Федеральное государственное бюджетное образовательное учреждение высшего образования «Национальный исследовательский университет «МЭИ»

Институт информационных и вычислительных технологий

Кафедра Управления и интеллектуальных технологий

**Отчёт по лабораторной работе № 4**

По курсу «Разработка ПО систем управления»

«Основы языка С++»

# Выполнил студент группы А-01-19

# Кононенко Александр Вячеславович

# Проверили:

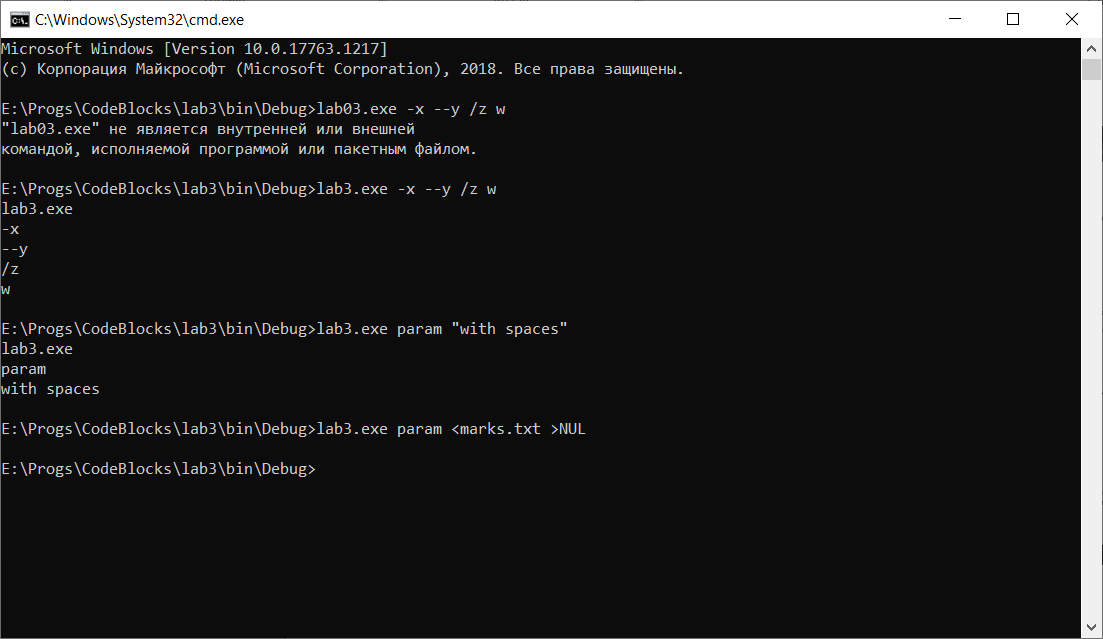
# Мохов А. С

# Козлюк Д. А

Москва 2020

Задача: с помощью API cURL осуществить считывание данных с сайта, а также осуществить вывод в svg файл версии ОС Windows и имени компьютера пользователя.

**Пункт 3.2.5 ЛР 4.1**



В 1 случае 5 аргументов разделены пробелами, поэтому функция main принимает 5 аргументов, которые являются 5 массивами символов типа char.

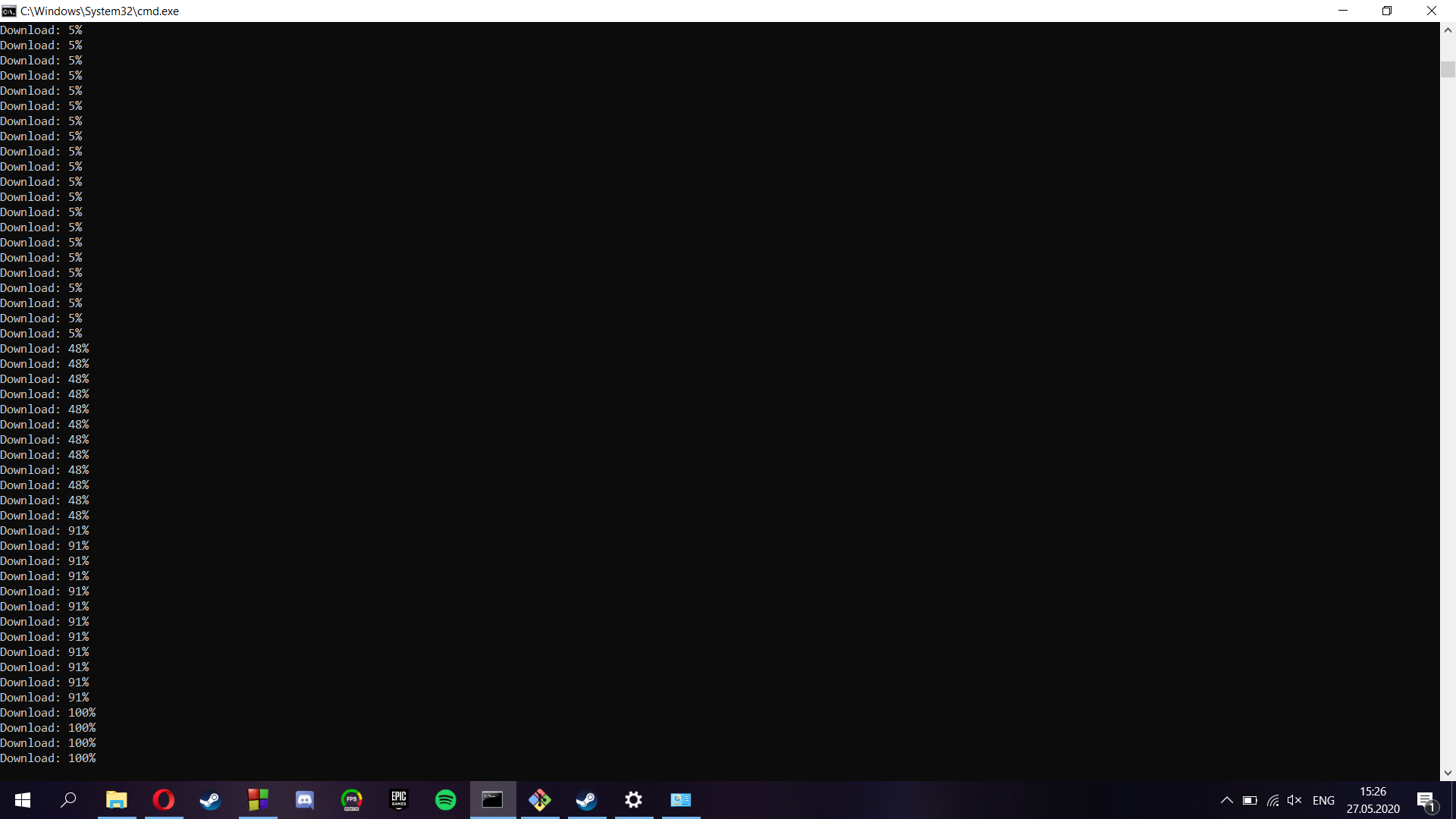
Во 2 случае функция main принимает 3 аргумента, один из них задаётся с двойными кавычками, что определяет область внутри кавычек как единый массив символов.

В 3 случае всё выводится в пустой поток.

Логика решения ЛР 4.1 индивидуального задания 7:

С помощью cURL нужно было выводить на экран информацию о прогрессе скачивания файла. Это осуществлялось с помощью опции CURLOPT\_PROGRESSDATA. В функции обратного вызова для этой опции принимается указатель на буфер, значение скаченной информации и значение размера всего файла. Вывод на экран процента выполнения загрузки: процент = скаченное / размер файла \* 100. При инициализации размера файла в функцию передаётся, что размер файла равен 0, и процент скачивания выводится не корректно, поэтому при таком параметре нужно обнулять процент скачивания.

При тестировании индивидуального задания я, с помощью стороннего приложения, ограничил скорость интернет соединения до 3 кбайт/с, чтобы убедиться в правильном выполнении программы:



Код программы:

**main.cpp**

#include <iostream>

#include <vector>

#include "histogram.h"

#include "avg\_bin+scaling.h"

#include <curl/curl.h>

#include <sstream>

#include <string>

#include <windows.h>

using namespace std;

int progress\_callback(void \*clientp, double dltotal, double dlnow, double ultotal, double ulnow)

{

int dl\_percent;

if (dltotal == 0)

{

dl\_percent = 0;

}

else

{

dl\_percent = dlnow / dltotal \* 100;

}

cerr << "Download: " << dl\_percent << "%\n";

return 0;

}

Input

read\_input(istream& in, bool prompt)

{

Input data;

if (prompt) cerr << "Enter number count: ";

size\_t number\_count;

in >> number\_count;

data.number\_count = number\_count;

if (prompt) cerr << "Enter numbers: ";

data.numbers = input\_numbers(in, number\_count);

size\_t bin\_count;

if (prompt) cerr << "Enter column count: ";

in >> bin\_count;

data.bin\_count = bin\_count;

return data;

}

size\_t

write\_data(void\* items, size\_t item\_size, size\_t item\_count, void\* ctx) {

if (ctx)

{

const size\_t data\_size = item\_size \* item\_count;

stringstream\* buffer = reinterpret\_cast<stringstream\*>(ctx);

buffer->write(reinterpret\_cast<const char\*>(items), data\_size);

return data\_size;

}

else return 0;

}

Input

download(const string& address) {

stringstream buffer;

curl\_global\_init(CURL\_GLOBAL\_ALL);

CURL\* curl = curl\_easy\_init();

if(curl)

{

CURLcode res;

curl\_easy\_setopt(curl, CURLOPT\_PROGRESSDATA, &buffer);

curl\_easy\_setopt(curl, CURLOPT\_NOPROGRESS, 0);

curl\_easy\_setopt(curl, CURLOPT\_PROGRESSFUNCTION, progress\_callback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &buffer);

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, write\_data);

curl\_easy\_setopt(curl, CURLOPT\_URL, address.c\_str());

res = curl\_easy\_perform(curl);

if (res != CURLE\_OK)

{

cout << curl\_easy\_strerror(res);

exit(1);

}

curl\_easy\_cleanup(curl);

}

return read\_input(buffer, false);

}

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

{

DWORD info = GetVersion();

DWORD mask = 0x0000ffff;

DWORD version = info & mask;

DWORD platform = info >> 16;

DWORD build = platform;

DWORD version\_minor = (version & 0x0000ff00) >> 8;

DWORD version\_major = version & 0x000000ff;

char buffer[MAX\_COMPUTERNAME\_LENGTH+1];

DWORD size;

size=sizeof(buffer);

GetComputerName(buffer,&size);

Input input;

if (argc > 1) {

input = download(argv[1]);

}

else {

input = read\_input(cin, true);

}

const auto bins = make\_histogram(input);

double scaling = scale(bins);

size\_t avg\_bin = average\_bin(input.number\_count, input.bin\_count);

show\_histogram\_svg(bins, scaling, avg\_bin, version\_major, version\_minor, build, buffer);

return 0;

}

**histogram.cpp**

#include "histogram.h"

#include "svg.h"

#include <iostream>

#include <vector>

vector<double> input\_numbers(istream& in, const size\_t number\_count) {

vector<double> result(number\_count);

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

in >> result[i];

}

return result;

}

vector <size\_t> make\_histogram(Input input)

{

double min = input.numbers[0];

double max = input.numbers[0];

find\_minmax(input.numbers,min,max);

vector<size\_t> bins(input.bin\_count,0);

for (double number : input.numbers)

{

size\_t bin;

bin = (number - min) / (max - min) \* input.bin\_count;

if (bin == input.bin\_count)

{

bin--;

}

bins[bin]++;

}

return bins;

}

void

show\_histogram\_svg(const vector<size\_t>& bins, double scaling, size\_t avg\_bin, int version\_major, int version\_minor, int build, string PC\_name)

{

const auto IMAGE\_WIDTH = 400;

const auto IMAGE\_HEIGHT = 300;

const auto TEXT\_LEFT = 20;

const auto TEXT\_BASELINE = 20;

const auto TEXT\_WIDTH = 50;

const auto BIN\_HEIGHT = 30;

const auto BLOCK\_WIDTH = 10;

svg\_begin(IMAGE\_WIDTH, IMAGE\_HEIGHT);

double top = 0;

for (size\_t bin : bins)

{

string stroke, fill;

if (bin <= avg\_bin)

{

stroke = "green";

fill = "#00FF00";

}

else

{

stroke = "red";

fill = "#FF0000";

}

const double bin\_width = BLOCK\_WIDTH \* bin \* scaling;

svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(bin));

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, stroke, fill);

top += BIN\_HEIGHT;

}

win\_version(TEXT\_LEFT, top + TEXT\_BASELINE, version\_major, version\_minor, build, PC\_name);

svg\_end();

}

void find\_minmax(const vector<double> numbers, double& min, double& max)

{

if (numbers.size()!= 0)

{

min = numbers[0];

max = numbers[0];

for (double number : numbers)

{

if (number < min)

{

min = number;

}

if (number > max)

{

max = number;

}

}

}

else cerr << "ERROR1\n";

}

**svg.cpp**

#include <string>

#include <iostream>

#include <fstream>

#include "svg.h"

using namespace std;

void svg\_rect(double x, double y, double width, double height, string stroke, string fill)

{

cout << "<rect x='"<<x<< "' y='" <<y<<"' width='" <<width <<"' height='" <<height <<"' stroke='"<<stroke<<"' fill='"<<fill<<"'/>";

cout << "\n";

}

void svg\_text(double left, double baseline, string text)

{

cout << "<text x='" << left << "' y='" << baseline <<"'>" <<text <<"</text>";

cout << "\n";

}

void svg\_begin(double width, double height)

{

cout << "<?xml version='1.0' encoding='UTF-8'?>\n";

cout << "<svg ";

cout << "width='" << width << "' ";

cout << "height='" << height << "' ";

cout << "viewBox='0 0 " << width << " " << height << "' ";

cout << "xmlns='http://www.w3.org/2000/svg'>\n";

}

void win\_version(double left, double baseline, int version\_major, int version\_minor, int build, string PC\_name)

{

cout << "<text x='" << left << "' y='" << baseline <<"'> Windows v" << version\_major << "." << version\_minor << " (build " << build << ") " << PC\_name << "</text>";

cout << "\n";

}

void svg\_end()

{

cout << "</svg>\n";

}

**avg\_bin+scaling.cpp**

#include <iostream>

#include <vector>

#include "avg\_bin+scaling.h"

double scale (const vector <size\_t>& bins)

{

const size\_t MAX\_ASTERISK = 40;

size\_t max\_count = 0;

for (size\_t bin:bins)

{

if (max\_count < bin)

{

max\_count = bin;

}

}

double scaling = 1;

if (max\_count > MAX\_ASTERISK)

{

scaling = MAX\_ASTERISK / (double)max\_count;

}

return scaling;

}

size\_t average\_bin (size\_t number\_count, size\_t bin\_count)

{

size\_t avg\_bin = number\_count / bin\_count;

return avg\_bin;

}