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»

# «Библиотеки и низкоуровневые операции»

# *Выполнил:*

# Студент группы А-03-21

# Куракин П.В.

**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.

Работу нужно вести на основе кода общего задания к ЛР № 3 в старом репозитарии в отдельной ветке без создания нового проекта. По этой причине во всех примерах используется lab03.exe.

# 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 6**

С помощью функции curl\_easy\_getinfo() печатайте на стандартный вывод ошибок размер файла, загружаемого по сети.

curl\_off\_t dl;

res = curl\_easy\_getinfo(curl, CURLINFO\_SIZE\_DOWNLOAD\_T, &dl);

if(!res) {

cerr << "Downloaded " << dl << " bytes\n";

}

**Исходный код всех модулей**

#include <iostream>

#include <vector>

#include "histogram.h"

#include "svg.h"

#include <curl/curl.h>

#include <sstream>

#include <string>

#include <windows.h>

using namespace std;

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

vector<double> result(count);

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

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

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;

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

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

size\_t bin\_count;

in >> data.bin\_count;

return data;

}

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

double min ;

double max;

find\_minmax (data, min, max);

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

for (double number : data.numbers) {

size\_t bin = (size\_t)((number - min) / (max - min) \* data.bin\_count);

if (bin == data.bin\_count) {

bin--;

}

bins[bin]++;

}

return bins;

}

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;

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){

cerr << curl\_easy\_strerror(res) << endl;

exit(1);

}

curl\_off\_t dl;

res = curl\_easy\_getinfo(curl, CURLINFO\_SIZE\_DOWNLOAD\_T, &dl);

if(!res) {

cerr << "Downloaded " << dl << " bytes\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);

}

// Обработка данных

const auto bins = make\_histogram(input);

// Вывод данных

show\_histogram\_svg (bins);

return 0;

}

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, bool flag, int n) {

Input data;

if (prompt)

cerr << "Enter column count: ";

size\_t bin\_count;

in >> data.bin\_count;

if (prompt)

cerr << "Enter number count: ";

size\_t number\_count;

if (flag)

in >> number\_count;

else

number\_count=n;

if (prompt)

cerr << "Enter numbers: ";

if (flag)

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

else

{

int A = 0;

int B = 100;

vector<double> result(n);

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

result[i]= A + rand() % ((B + 1) - A);

}

data.numbers=result;

}

return data;

}

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

double max;

double min;

find\_minmax(data, min, max);

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

for (double number : data.numbers) {

size\_t bin = (size\_t)((number - min) / (max - min) \* data.bin\_count);

if (bin == data.bin\_count) {

bin--;

}

bins[bin]++;

}

return bins;

}

void show\_histogram\_text(vector<size\_t> bins){

const size\_t SCREEN\_WIDTH = 80;

const size\_t MAX\_ASTERISK = SCREEN\_WIDTH - 4 - 1;

size\_t max\_count = 0;

for (size\_t count : bins) {

if (count > max\_count) {

max\_count = count;

}

}

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

for (size\_t bin : bins) {

if (bin < 100) {

cout << ' ';

}

if (bin < 10) {

cout << ' ';

}

cout << bin << "|";

size\_t height = bin;

if (scaling\_needed) {

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

height = (size\_t)(bin \* scaling\_factor);

}

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

cout << '\*';

}

cout << '\n';

}

return;

}

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,bool flag, int n) {

stringstream buffer;

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)

{

cerr << curl\_easy\_strerror(res) << endl;

exit(1);

}

}

curl\_easy\_cleanup(curl);

return read\_input(buffer, false, flag,n);

}

int

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

srand(time(NULL));

bool flag=true;

int n=0;

Input input;

if (argc > 1) {

if (strcmp(argv[1], "-generate") == 0 || strcmp(argv[2], "-generate" ) == 0){

if (strcmp(argv[1], "-generate") == 0){

n=atoi(argv[2]);

if (n==0){

cerr << "Enter the number of generated numbers after ' -generate' ";

return 0;

}

flag=false;

input = download(argv[3],flag,n);}

if (strcmp(argv[2], "-generate") == 0){

n=atoi(argv[3]);

if (n==0){

cerr << "Enter the number of generated numbers after ' -generate' ";

return 0;

}

flag=false;

input = download(argv[1],flag,n);}

}

else

input = download(argv[1],true,0);

}

else {

input = read\_input(cin, true, true,n);

}

// Обработка данных

const auto bins = make\_histogram(input);

// Вывод данных

show\_histogram\_svg(bins);

return 0;

}

**histogram.cpp**

#include "histogram.h"

#include <vector>

#include<iostream>

using namespace std;

void find\_minmax(struct Input t, double& min, double& max) {

if (t.numbers.size()>0){

min = t.numbers[0];

max = t.numbers[0];

for (double number : t.numbers) {

if (number < min) {

min = number;

}

if (number > max) {

max = number;

}

}}

}

**histogram.h**

#ifndef HISTOGRAM\_H\_INCLUDED

#define HISTOGRAM\_H\_INCLUDED

#include <vector>

using namespace std;

struct Input {

vector<double> numbers;

size\_t bin\_count;

};

void find\_minmax(struct Input t, double& min, double& max);

vector <size\_t> make\_histogram (vector<double> numbers, size\_t bin\_count);

vector <double> input\_numbers(size\_t count);

#endif // HISTOGRAM\_H\_INCLUDED

**svg.cpp**

#include <vector>

#include <iostream>

#include "svg.h"

#include <string>

#include <sstream>

#include <windows.h>

using namespace std;

string version\_info() {

stringstream buffer;

DWORD info = GetVersion();

DWORD mask = 0x0000ffff;

DWORD version = info & mask;

DWORD platform = info >> 16;

DWORD mask\_2 = 0x0000ff;

if ((info & 0x80000000) == 0)

{

DWORD version\_major = version & mask\_2;

DWORD version\_minor = version >> 8;

DWORD build = platform;

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

}

char computer\_name[MAX\_COMPUTERNAME\_LENGTH + 1];

DWORD size = MAX\_COMPUTERNAME\_LENGTH+1;

GetComputerNameA(computer\_name, &size);

buffer << "Computer name:" << computer\_name;

return buffer.str();

}

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) {

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

}

string color\_ask (istream& stream){

string color;

cerr << "Color - ? :";

stream>>color;

return (color);

}

void show\_histogram\_text (vector<size\_t> bins) {

const size\_t SCREEN\_WIDTH = 80;

const size\_t MAX\_ASTERISK = SCREEN\_WIDTH - 4 - 1;

size\_t max\_count = 0;

for (size\_t count : bins) {

if (count > max\_count) {

max\_count = count;

}

}

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

for (size\_t bin : bins) {

if (bin < 100) {

cout << ' ';

}

if (bin < 10) {

cout << ' ';

}

cout << bin << "|";

size\_t height = bin;

if (scaling\_needed) {

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

height = (size\_t)(bin \* scaling\_factor);

}

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

cout << '\*';

}

cout << '\n';

} }

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

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;

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

for (size\_t count : bins)

{

if ( count >max\_count)

{

max\_count = count;

}

}

const bool scaling\_needed = max\_count > (IMAGE\_WIDTH/BLOCK\_WIDTH-TEXT\_WIDTH/BLOCK\_WIDTH);

double scaling\_factor=1;

if (scaling\_needed){

scaling\_factor =(double)(IMAGE\_WIDTH - TEXT\_WIDTH)/(max\_count\*BLOCK\_WIDTH);

}

size\_t i=0;

for (size\_t bin : bins) {

//auto color = color\_ask();

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

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

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

top += BIN\_HEIGHT;

}

svg\_text(TEXT\_LEFT, top+BIN\_HEIGHT, version\_info());

svg\_end();

}

**svg.h**

#ifndef SVG\_H\_INCLUDED

#define SVG\_H\_INCLUDED

#include <iostream>

#include <vector>

using namespace std;

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

void svg\_end();

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

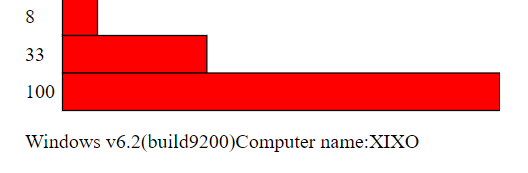
void svg\_rect(double x, double y, double width, double height, string stroke="black", string FILL="green");

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

string version\_info();

string color\_ask (istream& stream = cin);

#endif // SVG\_H\_INCLUDED

**Итоговый вид программы   
**