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 программного обеспечения систем управления»**

# «Декомпозиция программы»

Выполнила студентка группы А-02-20

Воеводова Надежда Андреевна

Проверил

Мохов А.С.

Козлюк Д.А.

Василькова П.Д.

Москва 2021

Ссылка на репозитарий: https://github.com/VoevodovaNA/lab03

**Цель работы:**

1. Уметь структурировать программу при помощи функций.
2. Уметь писать модульные тесты.

# Задание:

# Часть 1. Декомпозиция программы функциями

Программа для построения гистограммы из ЛР № 1 состоит из одной функции main() на более чем 100 строк, из-за чего в ней неудобно ориентироваться. Необходимо выделить части программы в функции:

* Ввод чисел:
  + принимает количество чисел, которое необходимо ввести;
  + возвращает вектор чисел.
* Поиск наибольшего и наименьшего значения:
  + принимает вектор чисел;
  + возвращает два результата — min и max.
* Расчет гистограммы:
  + принимает вектор чисел и количество корзин;
  + возвращает вектор количеств чисел в каждой корзине;
  + *вызывает* в процессе работы функцию поиска min и max.

# Часть 2. Вывод гистограммы как изображения в формате SVG

Требуется вместо текстовой гистограммы рисовать картинку, например:

Изображение выглядит как стрела

Автоматически созданное описание

# Часть 3. Модульное тестирование

Написать модульный тест для функции поиска минимума и максимума.

**Индивидуальное задание. Вариант 7**

Вычислите среднюю высоту столбца. Если столбец выше, цвет столбца должен быть красным, если ниже или равен средней высоте, цвет зеленый.

**Логика решения варианта:**

Создаём функцию, которая состоит из цикла, суммирующего все числа из массива, и деления этой суммы на количество всех элементов массива.

**Тесты для индивидуального задания:**

Тесты написаны для функции, которая вычисляет среднее значение. 1) тест, когда ожидается верное среднее значение из массива трёх чисел 2) тест с пустым массивом, ожидается, что среднее значение 0.

**Исходный код всех модулей:**

**main.cpp**

#include <iostream>

#include <vector>

#include <cmath>

#include <string>

#include <cstddef>

#include "histogram.h"

#include "svg.h"

using namespace std;

const size\_t SCREEN\_WIDTH = 80;

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

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

double min, max;

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

find\_minmax(numbers, min, max);

for (double number : numbers) {

size\_t bin = (size\_t)trunc(((number - min) / (max - min))

\* (double)bin\_count);

if (bin == bin\_count) {

bin--;

}

bins[bin]++;

}

return bins;

}

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

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 = MAX\_ASTERISK / (double)max\_count;

height = static\_cast<size\_t>((double)bin \* scaling\_factor);

}

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

cout << '\*';

cout << '\n';

}

}

int main() {

size\_t number\_count;

cerr << "Enter number count: ";

cin >> number\_count;

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, min, max, bin\_count);

show\_histogram\_svg(bins,stroke,fill);

\*/

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

auto val\_sign = fun\_sign(numbers, bin\_count);

show\_histogram\_svg(bins,val\_sign);

return 1;

}

**histogram.cpp**

#include "histogram.h"

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

{

if (numbers.size() == 0)

return;

else

{

min = numbers[0];

max = numbers[0];

for (double number : numbers)

{

if (number < min)

{

min = number;

}

if (number > max)

{

max = number;

}

}

}

}

double mean(const vector<size\_t>& bins) {

if (bins.size()==0)

return 0;

double result;

double sum=0;

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

{

sum = sum + (double)(bins[i]);

}

result=sum/((double)(bins.size()));

return result;

}

**histogram.h**

#ifndef HISTOGRAM\_H\_INCLUDED

#define HISTOGRAM\_H\_INCLUDED

#include <vector>

#include <iostream>

#include <math.h>

#include <string>

using namespace std;

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

double mean(const vector<size\_t>& bins);

#endif // HISTOGRAM\_H\_INCLUDED

**svg.cpp**

#include "svg.h"

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

}

double fun\_sign(const vector<double> numbers, size\_t bin\_count)

{

double max, min;

find\_minmax(numbers, min, max);

double bin\_size = (max - min) / (double)bin\_count;

bin\_size = round(bin\_size \* 100) / 100;

double val\_sign=0;

val\_sign = val\_sign + bin\_size;

return val\_sign;

}

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

{

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 SCREEN\_WIDTH = 80;

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

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

double top = 0;

/\* string stroke;

string fill;

for (int i=0; i < bins.size(); i++){

if (bins[i]<=sr)

{

stroke="green";

fill="green";

}

if (bins[i]>sr)

{

stroke="red";

fill="red";

}

}

\*/

size\_t max\_count = 0;

for (size\_t count : bins)

{

if (count > max\_count)

max\_count = count;

}

double sr = mean(bins);

const bool scaling\_needed = (max\_count > MAX\_ASTERISK);

if (scaling\_needed)

{

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

for (size\_t bin : bins)

{

const double height = (double)bin \* scaling\_factor;

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

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

if ((double)bin <= sr)

{

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, "green",

"#ffeeee");

}

else

{

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

"#ffeeee");

}

top += BIN\_HEIGHT;

}

}

else

{

for (size\_t bin : bins)

{

const double bin\_width = static\_cast<double>(BLOCK\_WIDTH \* bin);

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

if ((double)bin <= sr)

{

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, "green",

"#ffeeee");

}

else

{

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

"#ffeeee");

}

top += BIN\_HEIGHT;

}

}

svg\_end();

}

**svg.h**

#ifndef SVG\_H\_INCLUDED

#define SVG\_H\_INCLUDED

#include <vector>

#include <iostream>

using namespace std;

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

void svg\_end();

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

void svg\_rect(double x, double y, double width, double height, string stroke,

string fill);

double fun\_sign(const vector<double> numbers, size\_t bin\_count);

void show\_histogram\_svg(const vector<size\_t>& bins, double val\_sign);

#endif // SVG\_H\_INCLUDED

**test.cpp**

#include "histogram.h"

#include "svg.h"

#include <cassert>

#include <vector>

void

test\_positive()

{

double min = 0;

double max = 0;

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

assert(min == 1);

assert(max == 3);

return;

}

void test\_negative()

{

double min = 0;

double max = 0;

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

assert(min == -3);

assert(max == -1);

return;

}

void test\_equal()

{

double min = 0;

double max = 0;

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

assert(min == 1);

assert(max == 1);

return;

}

void test\_one\_number()

{

double min = 0;

double max = 0;

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

assert(min == 1);

assert(max == 1);

return;

}

void test\_empty()

{

double min=0;

double max=0;

cerr << "test" << endl;

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

}

void test1()

{

vector<size\_t> fortest1 = {3,4,5};

size\_t expected=4;

double e=10e-5;

size\_t val = mean(fortest1);

assert(val - expected < e);

}

void test2()

{

vector<size\_t> fortest2;

size\_t expected=0;

size\_t val = mean(fortest2);

assert(val == expected);

}

int

main()

{

test\_positive();

test\_negative();

test\_equal();

test\_one\_number();

test\_empty();

test1();

test2();

return 0;

}