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**

**«Декомпозиция и контроль корректности программ»**

# По курсу «Разработка ПО систем управления»

# «Основы языка С++»

# Выполнил студент группы А-03-19

# Мирошников А. М.

# Проверили

# Мохов А. С

# Козлюк Д. А

Москва 2020

ссылка на репозитарий: <https://github.com/MiroshnikovAM/cs-lab03>  
  
2) Постановка задачи (Вариант 10) :

Отображайте гистограмму вертикально, с подписями сверху, по аналогии с заданием этого варианта в лабораторной работе 1. Предусмотреть расчет IMAGE\_HEIGHT таким образом, чтобы вся гистограмма вмещалась в область рисунка.

3) Логика решения варианта:

Для вертикального отображения гистограммы необходимо её "транспонировать" и масштабировать.

Для этого код подвергся небольшому рефакторингу с изменением номенклатуры.

Если же раньше мы меняли height, то теперь меняем в цикле width.

Вместо курсор-переменной top введена left.

Дабы избежать изменений в подпрограммах, изменен порядок подачи аргументов при вызове

функций svg\_text и svg\_rect.

Для масштабирования:

Убраны устаревшие элементы SCREEN\_WIDTH и сопутствующие проверки scaling\_needed.

Теперь масштабирование производится всегда, причем "корзина" с максимальным количеством

элементов всегда занимает всю высоту картинки, а остальные корзины масштабируются под неё.

Добавлено несколько косметических изменений в код констант.

Для тестирования :

Согласно текста задания, цитирую:

"должно быть минимум 2 теста, проверяющих работу вашего индивидуального задания."

т.е. в моём случае индивидуальное задание касается исключительно svg-модуля, как и тесты.

svg-модуль формирует по сути текстовый документ, и для теста функций пришлось их переработать.

Если раньше svg-функции были типа void, и выводили всё в cout, то теперь они string.

Формируемые таким образом строки конкатенируются и итоговая строка подаётся в поток вывода,

формируя непосредственно текст svg-файла.

Такой подход позволяет проверять данные, которые возвращают svg-функции.

Написанные тесты проверяют случаи с разным количеством "попугаев" в корзинах-bins

на сходство с ожидаемым выводом.

Данный подход позволяет написать и более глубокие тесты при помощи парсеров, но, полагаю, это

несколько выходит за рамки данной лабораторной работы.

4) Исходный код всех модулей:

============

= main.cpp =

============

#include <iostream>

#include <vector>

#include "histogram.h"

#include "svg\_module.h"

using namespace std;

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

int main() {

size\_t number\_count;

cerr << "Enter number count: ";

cin >> number\_count;

if (number\_count == 0) {

cerr << "ERROR: Empty vector";

} else {

cerr << "Enter numbers: ";

const auto numbers = input\_numbers(number\_count);

size\_t bin\_count;

cerr << "Enter column count: ";

cin >> bin\_count;

double min, max;

find\_minmax(numbers, min, max);

const auto bins = make\_histogram(numbers, bin\_count, min, max);

cout << show\_histogram\_svg(bins);

}

return 0;

}

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

vector<double> result(count);

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

cin >> result[i];

}

return result;

}

===============

= histogram.h =

===============

#pragma once

#include <vector>

using namespace std;

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

vector<size\_t> make\_histogram(const vector<double> & numbers, size\_t bin\_count, double min, double max);

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

=================

= histogram.cpp =

=================

#include <vector>

#include <iostream>

using namespace std;

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

if (numbers.size()) {

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(const vector<double> & numbers, size\_t bin\_count, double min, double max) {

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

for (double number : numbers) {

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

if (bin == bin\_count) {

bin--;

}

bins[bin]++;

}

return(bins);

}

void show\_histogram\_text(const 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\_module.h =

================

#pragma once

#include <vector>

#include <string>

using namespace std;

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

string svg\_end();

string svg\_rect(double x, double y, double width, double height, string stroke, string fill);

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

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

==================

= svg\_module.cpp =

==================

#include <iostream>

#include <vector>

#include <sstream>

#include <string>

using namespace std;

string svg\_begin(double width, double height) {

ostringstream strs;

strs << "<?xml version='1.0' encoding='UTF-8'?>\n";

strs << "<svg ";

strs << "width='" << width << "' ";

strs << "height='" << height << "' ";

strs << "viewBox='0 0 " << width << " " << height << "' ";

strs << "xmlns='http://www.w3.org/2000/svg'>\n";

string output = strs.str();

return output;

}

string svg\_end() {

ostringstream strs;

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

string output = strs.str();

return output;

}

string svg\_rect(double x, double y, double width, double height, string stroke, string fill) {

ostringstream strs;

strs << "\t";

strs << "<rect x='" << x << "' y='" << y

<< "' width='" << width << "' height='" << height

<< "' stroke='" << stroke << "' fill='" << fill

<< "' />";

strs << endl;

string output = strs.str();

return output;

}

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

ostringstream strs;

strs << "\t";

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

strs << endl;

string output = strs.str();

return output;

}

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

const auto IMAGE\_WIDTH = 300;

const auto IMAGE\_HEIGHT = 300;

const auto TEXT\_TOP = 20;

const auto TEXT\_HEIGHT = 30;

const auto BIN\_WIDTH = 30;

const auto TEXT\_BASELINE = BIN\_WIDTH / 2;

string output;

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

double left = 0;

string stroke = "black";

string fill = "blue";

size\_t max\_count = 0;

for (size\_t count : bins) {

if (count > max\_count) {

max\_count = count;

}

}

for (size\_t bin : bins) {

const double scaling\_factor = (double)(IMAGE\_HEIGHT - TEXT\_HEIGHT) / max\_count;

const double bin\_height = bin \* scaling\_factor;

output += svg\_text(left + TEXT\_BASELINE, TEXT\_TOP, to\_string(bin));

output += svg\_rect(left, TEXT\_HEIGHT, BIN\_WIDTH, bin\_height, stroke, fill);

left += BIN\_WIDTH;

}

output += svg\_end();

return output;

}

============

= test.cpp =

============

#include "histogram.h"

#include "svg\_module.h"

#include <cassert>

#include <string>

void test\_positive() {

double min = 0;

double max = 0;

find\_minmax({1, 2, 3}, min, max);

assert(min == 1);

assert(max == 3);

}

void test\_negative() {

double min = 0;

double max = 0;

find\_minmax({-1, -2, -3}, min, max);

assert(min == -3);

assert(max == -1);

}

void test\_equal() {

double min = 0;

double max = 0;

find\_minmax({1, 1, 1}, min, max);

assert(min == 1);

assert(max == 1);

}

void test\_empty() {

double min = 0;

double max = 0;

find\_minmax({}, min, max);

assert(min == 0);

assert(max == 0);

}

void test\_scale1() {

string h1 = show\_histogram\_svg({1});

string expected\_string = "<?xml version='1.0' encoding='UTF-8'?>\n";

expected\_string += "<svg width='300' height='300' viewBox='0 0 300 300' xmlns='http://www.w3.org/2000/svg'>\n";

expected\_string += "\t<text x='15' y='20'>1</text>\n";

expected\_string += "\t<rect x='0' y='30' width='30' height='270' stroke='black' fill='blue' />\n";

expected\_string += "</svg>\n";

assert(h1 == expected\_string);

}

void test\_scale2() {

string h1 = show\_histogram\_svg({1, 2});

string expected\_string = "<?xml version='1.0' encoding='UTF-8'?>\n";

expected\_string += "<svg width='300' height='300' viewBox='0 0 300 300' xmlns='http://www.w3.org/2000/svg'>\n";

expected\_string += "\t<text x='15' y='20'>1</text>\n";

expected\_string += "\t<rect x='0' y='30' width='30' height='135' stroke='black' fill='blue' />\n";

expected\_string += "\t<text x='45' y='20'>2</text>\n";

expected\_string += "\t<rect x='30' y='30' width='30' height='270' stroke='black' fill='blue' />\n";

expected\_string += "</svg>\n";

assert(h1 == expected\_string);

}

int main() {

test\_positive();

test\_negative();

test\_equal();

test\_empty();

test\_scale1();

test\_scale2();

}