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»**

Выполнил студент группы А-02-19

Сонин А. В.

Проверили

Мохов А. С.

Козлюк Д. А.

Москва 2020

**Цель работы**

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

#### Вариант 16

После запроса количества столбцов запросить цвет линий для каждого столбца. Проверять ввод: цвет должен либо начинаться с #, либо не иметь внутри пробелов.

**Логика решения:**

Создаём отдельную функцию для ввода в программу цвета линий для каждого столбца - vector <string> input\_colors(size\_t bin\_count), эта фунцкия является вектором(одномерным массивом) с именем input\_colors и параметром = числу столбцов bin\_count.

Эта функция обрабатывает входную строку(вводится из консоли или текстого файла), функция присваивает каждому элементу массива colors[i] одно слово. Далее программа проверяет слова на правильность: цвет(слово) должен либо начинаться с #(формат #RRGGBB), либо не иметь внутри пробелов. Результат функции это массив цветов для линий каждого столбца.

Далее в функции main() мы присвоим результат этой функции переменной colors которую потом подадим на ввод функции show\_histogram\_svg(bins, colors, bin\_count); которая рисуют гистограмму.

Ссылка на репозитарий (GitHub).

<https://github.com/SoninAV/lab03/>

**Код программы:**

**main.cpp**

#include <iostream>

#include <vector>

#include <string>

#include <sstream>

#include <cmath>

#include <cstdio>

#include <windows.h>

#include "histogram.h"

#include "svg.h"

using namespace std;

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;

}

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';

}

}

int main()

{

size\_t number\_count;

cerr << "Enter number count: ";

cin >> number\_count;

cerr << "Enter numbers: ";

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

size\_t bin\_count;

cerr << "Enter column count: ";

cin >> bin\_count;

double min;

double max;

find\_minmax(numbers,min,max);

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

cerr << "Enter column colors: ";

const auto colors = input\_colors(bin\_count);

show\_histogram\_svg(bins, colors, bin\_count);

return 0;

}

**histogram.h**

#ifndef HISTOGRAM\_H\_INCLUDED

#define HISTOGRAM\_H\_INCLUDED

#include <vector>

#include <string>

using namespace std;

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

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

#endif // HISTOGRAM\_H\_INCLUDED

**histogram.cpp**

#include<iostream>

#include "histogram.h"

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 <string> input\_colors(size\_t bin\_count)

{

vector<string> colors(bin\_count);

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);

}

return colors;

}

**Svg.h**

#ifndef SVG\_H\_INCLUDED

#define SVG\_H\_INCLUDED

#include <vector>

#include <string>

using namespace std;

vector<string> colors(size\_t bin\_count);

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 = "black");

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

#endif

**svg.cpp**

#include <iostream>

#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>";

}

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, const vector<string>& colors, size\_t bin\_count){

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;

}

}

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;

}

}

svg\_end();

}