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 и низкоуровневые операции»

Выполнила студентка группы А-02-20

Воеводова Надежда Андреевна

Проверил

Мохов А.С.

Козлюк Д.А.

Василькова П.Д.

Москва 2021

Ссылка на репозитарий: https://github.com/VoevodovaNA/lab03

**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 7**

При помощи CURLOPT\_PROGRESSFUNCTION добавьте в программу отображение прогресса загрузки файла. Каждый раз, когда cURL рапортует прогресс, печатайте строку вида: Progress: 42%.

**Указание.** Чтобы cURL рапортовала прогресс, нужно установить CURLOPT\_NOPROGRESS в 0.

**Логика решения варианта:**

Для решения задачи я написала функцию progress\_callback , в которой рассчитывается процент прогресса, для этого я умножила 100 на загруженные данные и поделила на общее количество. Далее использовала функцию CURLOPT\_PROGRESSFUNCTION, пример использования функции посмотрела в интернете и вставила в функцию Input download.

**Исходный код всех модулей**

**main.cpp**

#include <iostream>

#include <vector>

#include <curl/curl.h>

#include <sstream>

#include <string>

#include <windows.h>

#include "histogram.h"

#include "svg.h"

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 promt) {

Input data;

if(promt)

cerr << "Enter number count: ";

size\_t number\_count;

in >> number\_count;

if(promt)

cerr << "Enter numbers: ";

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

if(promt)

cerr << "Enter column count: ";

in >> data.bin\_count;

return data;

}

size\_t

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

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;

}

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

double result;

result=100\*dlnow/dltotal;

if (dltotal=0) {

cerr << "0%" << endl;

}

cerr << "progress: " << result << "%" << endl;

return CURL\_PROGRESSFUNC\_CONTINUE;

}

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\_WRITEFUNCTION, write\_data);

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

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

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

res = curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

if(res)

{

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

exit(1);

}

}

return read\_input(buffer, false);

}

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

{

Input input;

cerr << argc;

if (argc > 1)

{

input = download(argv[1]);

}

else

{

input = read\_input(cin, true);

}

const auto bins = make\_histogram(input);

show\_histogram\_svg(bins);

return 0;

}

**histogram.cpp**

#include "histogram.h"

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

{

if (numbers.size() == 0)

return;

else

{

min = numbers[0];

max = numbers[0];

for (double number : numbers)

{

if (number < min)

{

min = number;

}

if (number > max)

{

max = number;

}

}

}

}

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

if (bins.size()==0)

return 0;

double result;

double sum=0;

for (size\_t i = 0; i<bins.size(); i++)

{

sum = sum + (double)(bins[i]);

}

result=sum/((double)(bins.size()));

return result;

}

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

{

double min, max;

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

vector<size\_t> result(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--;

}

result[bin]++;

}

return result;

}

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

{

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';

}

}

**histogram.h**

#ifndef HISTOGRAM\_H\_INCLUDED

#define HISTOGRAM\_H\_INCLUDED

#include <vector>

#include <iostream>

#include <math.h>

#include <string>

using namespace std;

const size\_t SCREEN\_WIDTH = 80;

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

struct Input {

vector<double> numbers;

size\_t bin\_count;

string stroke;

string fill;

};

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

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

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

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

#endif // HISTOGRAM\_H\_INCLUDED

**svg.cpp**

#include "svg.h"

#include "histogram.h"

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

make\_info\_text() {

stringstream buffer;

DWORD info = GetVersion();

DWORD mask = 0x0000ffff;

DWORD mask\_2 = 0x000000ff;

DWORD platform = info >> 16;

DWORD version = info & mask;

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 << ") \n ";

}

DWORD size = MAX\_COMPUTERNAME\_LENGTH+1;

char computer\_name[size];

GetComputerNameA(computer\_name, &size);

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

return buffer.str();

}

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

{

const auto IMAGE\_WIDTH = 500;

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 SCREEN\_WIDTH = 80;

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

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

double top = 0;

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;

double sr = mean(bins);

if (scaling\_needed)

{

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

for (size\_t bin : bins)

{

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

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

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

if ((double)bin <= sr)

{

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, "green",

"#ffeeee");

}

else

{

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

"#ffeeee");

}

top += BIN\_HEIGHT;

}

}

else

{

for (size\_t bin : bins)

{

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

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

if ((double)bin <= sr)

{

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, "green",

"#ffeeee");

}

else

{

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

"#ffeeee");

}

top += BIN\_HEIGHT;

}

}

svg\_text(0, top + TEXT\_BASELINE, make\_info\_text());

svg\_end();

}

**svg.h**

#ifndef SVG\_H\_INCLUDED

#define SVG\_H\_INCLUDED

#include <vector>

#include <iostream>

#include <sstream>

#include <windows.h>

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);

string make\_info\_text();

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

#endif // SVG\_H\_INCLUDED