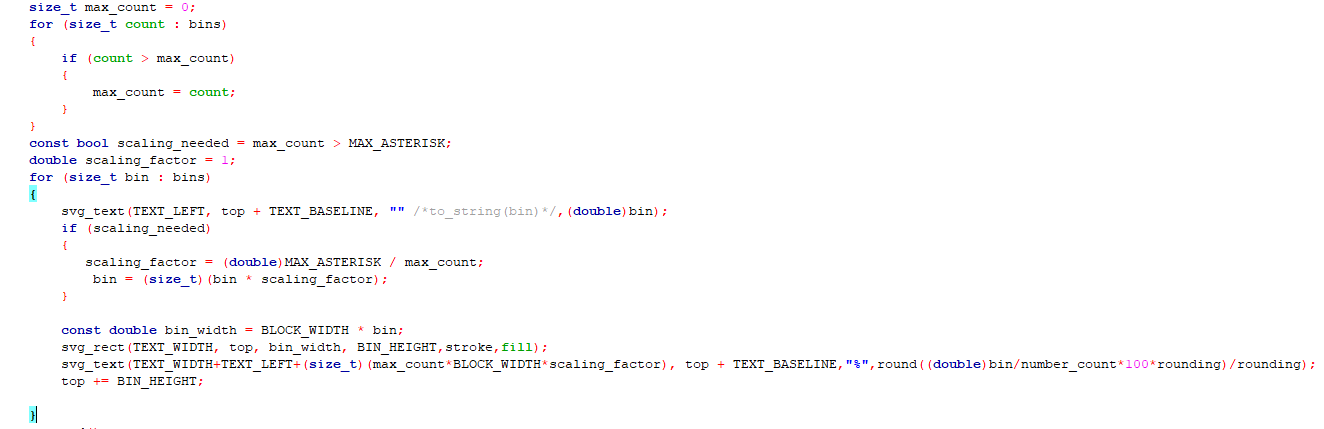
Отчёт по лабораторной работе №3 по теме «Декомпозиция программ»

по предмету

Разработка Программного Обеспечения Систем Управления

Изотова Артёма Александровича, А-03-19

1. Реализацию масштабирования для отрисовки прямоугольников сделал аналогично масштабированию для звёздочек, для начала нахожу максимальное количество чисел в корзине и узнаю нужно ли масштабирование и уже в цикле отрисовки проверяю если масштабирование необходимо, то bin умножаю на коэффициент масштабирования, который отвечает за ширину прямоугольников:



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;

double scaling\_factor = 1;

for (size\_t bin : bins)

{

svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, "" ,(double)bin);

if (scaling\_needed)

{

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

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

}

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

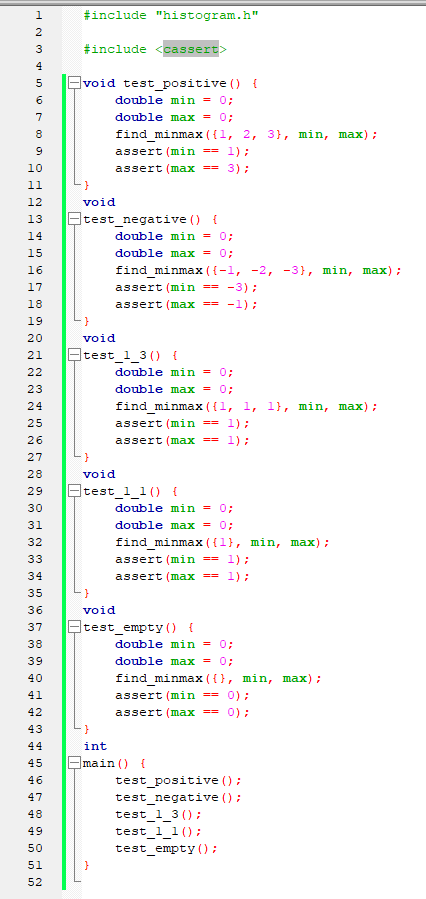
svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT,stroke,fill);

svg\_text(TEXT\_WIDTH+TEXT\_LEFT+(size\_t)(max\_count\*BLOCK\_WIDTH\*scaling\_factor), top + TEXT\_BASELINE,"%",round((double)bin/number\_count\*100\*rounding)/rounding);

top += BIN\_HEIGHT;

}

1. Реализация тестов. Для реализации мы создали test.cpp и подключили к нему histogram.h, с помощью которого мы сможем использовать функцию find\_minmax для проверки. Также мы подключили cassert это модуль для проверки ошибок, в частности мы используем функцию assert, которая сравнивает ожидаемые значения с реальными и в случае, если это не так программа выдаёт ошибку, и мы прогоним 5 тестов, где у нас массив положительных чисел, отрицательных, массив из одинаковых чисел, из одного числа , и без чисел.



#include "histogram.h"

#include <cassert>

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\_1\_3() {

double min = 0;

double max = 0;

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

assert(min == 1);

assert(max == 1);

}

void

test\_1\_1() {

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);

assert(min == 0);

assert(max == 0);

}

int

main() {

test\_positive();

test\_negative();

test\_1\_3();

test\_1\_1();

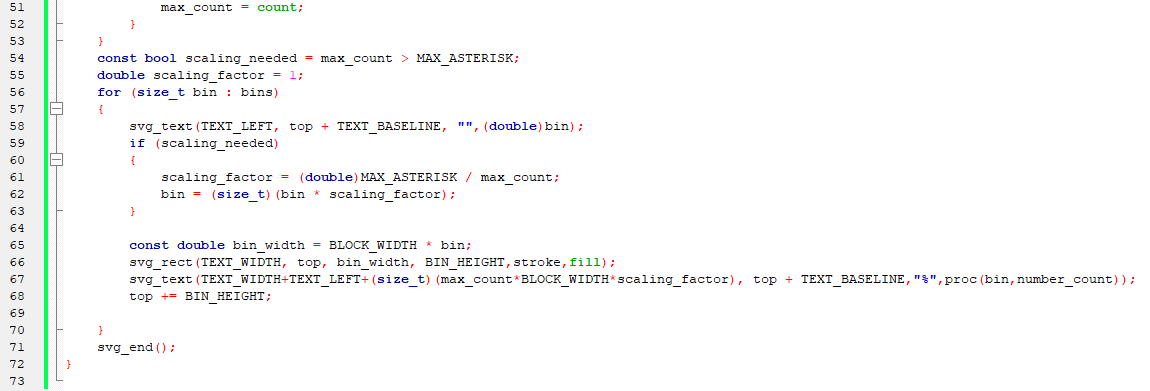
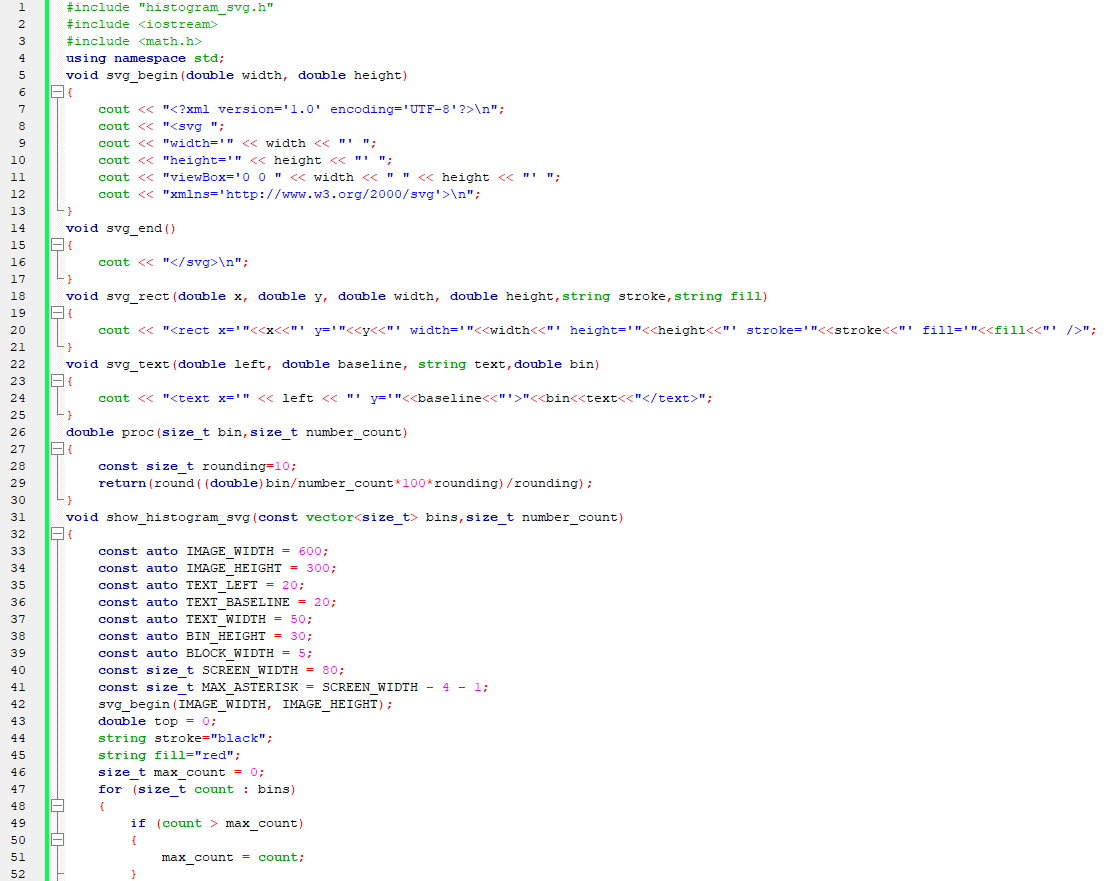
test\_empty();

}

1. Реализация моего варианта дз. Мне было необходимо добавить после прямоугольников проценты чисел входящих в корзину от общего количества чисел, причём они должны были быть на одном уровне и округлены до одного числа после запятой. Для этого мы рассчитываем отступ от левого края причём, рассчитывать необходимо всегда от максимального прямоугольника, чтоб они были на одном уровне и изменятся будет только высота по y, причём также необходимо учесть масштабирование, если максимальный столбец уменьшен, то и расстояние тоже должно уменьшится.

TEXT\_WIDTH+TEXT\_LEFT+(size\_t)(max\_count\*BLOCK\_WIDTH\*scaling\_factor)

Вот такой параметр я передаю для нахождение отступа по Х , приведение типов необходимо так как расстояние должно быть целым. Для реализации округления я подключил модуль math.h, так как в нём есть функция round ,которая округляет в сторону правильного математического округления.



#include "histogram\_svg.h"

#include <iostream>

#include <math.h>

using namespace std;

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\_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 svg\_text(double left, double baseline, string text,double bin)

{

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

}

double proc(size\_t bin,size\_t number\_count)

{

const size\_t rounding=10;

return(round((double)bin/number\_count\*100\*rounding)/rounding);

}

void show\_histogram\_svg(const vector<size\_t> bins,size\_t number\_count)

{

const auto IMAGE\_WIDTH = 600;

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 = 5;

const size\_t SCREEN\_WIDTH = 80;

const size\_t MAX\_ASTERISK = SCREEN\_WIDTH - 4 - 1;

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

double top = 0;

string stroke="black";

string fill="red";

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;

double scaling\_factor = 1;

for (size\_t bin : bins)

{

svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, "",(double)bin);

if (scaling\_needed)

{

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

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

}

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

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT,stroke,fill);

svg\_text(TEXT\_WIDTH+TEXT\_LEFT+(size\_t)(max\_count\*BLOCK\_WIDTH\*scaling\_factor), top + TEXT\_BASELINE,"%",proc(bin,number\_count));

top += BIN\_HEIGHT;

}

svg\_end();

}