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»

# «Декомпозиция программы»

# Выполнила студентка группы А-03-20

# Кашина Е. В.

# Проверили

# Мохов А. С

# Козлюк Д. А

Москва 2021

# Цель работы

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

# Часть 1. Декомпозиция программы функциями

## Задание

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

## Задача

Требуется вместо текстовой гистограммы рисовать картинку

# Часть 3. Модульное тестирование

Написать модульный тест для функции поиска минимума и максимума.

**Индивидуальное задание**

**Вариант 10**

Отображайте гистограмму вертикально с подписями сверху (Аналогично заданию из ЛР №1)

3 5 2

\* \* \*

\* \* \*

\* \*

\*

Предусмотреть расчет IMAGE\_HEIGHT таким образом, чтобы вся гистограмма вмещалась в область рисунка.

**Логика решения варианта :**

Логика построения гистограммы : необходимо изменить аргументы , которые подаются в функции svg\_text и svg\_rect. Теперь у нас добавляется смещение не к вертикальной координате y , а к горизонтальной x. В связи с этим поменяем название некоторых констант , чтобы легче было ориентироваться в программе. Масштабирование будем делать всегда и немного изменим его. Теперь при нахождении scaling\_factor будем в числителе писать не MAX\_ASTERISK , а (IMAGE\_HEIGHT-TEXT\_HEIGHT). Тогда корзинка с максимальным количеством элементов будет занимать всю длину картинки, а остальные ориентироваться на нее.

**Модульные тесты :**

Работать будет с модулем SVG. Будем проверять , правильно ли строятся столбцы для 1) двух гистограмм , у которых во всех корзинах одно и то же количество элементов 2) для двух гистограмм с разным количеством элементов в корзинах 3)предусмотреть обработку пустого массива (проверка длины массива и выход из функции , если она нулевая).

Для этого немного изменим функцию показа гистограммы , добавив в аргументы высоту блока через ссылку.



**Код программы.**

**main.cpp**

#include <iostream>

#include <vector>

#include "histogram.h"

#include "svg.h"

using namespace std;

vector<double>

input\_numbers(size\_t count)

{

vector<double> result(count);

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

{

cin >> result[i];

}

return result;

}

int

main() {

size\_t number\_count;

cerr << "Enter number count: ";

cin >> number\_count;

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);

size\_t bin\_height;

show\_histogram\_svg(bins, bin\_height);

return 0;

}

**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);

vector<size\_t>

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

vector<size\_t>

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

#endif // HISTOGRAM\_H\_INCLUDED

**histogram.cpp**

#include "histogram.h"

#include <iostream>

void

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

{

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 , max ;

find\_minmax(numbers,min,max);

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

for (double number : numbers)

{

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

if (bin == bin\_count) {

bin--;

}

result[bin]++;

}

return result;

}

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.h**

#include "histogram.h"

#include <iostream>

void

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

{

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 , max ;

find\_minmax(numbers,min,max);

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

for (double number : numbers)

{

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

if (bin == bin\_count) {

bin--;

}

result[bin]++;

}

return result;

}

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.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\_end()

{

cout << "</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 width, double height, string stroke, string fill)

{

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

}

void

show\_histogram\_svg(const vector<size\_t>& bins, size\_t& bin\_height)

{

if (bins.size() == 0)

{

return;

}

const auto IMAGE\_WIDTH = 400;

const auto IMAGE\_HEIGHT = 300;

const auto TEXT\_LEFT = 15;

const auto TEXT\_BASELINE = 20;

const auto TEXT\_HEIGHT = 30;

const auto BIN\_WIDTH = 30;

const auto BLOCK\_HEIGHT = 10;

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

double left = 0;

size\_t max\_count = bins[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;

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

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

svg\_rect(left, TEXT\_HEIGHT, BIN\_WIDTH, bin\_height, "white", "#c71585" );

left += BIN\_WIDTH;

}

svg\_end();

}

**test.cpp**

#include <cassert>

#include "histogram.h"

#include "svg.h"

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\_ordinary ()

{

double min=0;

double max=0;

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

assert(min == 1);

assert(max == 1);

}

void

test\_single()

{

double min=0;

double max=0;

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

assert(min == 1);

assert(max == 1);

}

void

test\_empty ()

{

double min=0;

double max=0;

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

}

void the\_same ()

{

size\_t height\_1;

show\_histogram\_svg({1,1,1}, height\_1);

size\_t height\_2;

show\_histogram\_svg({5,5,5},height\_2);

assert (height\_1==height\_2);

}

void bin\_empty ()

{

size\_t height;

show\_histogram\_svg({},height);

}

void scalling()

{

size\_t height\_1;

show\_histogram\_svg({1,2,3}, height\_1);

size\_t height\_2;

show\_histogram\_svg({6,5,4}, height\_2);

assert(height\_1>height\_2);

}

int main()

{

test\_positive ();

test\_negative ();

test\_ordinary ();

test\_single ();

test\_empty ();

the\_same();

bin\_empty ();

scalling ();

}