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 программного обеспечения систем управления»**

# «Библиотеки, низкоуровневые операции»

Выполнил студент группы А-02-20

Дашин И.Н.

Проверил

Мохов А.С.

Козлюк Д.А.

Василькова П.Д.

Москва 2021

**Цель работы:**

1. Уметь устанавливать и подключать к программе внешние библиотеки.
2. Уметь использовать типовые элементы API: функции обратного вызова, битовые флаги и маски, массивы и строки C.
3. Уметь работать с параметрами командной строки программы.
4. Уметь применять побитовые операции для типовых сценариев.
5. Уметь работать с API, принимающими указатели, в том числе строки C.
6. Знать характерные особенности документации на API библиотек.

# Задание:

# Часть 1. Библиотеки

Добавить возможность построения гистограммы по данным из файла из сети. Адрес файла задается аргументом командной строки программы. Если адрес не задан, читать данные со стандартного ввода, как раньше.

# Часть 2. Низкоуровневые операции

# Код, который будет написан в этой части работы, должен в итоге формировать строку, которую нужно отобразить снизу итогового SVG в формате:

Windows v5.1 (build 1234)

Computer name: My-Comp

**Индивидуальное задание**

#### Вариант 8

С помощью функции curl\_easy\_getinfo() печатайте на стандартный вывод ошибок IP-адрес сервера, с которого скачан файл (не local).

В документации к лабораторной находим функцию вывода IP-адреса и используем ее в программе по примеру:

{

  char \*ip;

  curl\_easy\_setopt(curl, CURLOPT\_URL, "https://example.com");

  /\* Perform the request, res will get the return code \*/

  res = curl\_easy\_perform(curl);

  /\* Check for errors \*/

  if((res == [CURLE\_OK](https://curl.se/libcurl/c/libcurl-errors.html#CURLEOK)) &&

  !curl\_easy\_getinfo(curl, CURLINFO\_PRIMARY\_IP, &ip) && ip) {

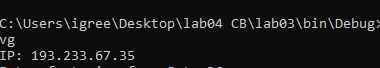
  printf("IP: %s\n", ip);

  }

  /\* always cleanup \*/

  curl\_easy\_cleanup(curl);

}



**Lab03.cpp**

#include <iostream>

#include <vector>

#include "svg.h"

#include "histogram.h"

#include <curl/curl.h>

#include <sstream>

#include <string>

#include <windows.h>

using namespace std;

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

auto data\_size = item\_size \* item\_count;

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

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

return data\_size;

}

Input download(const string& address) {

stringstream buffer;

char \*ip;

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

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

if(curl) {

CURLcode res;

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

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

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

res = curl\_easy\_perform(curl);

if (res != CURLE\_OK) {

cout << address<<endl;

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

exit(1);

}

if((res == CURLE\_OK) && !curl\_easy\_getinfo(curl, CURLINFO\_PRIMARY\_IP, &ip) && ip) {

cerr << "IP: " << ip <<"\n";

}

curl\_easy\_cleanup(curl);

}

return read\_input(buffer, false);

}

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

Input input;

if (argc > 1) {

input = download(argv[1]);

} else {

input = read\_input(cin, true);

}

const auto bins = make\_histogram(input);

show\_histogram\_svg(bins);

make\_info\_text();

return 0;

}

**histogram.cpp**

#include "histogram.h"

#include <vector>

#include <string>

#include <iostream>

using namespace std;

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

vector<double> result(count);

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

in >> result[i];

}

return result;

}

Input read\_input(istream& in, bool prompt) {

Input data;

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

size\_t number\_count;

in >> number\_count;

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

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

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

in >> data.bin\_count;

return data;

}

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

if (numbers.size() == 0) return;

min = numbers[0];

max = numbers[0];

for (double x : numbers) {

if (x < min) {

min = x;

}

if (x > max) {

max = x;

}

}

}

vector<size\_t> make\_histogram(Input data) {

const auto bin\_count = data.bin\_count;

const auto numbers = data.numbers;

double min, max;

size\_t number\_count = numbers.size();

find\_minmax(numbers, min, max);

vector<size\_t> bins(bin\_count);

double bin\_size = (max - min) / bin\_count;

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

bool found = false;

for (size\_t j = 0; (j < bin\_count - 1) && !found; j++) {

auto lo = min + j \* bin\_size;

auto hi = min + (j + 1) \* bin\_size;

if ((lo <= numbers[i]) && (numbers[i] < hi)) {

bins[j]++;

found = true;

}

}

if (!found) {

bins[bin\_count - 1]++;

}

}

return bins;

}

**histogram.h**

#pragma once

#include <iostream>

#include <vector>

using namespace std;

struct Input {

vector<double> numbers;

size\_t bin\_count;

};

vector<double> input\_numbers(istream& in, size\_t count);

Input read\_input(istream& in, bool prompt);

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

vector<size\_t> make\_histogram(Input data);

**svg.cpp**

#include "svg.h"

#include <vector>

#include <sstream>

#include <string>

#include <windows.h>

using namespace std;

string make\_info\_text() {

stringstream buffer;

DWORD WINAPI info = GetVersion();

DWORD mask = 0x0000ffff;

DWORD platform = info >> 16;

DWORD version = info & mask;

DWORD mask2 = 0x00ff;

DWORD version\_major = version >> 8;

DWORD version\_minor = version & mask2;

DWORD build;

if ((info & 0x10000000) == 0) {

build = platform;

}

buffer <<"Windows v"<<version\_minor<<"."<<version\_major<<" (build "<<build <<")\n";

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

DWORD size;

size=sizeof(length);

GetComputerName(length,&size);

buffer <<"Computer name: "<<length;

return buffer.str();

}

int font\_size(istream& in) {

int size;

cerr << "Enter font size from 8 to 36:\n";

in >> size;

bool wrong = true;

while (wrong) {

if (size >= 8 && size <= 36) {

wrong = false;

}

else {

if (size < 8) {

cerr << "Enter again, size is too low\n";

in >> size;

}

if (size > 36) {

cerr << "Enter again, size is too big\n";

in >> size;

}

}

}

return size;

}

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

svg\_end() {

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

}

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

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

}

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 << "' />";

}

void show\_histogram\_svg(const vector<size\_t>& bins) {

const auto IMAGE\_WIDTH = 1000;

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;

const size\_t MAX\_ASTERISK = IMAGE\_WIDTH - TEXT\_WIDTH;

const int size = font\_size(cin);

size\_t max\_count = bins[0];

for (size\_t bin : bins) {

if (bin > max\_count) {

max\_count = bin;

}

}

const bool scaling\_needed = max\_count \* BLOCK\_WIDTH > MAX\_ASTERISK;

double scaling\_factor;

if (scaling\_needed)

{

scaling\_factor = static\_cast<double>(MAX\_ASTERISK) / (max\_count \* BLOCK\_WIDTH);

}

else {

scaling\_factor = 1;

}

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

double top = 50;

for (size\_t bin : bins) {

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

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

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, "black", "#c72af7");

top += BIN\_HEIGHT;

}

string version = make\_info\_text();

svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, version, size);

svg\_end();

}

**svg.h**

#pragma once

#include <iostream>

#include <vector>

#include <string>

#include <windows.h>

#include <sstream>

using namespace std;

void svg\_begin(double width, double height);

void svg\_end();

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

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

void show\_histogram\_svg(const vector<size\_t>& bins);

int font\_size(istream& in);

string make\_info\_text();