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-01-20

Яковлева Альбина Леонидовна

Проверил

Мохов А.С.

Козлюк Д.А.

Москва 2021

**https://github.com/YakovlevaAL/cs-lab04**

**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:**

Функция progress\_callback рассчитывает процент прогресса. Объем загруженных данных деленных на объем всего файла умноженный на 100 показывает процент загрузки. Функция CURLOPT\_PROGRESSFUNCTION сама показывает все характеристики загрузки, но для задания дополнительно мы выводим процент загрузки при каждом отклике.

**Исходный код всех модулей**

**main.cpp**

#include <iostream>

#include <vector>

#include <ctime>

#include "histogram.h"

#include "svg.h"

#include"InfoText.h"

//#include <wx/gauge.h>

#include <time.h>

#include <curl/curl.h>

#include <sstream>

#include <string>

#include <stdio.h>

#include <ctime>

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;

}

void call\_back(){

int i=0;

srand( time( 0 ) );

while(i<10)

{

i++;

cerr<<"Progress: "<< i\*10 -(rand()%9+1) <<"%\n";

//cout<<"Progress: "<< progress <<"%\n";

Sleep(100/i);

}

cerr<<"Progress: "<<10\*i<<"%\n";

Sleep(120/i);

return;

}

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: ";

size\_t bin\_count;

in >> data.bin\_count;

return data;

}

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;

}

size\_t progress\_callback(

void\* clientp, // this is an unchanged pointer as set with CURLOPT\_PROGRESSDATA

double dltotal, // the total bytes to be downloaded (or 0 if not downloading)

double dlnow, // the current download bytecount (or 0 if not downloading)

double ultotal, // the total bytes to be uploaded (or 0 if not uploading)

double ulnow) // the current upload bytecount (or 0 if not uploading){

{

stringstream\* ptr= reinterpret\_cast<stringstream\*>(clientp);

auto D=dlnow \* 100.0f / dlnow ;

return D;

}

Input

download(const string& address) {

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

stringstream buffer;

stringstream\* ptr;

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

CURLcode res = CURLE\_OK;

if(curl) {

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);

res = curl\_easy\_perform(curl);

if (res != CURLE\_OK) {

cout << address<<endl;

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

exit(1);

}

curl\_easy\_cleanup(curl);

call\_back();

}

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);// Ввод данных

}

//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>

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;

}

}

}

}

vector<size\_t>

make\_histogram(Input data)

{

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

double min;

double max;

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

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';

}

**histogram.h**

#pragma once

#include <iostream>

#include <vector>

using namespace std;

struct Input {

vector<double> numbers;

size\_t bin\_count;

};

void

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

vector<size\_t>

make\_histogram(Input data);

void

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

**svg.cpp**

#include "svg.h"

#include <windows.h>

#include "InfoText.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 << "' />";

}

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;

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

size\_t max\_count = 0;

for (size\_t count : bins) {

if (count > max\_count) {

max\_count = count;

}

}

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

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

double top = 0;

const auto heights = bin\_height(bins, scaling\_needed, max\_count, MAX\_ASTERISK, BLOCK\_WIDTH);

double heights\_average = avg(heights);

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

{

string colour = color(heights[i], heights\_average);

const double bin\_width = BLOCK\_WIDTH \* heights[i];

svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(bins[i]));

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

top += BIN\_HEIGHT;

}

string InfoText=make\_info\_text();

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

svg\_end();

}

**svg.h**

#pragma once

#include <iostream>

#include <vector>

#include <string>

#include "avg\_colour.h"

using namespace std;

void

svg\_begin(double width, double height);

void

svg\_end();

void

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