},

{"-1u", NUMBER\_NS::DEC},

{"0x1u", NUMBER\_NS::HEX},

{"0xbu", NUMBER\_NS::HEX},

{"0012u", NUMBER\_NS::OCT},

{"1ul", NUMBER\_NS::DEC},

{"+1ul", NUMBER\_NS::DEC},

{"-1ul", NUMBER\_NS::DEC},

{"0x1ul", NUMBER\_NS::HEX},

{"0xbul", NUMBER\_NS::HEX},

{"0012ul", NUMBER\_NS::OCT},

{"1lu", NUMBER\_NS::DEC},

{"+1lu", NUMBER\_NS::DEC},

{"-1lu", NUMBER\_NS::DEC},

{"0x1lu", NUMBER\_NS::HEX},

{"0xblu", NUMBER\_NS::HEX},

{"0012lu", NUMBER\_NS::OCT},

{"0x01v2lu", NUMBER\_NS::NONE},

{"0xABalu", NUMBER\_NS::HEX},

{"+++--0xABalu", NUMBER\_NS::NONE},

{"0xABalu++", NUMBER\_NS::NONE},

{"xABalu", NUMBER\_NS::NONE},

{"0ABalu", NUMBER\_NS::NONE},

{"01ABalu", NUMBER\_NS::NONE},

{"", NUMBER\_NS::NONE},

{"0", NUMBER\_NS::DEC},

{" ", NUMBER\_NS::NONE},

{"+", NUMBER\_NS::NONE},

{"l", NUMBER\_NS::NONE},

{"+l", NUMBER\_NS::NONE},

{"+1uul", NUMBER\_NS::NONE},

{"+0xlU", NUMBER\_NS::NONE},

{"+0lU", NUMBER\_NS::DEC},

{"+0x1lU", NUMBER\_NS::HEX},

{"+0x-", NUMBER\_NS::NONE},

{"+01x1lU", NUMBER\_NS::NONE},

{"+0101lAU", NUMBER\_NS::NONE},

{"+0101ll", NUMBER\_NS::OCT},

{"+101ll", NUMBER\_NS::DEC},

{"+101uu", NUMBER\_NS::NONE},

{"+101Ll", NUMBER\_NS::DEC},

{"+101Lll", NUMBER\_NS::NONE},

{"+101uLL", NUMBER\_NS::DEC},

{"+101llu", NUMBER\_NS::DEC},

{"+101lLu", NUMBER\_NS::DEC},

{"+0x101luL", NUMBER\_NS::NONE},

};

std::string ConvertFormatToString(NUMBER\_NS ns)

{

switch (ns) {

case NUMBER\_NS::DEC:

return std::string{"number is dec"};

case NUMBER\_NS::OCT:

return std::string{ "number is oct" };

case NUMBER\_NS::HEX:

return std::string{ "number is hex" };

default:

return std::string{ "invalid number format" };

}

}

int main()

{

RightFormatChecker formatter;

for (auto& element : dict) {

NUMBER\_NS ns = formatter.canNumberExist(element.first);

bool actualResult = (int)element.second == (int)ns;

std::cout << actualResult << " '" << element.first << "' " << ConvertFormatToString(ns) << std::endl;

}

return 0;

}

1. **RightFormatter.сpp**

#include "RightFormatter.h"

#include <algorithm>

#include <cctype>

int a = 10llu;

NUMBER\_NS RightFormatChecker::canNumberExist(const std::string number)

{

if (number.length() < 1) {

return NUMBER\_NS::NONE;

}

if (isRightDecFormat(number)) {

return NUMBER\_NS::DEC;

}

else if (isRightOctFormat(number)) {

return NUMBER\_NS::OCT;

}

else if (isRightHexFormat(number)) {

return NUMBER\_NS::HEX;

}

return NUMBER\_NS::NONE;

}

bool RightFormatChecker::isRightOctFormat(const std::string number)

{

if (!isRightLength(number, NUMBER\_NS::OCT) || !isRightSignFormat(number) || !isRightSizeFormat(number)) {

return false;

}

std::string trimNumber = trimString(number);

int length = trimNumber.length();

if (trimNumber[0] != '0') {

return false;

}

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

size\_t pos = OCT\_SYMBOLS.find(trimNumber[i]);

if (pos != std::string::npos) {

continue;

}

return false;

}

return true;

}

bool RightFormatChecker::isRightHexFormat(const std::string number)

{

if (!isRightLength(number, NUMBER\_NS::HEX) || !isRightSignFormat(number) || !isRightSizeFormat(number)) {

return false;

}

std::string trimNumber = trimString(number);

int length = trimNumber.length();

if (trimNumber[0] != '0' || trimNumber[1] != 'x') {

return false;

}

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

size\_t pos = HEX\_SYMBOLS.find(trimNumber[i]);

if (pos != std::string::npos) {

continue;

}

return false;

}

return true;

}

bool RightFormatChecker::isRightDecFormat(const std::string number)

{

if (!isRightLength(number, NUMBER\_NS::DEC) || !isRightSignFormat(number) || !isRightSizeFormat(number)) {

return false;

}

std::string trimNumber = trimString(number);

int length = trimNumber.length();

if (trimNumber[0] == '0' && length > 1) {

return false;

}

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

size\_t pos = DEC\_SYMBOLS.find(trimNumber[i]);

if (pos != std::string::npos) {

continue;

}

return false;

}

return true;

}

std::string RightFormatChecker::trimString(const std::string number)

{

std::string trimNumber = number;

for (char symbol : LU\_SYMBOLS) {

trimNumber.erase(std::remove(trimNumber.begin(), trimNumber.end(), symbol), trimNumber.end());

}

for (char symbol : SIGN\_SYMBOLS) {

trimNumber.erase(std::remove(trimNumber.begin(), trimNumber.end(), symbol), trimNumber.end());

}

return trimNumber;

}

bool RightFormatChecker::isRightLength(const std::string number, NUMBER\_NS ns)

{

int length = trimString(number).length();

switch (ns) {

case NUMBER\_NS::DEC:

return length > 0;

case NUMBER\_NS::OCT:

return length > 1;

case NUMBER\_NS::HEX:

return length > 2;

default:

return false;

}

}

bool RightFormatChecker::isRightSignFormat(const std::string number)

{

std::string signNumber = SIGN\_SYMBOLS.find(number[0]) == std::string::npos ? '+' + number : number;

int length = signNumber.length();

if (length < 2) {

return false;

}

for (int i = 1; i < length; i++) {

char symbol = number[i];

if (SIGN\_SYMBOLS.find(symbol) == std::string::npos) {

continue;

}

return false;

}

return true;

}

bool RightFormatChecker::isRightSizeFormat(const std::string number)

{

std::string lowerNumber = number;

for (char& c : lowerNumber) {

c = c | ' ';

}

int length = lowerNumber.length();

for (char symbol : LU\_SYMBOLS) {

int count = std::count(lowerNumber.begin(), lowerNumber.end(), symbol);

size\_t position = lowerNumber.find(symbol);

switch (symbol)

{

case ('l'):

if (count < 2) {

if (position == std::string::npos || position == length - 1) {

break;

}

if (LU\_SYMBOLS.find(lowerNumber[position + 1]) != std::string::npos) {

break;

}

}

if (count == 2) {

if (position < length - 3) {

break;

}

if (lowerNumber[position] == lowerNumber[position + 1] && lowerNumber[position] == symbol) {

break;

}

}

return false;

case ('u'):

if (count > 1) {

return false;

}

if (position == std::string::npos || position == length - 1) {

break;

}

if (LU\_SYMBOLS.find(lowerNumber[position + 1]) != std::string::npos) {

break;

}

return false;

}

}

return true;

}

**Вывод**

В ходе выполнения лабораторной работы были изучены принципы функционирования лексического анализатора, исследованы способы практической реализации элементов лексического анализатора