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**

**«Библиотеки и низкоуровневые операции»**

# По курсу «Разработка ПО систем управления»

# «Основы языка С++»

# Выполнил студент группы А-03-19

# Симич А.Б.

Москва 2020

1)Ссылка на репозитарий: <https://github.com/Alexandr-Simich/lab-04>  
  
2) Постановка задачи (Вариант 16) :

С помощью функции curl\_easy\_getinfo() печатайте на стандартный вывод ошибок время, затраченное на установку соединения с сервером (connect).

3) Логика решения варианта:

Используя функцию curl\_easy\_getinfo(), вывожу время затраченное на установку соединения с сервером.

4) Исходный код всех модулей:

**main.cpp**

#include "histogram.h"

#include "svg.h"

using namespace std;

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

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

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

Input download(const string& address);

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

{

Input input;

if (argc > 1)

{

input = download(argv[1]);

auto colors = input\_colors(input.bin\_count, false);

const auto bins = make\_histogram(input);

show\_histogram\_svg(bins, input.bin\_count, false, argv[1], colors);

}

else

{

input = read\_input(cin, true);

const auto bins = make\_histogram(input);

auto colors = input\_colors(input.bin\_count, true);

show\_histogram\_svg(bins, input.bin\_count, true, "123", colors);

}

return 0;

}

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 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;

}

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);

res = curl\_easy\_perform(curl);

if (res)

{

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

exit(1);

}

}

curl\_easy\_cleanup(curl);

return read\_input(buffer, false);

}

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;

}

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

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

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

**histogram.cpp**

#include "histogram.h"

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

{

vector<double> result(count);

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

{

cin >> result[i];

}

return result;

}

vector <size\_t> make\_histogram(Input input)

{

double min, max;

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

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

for (double number : input.numbers)

{

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

if (bin == input.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";

}

}

**Svg.h**

#pragma once

#include <iostream>

#include <vector>

#include <windows.h>

#include <curl/curl.h>

#include <sstream>

#include <string>

using namespace std;

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

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

void svg\_rect(double x, double y, double wid, double heig, string stroke, string fill);

void svg\_end();

void show\_histogram\_svg(const vector<size\_t>& bins, size\_t bin\_count, bool flag, const string& address, vector<string>& colors);

vector <string> input\_colors(size\_t bin\_count, bool flag);

**svg.cpp**

#include "svg.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\_text(double left, double baseline, string text)

{

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

}

void svg\_rect(double x, double y, double wid, double heig, string stroke, string fill = "white")

{

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

};

void svg\_end()

{

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

}

void show\_histogram\_svg(const vector<size\_t>& bins, size\_t bin\_count, bool flag, const string& address, vector<string>& colors)

{

DWORD info = GetVersion();

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

DWORD version = info & mask;

mask = 0x000000ff;

DWORD platform = info >> 16;

DWORD version\_major = version & mask;

DWORD version\_minor = version >> 8;

DWORD build;

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

{

build = platform;

// printf("Windows v%u.%u (build %u)\n", version\_major, version\_minor, build);

}

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

DWORD size = sizeof(computer\_name);

GetComputerNameA(computer\_name, &size);

// printf("Computer name: %s\n", computer\_name);

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);

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) > (IMAGE\_WIDTH - TEXT\_WIDTH);

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

if (scaling\_needed)

{

double top = 0;

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

{

const double bin\_width = double(BLOCK\_WIDTH \* bins[i] \* scaling\_factor);

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

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, colors[i]);

top += BIN\_HEIGHT;

}

if(flag == true)

{

cout << "<text x='" << TEXT\_LEFT << "' y='" << top + TEXT\_BASELINE << "'>" << "Windows v" << version\_major << "." << version\_minor << " (build " << build << ")" << "</text>" << endl;

cout << "<text x='" << TEXT\_LEFT << "' y='" << top + 20 + TEXT\_BASELINE << "'>" << "Computer name: " << computer\_name << "</text>" << endl;

}

else

{

cout << "<text x='" << TEXT\_LEFT << "' y='" << top + TEXT\_BASELINE << "'>" << "Windows v" << version\_major << "." << version\_minor << " (build " << build << ")" << "</text>" << endl;

cout << "<text x='" << TEXT\_LEFT << "' y='" << top + 20 + TEXT\_BASELINE << "'>" << "Computer name: " << computer\_name << "</text>" << endl;

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

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

if(curl)

{

CURLcode res;

double connect;

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

res = curl\_easy\_perform(curl);

if(CURLE\_OK == res)

{

res = curl\_easy\_getinfo(curl, CURLINFO\_CONNECT\_TIME, &connect);

if(CURLE\_OK == res)

{

// printf("Time: %.1f", connect);

cout << "<text x='" << TEXT\_LEFT << "' y='" << top + 50 + TEXT\_BASELINE << "'>" << "Time: " << connect << "</text>" << endl;

}

}

curl\_easy\_cleanup(curl);

}

}

}

else

{

double top = 0;

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

{

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

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

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, colors[i]);

top += BIN\_HEIGHT;

}

if(flag == true)

{

cout << "<text x='" << TEXT\_LEFT << "' y='" << top + TEXT\_BASELINE << "'>" << "Windows v" << version\_major << "." << version\_minor << " (build " << build << ")" << "</text>" << endl;

cout << "<text x='" << TEXT\_LEFT << "' y='" << top + 20 + TEXT\_BASELINE << "'>" << "Computer name: " << computer\_name << "</text>" << endl;

}

else

{

cout << "<text x='" << TEXT\_LEFT << "' y='" << top + TEXT\_BASELINE << "'>" << "Windows v" << version\_major << "." << version\_minor << " (build " << build << ")" << "</text>" << endl;

cout << "<text x='" << TEXT\_LEFT << "' y='" << top + 20 + TEXT\_BASELINE << "'>" << "Computer name: " << computer\_name << "</text>" << endl;

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

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

if(curl)

{

CURLcode res;

double connect;

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

res = curl\_easy\_perform(curl);

if(CURLE\_OK == res)

{

res = curl\_easy\_getinfo(curl, CURLINFO\_CONNECT\_TIME, &connect);

if(CURLE\_OK == res)

{

// printf("Time: %.1f", connect);

cout << "<text x='" << TEXT\_LEFT << "' y='" << top + 50 + TEXT\_BASELINE << "'>" << "Time: " << connect << "</text>" << endl;

}

}

curl\_easy\_cleanup(curl);

}

}

}

svg\_end();

}

vector <string> input\_colors(size\_t bin\_count, bool flag)

{

vector<string> colors(bin\_count);

if(flag)

{

bool flag2;

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

{

do

{

cin >> colors[i];

bool flag1 = true;

for (auto s : colors[i])

{

if (s == ' ')

{

flag1 = false;

}

}

if(colors[i][0] == '#')

{

if (flag1 == true)

{

flag2 = true;

}

else

{

flag2 = false;

cerr << "Error1";

}

}

else if(colors[i][0] != '#')

{

if(flag1 = true)

{

flag2 = true;

}

else

{

flag2 = false;

cerr << "Error2";

}

}

}

while(flag2 == false);

}

}

else

{

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

{

colors[i] = "red";

}

}

return colors;

}