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 № 3**

По курсу «Разработка ПО систем управления»

«Основы языка С++»

Выполнил студент группы А-02-19

Сон А.Е.

Проверили:

Мохов А. С

Козлюк Д. А

Москва 2020

<https://github.com/SonAE1/cs.lab03>

**Цель работы:**

**Часть 1:**

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.

**Часть 2:**

1.Уметь применять побитовые операции для типовых сценариев.

2.Уметь работать с API, принимающими указатели, в том числе строки C.

3.Знать характерные особенности документации на API библиотек.

**Задание**

**Часть 1:**

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.

Пример строки запуска:

lab03.exe http://uii.mpei.ru/study/courses/cs/lab03/marks.txt >marks.svg

**Часть 2:**

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

**Вариант 15**

Установите callback-функцию CURLOPT\_HEADERDATA, откуда печатайте полученные данные на стандартный вывод ошибок.

**Описание логики решения варианта:**

<https://curl.haxx.se/libcurl/c/CURLOPT_HEADERFUNCTION.html>

Используя информацию с этого сайта по аналогии с пунктом

**3.2.9 Сохранение данных из сети в буфер** написана функция обратного вызова **header\_callback**, где далее она вызывается в функции curl\_easy\_setopt(). Так же было необходимо создать второй буфер.

**Main.cpp**

#include <curl/curl.h>

#include "histogram.h"

#include <iostream>

#include "svg.h"

#include <vector>

#include <cmath>

#include <sstream>

#include <string>

#include <windows.h>

#include <stdio.h>

using namespace std;

const size\_t SCREEN\_WIDTH = 500;

const size\_t MAX\_ASTERISK = SCREEN\_WIDTH - 3 - 1;

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;

size\_t number\_count;

if( prompt )

{

cerr << "Enter number count: ";

in >> number\_count;

cerr << "Enter numbers: ";

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

cerr << "Enter bin count:";

in >>data.bin\_count;

cerr << "Enter number of scale:";

in >> data.scale;

}

else

{

in >> number\_count;

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

in >>data.bin\_count;

in >> data.scale;

}

return data;

}

vector<size\_t>

make\_histogram(Input data)

{

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

double min;

double max;

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

for(double x: data.numbers)

{

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

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

{

bin\_index--;

}

bins[bin\_index]++;

}

return bins;

}

size\_t

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

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

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

const char\* items2 = reinterpret\_cast<const char\*>(items);

buffer->write(items2, data\_size);

return data\_size ;

}

size\_t header\_callback(void\* items, size\_t item\_size, size\_t item\_count, void\* ctx)

{

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

stringstream\* buffer\_1 = reinterpret\_cast<stringstream\*>(ctx);

buffer\_1->write(reinterpret\_cast<const char\*>(items), data\_size);

return data\_size;

}

Input

download(const string& address) {

stringstream buffer;

stringstream buffer\_1;

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\_HEADERFUNCTION, header\_callback);

curl\_easy\_setopt(curl, CURLOPT\_HEADERDATA, &buffer\_1);

res = curl\_easy\_perform(curl);

string str;

while (buffer\_1) {

buffer\_1 >> str;

cerr << str << " ";

}

if(res!= CURLE\_OK)

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

exit(1);

}

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, input.scale);

}

**Histogram.h**

#ifndef HISTOGRAM\_H\_INCLUDED

#define HISTOGRAM\_H\_INCLUDED

#include <vector>

using namespace std;

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

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

struct Input

{

vector<double> numbers;

size\_t bin\_count;

size\_t scale;

};

#endif // HISTOGRAM\_H\_INCLUDED

**Histogram.cpp**

#include "histogram.h"

#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 x: numbers)

{

if(min>x)

min=x;

if(max<x)

max=x;

}

}

}

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

{

const size\_t SCREEN\_WIDTH = 800;

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 need = max\_count > MAX\_ASTERISK;

for (size\_t bin : bins)

{

if (bin < 100)

{

cout << ' ';

}

if (bin < 10)

{

cout << ' ';

}

cout << bin << "|";

size\_t height = bin;

if (need)

{

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';

}

}

**Svg.h**

#ifndef SVG\_H\_INCLUDED

#define SVG\_H\_INCLUDED

#include <vector>

#include <cmath>

#include <iostream>

#include <windows.h>

#include <stdio.h>

using namespace std;

DWORD WINAPI GetVersion(void);

string

make\_info\_text();

string

make\_info\_text1();

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

void svg\_end();

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

void svg\_scale(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, size\_t scale0);

void scale (const vector<size\_t>& bins, size\_t scale0,size\_t& interval);

#endif // SVG\_H\_INCLUDED

**Svg.cpp**

#include "svg.h"

#include <iostream>

#include <vector>

#include <cmath>

#include "histogram.h"

#include <sstream>

#include <windows.h>

#include <stdio.h>

using namespace std;

DWORD WINAPI GetVersion(void);

string

make\_info\_text()

{

stringstream buffer;

DWORD info = GetVersion();

DWORD mask\_major = 0b00000000'00000000'00000000'11111111;

DWORD mask = 0x0000ffff;

DWORD version = info & mask;

DWORD platform = info>>16;

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

{

DWORD version\_major = version & mask\_major;

DWORD version\_minor = version >> 8;

DWORD build = platform;

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

}

return buffer.str();

}

string

make\_info\_text1()

{

stringstream buffer;

char system\_dir[MAX\_PATH];

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

DWORD size = sizeof(comp\_name);

GetComputerNameA(comp\_name, &size);

buffer << "Computer Name:" << comp\_name;

return buffer.str();

}

const size\_t SCREEN\_WIDTH = 800;

const size\_t MAX\_ASTERISK = SCREEN\_WIDTH - 3 - 1;

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\_rect(double x, double y, double width, double height, string stroke = "black", string fill = "black")

{

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

}

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

{

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

}

void scale (const vector<size\_t>& bins, size\_t scale0, size\_t& interval)

{

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

for (size\_t bin : bins)

{

if (max\_bin<bin)

max\_bin=bin;

}

if (bins.size() != 0)

{

if (max\_bin%scale0!=0)

interval =max\_bin/scale0+1;

else

interval =max\_bin/scale0;

}

}

void show\_histogram\_svg(const vector<size\_t>& bins, size\_t scale0)

{

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\_WIDTH - TEXT\_LEFT;

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 need = max\_count\* BLOCK\_WIDTH > MAX\_ASTERISK;

const double scaling\_factor1 = (double)MAX\_ASTERISK / max\_count;

size\_t interval;

if (scale0>9 || scale0<2)

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

else

{

for (size\_t bin : bins)

{

size\_t height = bin;

if (need)

{

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

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

}

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

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

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT,"#000000","#A52A2A");

top += BIN\_HEIGHT;

}

scale ( bins, scale0, interval);

svg\_text (TEXT\_WIDTH-2,top + TEXT\_BASELINE,"|");

for(int i=1; i<=interval\*scale0; i++)

{

if (i%scale0==0)

svg\_text(TEXT\_WIDTH-2+i\*scaling\_factor1,top +TEXT\_BASELINE,"|");

else

svg\_text(TEXT\_WIDTH-2+i\*scaling\_factor1,top+ TEXT\_BASELINE,"-");

}

svg\_text(TEXT\_WIDTH,top+ 2\*TEXT\_BASELINE, to\_string(0));

for(int i=1; i<interval\*scale0; i++)

{

if ((i%scale0!=0) || ((i % scale0== 0) && ((i > scale0))))

svg\_text(TEXT\_WIDTH-2+i\*scaling\_factor1,top+ 2\*TEXT\_BASELINE," ");

else

svg\_text(TEXT\_WIDTH-2+i\*scaling\_factor1,top+ 2\*TEXT\_BASELINE, to\_string(i));

}

svg\_text(TEXT\_WIDTH-5+interval \* scale0\*scaling\_factor1,top+ 2\*TEXT\_BASELINE, to\_string(interval \* scale0) );

}

svg\_text(1,top+ 3\*TEXT\_BASELINE, make\_info\_text());

svg\_text(1,top+ 4\*TEXT\_BASELINE, make\_info\_text1());

}