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-03-19

Соколов И.А.

Проверили

Москва 2020

**Вариант 18**

С помощью функции curl\_easy\_getinfo() печатайте на стандартный вывод ошибок время, затраченное на получение IP-адреса сервера по его имени (name lookup).

**Ход решения**

В функции download вызывается функция библиотеки curl curl\_easy\_getinfo(), в которую мы передаём параметры, которые указывает функции на данные, которые хотим получить. В нашем случае это CURLINFO\_NAMELOOKUP\_TIME. Также в функцию передаём переменную namelookup, в которую и будет записано временя. В случае успешного получения времени, выводим его.

Информация о функции curl\_easy\_getinfo():

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

**Ссылка на репозитарий**

<https://github.com/SokolovIgA/lab3-4>

**Код программы**

**main.cpp**

#include <iostream>

#include <vector>

#include <sstream>

#include <string>

#include <curl/curl.h>

#include <cstdio>

#include "histogram.h"

#include "svg.h"

using namespace std;

vector<double> input\_numbers(istream& in, const 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 column count: ";

in >> data.bin\_count;

}

else

{

in >> number\_count;

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

in >> data.bin\_count;

}

return data;

}

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;

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

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

buffer->write(new\_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 != CURLE\_OK)

{

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

exit(1);

}

else

{

double namelookup\_time;

res = curl\_easy\_getinfo(curl, CURLINFO\_NAMELOOKUP\_TIME, &namelookup\_time);

if (res == CURLE\_OK)

{

cerr << "Name lookup time: " << namelookup\_time << endl;

}

}

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);

DWORD mask = 0x0000ffff;

DWORD mask\_major = 0x000000f;

DWORD info = GetVersion();

DWORD platform = info >> 16;

DWORD version = info & mask;

DWORD version\_major = version & mask\_major;

DWORD version\_minor = version >>8;

DWORD build;

if ((info & 0x40000000) == 0);

{

build = platform;

}

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

DWORD size = sizeof(computer\_name);

GetComputerName(computer\_name, &size);

// Вывод данных

show\_histogram\_svg(bins, version\_major, version\_minor, build, computer\_name);

return 0;

}

**histogram.h**

#ifndef HISTOGRAM\_H\_INCLUDED

#define HISTOGRAM\_H\_INCLUDED

#include<vector>

struct Input {

std::vector<double> numbers;

size\_t bin\_count;

};

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

std::vector<size\_t> make\_histogram(Input data);

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

#endif

**histogram.cpp**

#include<vector>

#include <iostream>

#include"histogram.h"

using namespace std;

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> result(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--;

}

result[bin]++;

}

return result;

}

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**

#ifndef SVG\_H\_INCLUDED

#define SVG\_H\_INCLUDED

#include <vector>

#include <string>

#include <windows.h>

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

void svg\_end();

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

void svg\_rect(double x, double y, double width, double height, std::string stroke = "black", std::string fill = "black");

void show\_histogram\_svg(const std::vector<size\_t>& bins, DWORD version\_major, DWORD version\_minor, DWORD build, char computer\_name[]);

size\_t find\_min(const std::vector<size\_t>& bins);

size\_t find\_max(const std::vector<size\_t>& bins);

#endif // SVG\_H\_INCLUDED

**svg.cpp**

#include "svg.h"

#include <vector>

#include <string>

#include <iostream>

using namespace std;

string text\_decoration18(size\_t bin)

{

size\_t k=-1;

while ((k<0) or (k>3))

{

cerr<<endl;

cerr<<"Now choose the text of "<<bin+1<<" column"<<endl;

cerr<<"0 - none"<<endl;

cerr<<"1 - underline"<<endl;

cerr<<"2 - overline"<<endl;

cerr<<"3 - line-through"<<endl;

cerr<<"If you enter another number you will have one more try!"<<endl;

cin>>k;

}

if (k==0)

return "none";

if (k==1)

return "underline";

if (k==2)

return "overline";

if (k==3)

return "line-through";

}

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, size\_t text, string text\_decoration = "none")

{

cout << "<text x='" << left << "' y='"<<baseline<<"' text-decoration='"<<text\_decoration<<"'>"<<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, DWORD version\_major, DWORD version\_minor, DWORD build, char 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;

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;

size\_t i = 0;

for (size\_t bin : bins)

{

size\_t height = bin;

if (scaling\_needed)

{

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, bin,text\_decoration18(i));

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

top += BIN\_HEIGHT;

i++;

}

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

cout << "<text x='" << left << "' y='" << top + 2\*TEXT\_BASELINE << "'>Windows v" << version\_major << "."

<< version\_minor << " (build " << build << ")</text>";

svg\_end();

}