РПОСУ. ЛР № 3. Декомпозиция программы

Цель работы

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

Часть 1. Декомпозиция программы функциями

Задание

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.

При отсутствии решения ЛР № 1 его можно взять здесь:

Указания

Важно. Работайте в проекте (File → New → Project → Console Application), а не в отдельном файле.

Функция ввода чисел

Итак, имеется код для ввода массива чисел:

```
vector<double> numbers(number_count);
for (size_t i = 0; i < number_count; i++) {
    cin >> numbers[i];
}
```

Требуется вынести его в функцию. Так как ее назначение — ввести числа (input numbers), логично дать ей имя `input_numbers`. Принято называть функции так, чтобы имя означало действие, а не процесс. Например, `number_input` (ввод чисел) было бы худшим вариантом.

От чего зависит `input_numbers()`? От количества чисел, то есть от `number_count`. Это будет параметр функции. Поскольку других количеств в функции не будет, параметр можно назвать просто `count`.

Каков результат работы функции? Вектор введенных чисел типа `vector<double>`. Поскольку это результат функции, внутри нее эта переменная будет называться `result`. В современном C++ возвращать вектор из функции удобно и эффективно.

Итого, функция будет использоваться следующим образом:

```
vector<double> numbers = input_numbers(number_count);
```

Это строчкой будет заменен код ввода чисел в функции `main()`.

Саму функцию `input_numbers()` нужно поместить над функцией `main()`:

```
vector<double>
input_numbers(size_t count) {
    vector<double> result(count);
    for (size_t i = 0; i < count; i++) {
        cin >> result[i];
    }
    return result;
}

// Продолжение старого кода:
int main() {
    size_t number_count;
    cout << "Enter number count: ";
    cin >> number_count;

    // Ввод чисел заменен вызовом функции:
    vector<double> numbers = input_numbers(number_count);
    ...
```

И в целом, функции должны быть описаны выше своего первого использования.

Вывод типов переменных

Выбран ли тип переменной `numbers` наилучшим образом?

Можно заметить, что в дальнейшем `numbers` не изменяется, следовательно, можно сделать ее константной:

```
const vector<double> numbers = input_numbers(number_count);
```

Теперь тип `numbers` занимает половину строки, что может быть неудобно читать. Можно заметить, что тип `numbers` обязан совпадать с типом возвращаемого значения функции `input_numbers()`, то есть избыточен. В современном C++ есть способ указать компилятору *вывести* тип переменной `numbers` из выражения, которое ей присваивается:

```
const auto numbers = input_numbers(number_count);
```

Ключевое слово `auto` не вводит новый тип. Переменная `numbers` по-прежнему является `const vector<double>`, просто это записано короче, так как компилятору это известно без явного указания. Стала ли программа более читаемой? Ответ неоднозначен. Код сократился, но теперь из него не очевидно (человеку), каков тип `numbers`; с другой стороны, читателю может и не быть важно, как представлены `numbers` — достаточно того, что там числа (это ясно из названия функции). Вывод: использовать `auto` следует там, где это сокращает код без сокрытия важных подробностей.

Внесите в программу описанные изменения и убедитесь, что она собирается.

Функция поиска минимума и максимума

При поиске минимума и максимума результатов два, оператор `return` не позволяет вернуть два значения. (Это можно обойти, но способ выходит за рамки ЛР.) Типовой вариант — использовать выходные параметры:

```
double min, max;
find_minmax(numbers, min, max);
```

Нужно сделать так, чтобы этот код работал следующим образом: в теле функции `find_minmax()` изменялись бы аргументы `min` и `max`, и эти изменения должны отразиться на переменных `min` и `max` в функции `main()`.

Предположим, функция определена так:

```
void
find_minmax(vector<double> numbers, double min, double max) {
    min = numbers[0];
    // (здесь код поиска минимума и максимума)
}
```

Желаемого поведения не будет: параметры `min` и `max` в функции — это отдельные переменные, хотя они и называются так же, как переменные в `main()`. Когда функция `find_minmax()` вызывается, параметры `min` и `max` получают *значения*, которым были равны второй и третий аргумент (говорят: передаются по значению). Любые изменения `min` и `max` в теле функции не отразятся на аргументах.

Используем ссылки (амперсанды после `double`):

```
void
find_minmax(vector<double> numbers, double& min, double& max) {
    min = numbers[0];
    // (здесь код поиска минимума и максимума)
}
```

Теперь параметры `min` и `max` — это не самостоятельные переменные, а просто имена для аргументов, переданных в функцию при вызове. Когда изменяется `min` или `max` в теле функции, изменяется и переменная в `main()`. Важно понимать, что переменные связаны не по имени, а по порядку передачи

в функцию. Например, переменные в `main()` можно было бы назвать `numbers_min` и `numbers_max`, это никак не повлияло бы на работу функции.

1. Выделите в программе функцию `find_minmax()`, добейтесь компиляции и корректной работы программы.

Передача по константной ссылке

Оптимально ли выбран тип входного параметра `numbers`? Во-первых, изменять его не требуется, он может быть константным. Во-вторых, поскольку он не объявлен ссылкой или указателем, это отдельная переменная, которая получает значением копию вектора, который передается в функцию. Однако копия не нужна. Итого выгоднее использовать константную ссылку: `const vector<double>& numbers`.

2. Измените тип параметра `numbers` функции `find_minmax()`. Проверьте, что программа компилируется, а ее поведение не изменилось.

3. Выделите расчет количества чисел в столбцах гистограммы в функцию `make_histogram()`.

4. Выделите отображение гистограммы в функцию `show_histogram_text()`.

Декомпозиция программы

Итоговая функция `main()` принимает вид (за вычетом оформления):

```
size_t number_count;
cin >> number_count;
const auto numbers = input_numbers(number_count);
size_t bin_count;
cin >> bin_count;

const auto bins = make_histogram(numbers, bin_count);

show_histogram_text(bins);
```

Можно видеть, что функциями фактически выделены блоки ввода данных, обработки данных и вывода результатов. Ввод данных частично находится вне функции `input_numbers()`, потому что иначе она оказалась бы слишком специфичной, «заточенной» под программу. Функция же ввода массива может быть полезной во многих случаях.

Помимо выделения блоков кода, функции определяют взаимодействие между частями программы и скрывают реализацию этих частей. Через типы параметров функций и их возвращаемых значений фиксируется, какой информацией обмениваются части программы *(интерфейсы* этих частей, не путать с интерфейсом пользователя). При грамотной декомпозиции интерфейсы сохраняются (иногда дополняясь) при модернизации частей программы.

5. Добейтесь корректной работы программы.

6. Сделайте коммит с текущим состоянием проекта и отправьте его на GitHub.

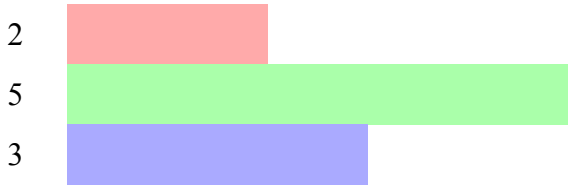
Если вы клонировали авторское решение ЛР № 1, изменить адрес репозитария на собственный можно такой командой (если ваш пользователь `UserName` завел чистый репозитарий `RepoName`):

```
git remote set-url origin https://github.com/UserName/RepoName.git
```

Часть 2. Вывод гистограммы как изображения в формате SVG

Задача

Требуется вместо текстовой гистограммы рисовать картинку, например:



Как браузер отображает страницы, или что такое SVG и при чем он здесь.

Возьмем любую страницу в интернете, например, эту. На ней присутствует визуальное оформление: заголовки крупные и жирные, ссылки подчеркнуты и по ним можно перейти, между абзацами есть отступы. Можно догадаться, что изображение на экране не приходит с сервера в готовом виде, потому что страница выглядит по-разному на телефоне, на компьютере, на экране другого разрешения (размера). Страница передается в виде разметки — текста со специальными элементами — на основании которой браузер рисует на экране один текст как заголовок, второй — как ссылку, третий — отдельным абзацем.

Например, возьмем такой абзац:

> Текст **с полужирными** словами и ссылкой в никуда.

Бразуер отображает его из следующего фрагмента разметки (язык разметки называется HTML, hyper-text markup language):

```
<p>Текст <strong>с полужирными</strong> словами и <a href="#">ссылкой</a> в никуда.</p>
```

Всегда можно нажать *Ctrl+U* и увидеть исходный код страницы (разметку).

SVG работает так же, только разметка описывает не текст с оформлением, а изображение. HTML указывает, как отображать определенные блоки текста, а SVG указывает, какие геометрические фигуры нарисовать. Если щелкнуть правой кнопкой по гистограмме выше и выбрать «Показать изображение», оно откроется отдельно. Тем же *Ctrl+U* можно посмотреть код разметки гистограммы, который в итоге будет выводить программа.

Таким образом, наша программа может печатать текст, только не псевдографику (звездочки), а разметку SVG. Если перенаправить вывод программы в файл с расширением `*.svg` и открыть его браузером, вместо разметки будет нарисовано изображение, которое она задает.

Формат SVG

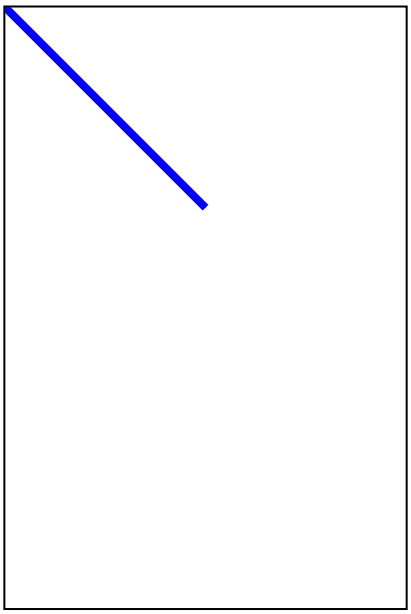
Изображения в формате SVG представляют собой текстовые файлы, в которых перечислены геометрические фигуры (линии, прямоугольники и прочие) и их свойства (положение, размер, цвет и другие). Кроме самих геометрических фигур, файл содержит заголовок и окончание.

Например, так описывается изображение размером 200×300 точек с линией из точки *(0, 0)* в точку *(100, 100)* синего цвета толщиной 4 точки:

```
<?xml version='1.0' encoding='UTF-8' ?>
<svg width='200' height='300' viewBox='0 0 200 300' xmlns='http://www.w3.org/2000/svg'>
<line x1='0' y1='0' x2='100' y2='100' stroke='blue' stroke-width='4'/>
</svg>
```

Здесь строка `<line... />` описывает линию. Выше нее находится заголовок файла, ниже нее — окончание файла.

Вот это изображение (для наглядности добавлена граница):



Система координат SVG отличается от математической: ось Y направлена вниз, то есть точка *(0,0)* находится в верхнем левом углу. Координаты в SVG действительные, то есть может быть точка *(0.5, 3.14)*.

Фрагменты вида `<svg>` или `<line>` называются *элементами,* или *тэгами.* Параметры вида `height='300'` называются *атрибутами* элементов. Подробнее об SVG можно прочитать здесь.

Кроме линий, SVG поддерживает множество элементов, среди них:

- `<text x='20' y='35'>anything you want</text>`: текст «anything you want», левый нижний угол которого в точке *(20,35);*
- `<rect x='0' y='0' width='100' height='200' />`: прямоугольник 100×200 с верхним левым углом в точке *(0,0)*.

Указания

С точки зрения SVG, гистограмма представляет собой элементы `<text>` с подписями столбцов и элементы `<rect>` напротив них — столбцы гистограммы.

Вывод отдельных элементов SVG (заголовка, окончания, текста, прямоугольника) стоит оформить в виде функций, который будут принимать параметры геометрии и печатать соответствующий текст. Весь вывод гистограммы в SVG реализуем в функции `show_histogram_svg()`, интерфейс которой не отличается от `show_historgam_text()`, вызов которой заменим вызовом новой функции.

Для проверки работы программы можно направлять ее стандартный вывод в файл с расширением SVG и просматривать его в браузере.

Вывод заголовка и окончания SVG

Отработаем цикл модификации и проверки программы на примере пустого изображения. Вот готовые функции вывода заголовка и окончания SVG:

```
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";
}
```

Обратите внимание на пробелы в строках, например, перед закрывающей кавычкой во фрагменте `"viewBox='0 0 "`. Они необходимы между атрибутами. Также обратите внимание на использование одинарных кавычек. Двойные кавычки ограничивают в C++ строковые литералы; одинарные кавычки внутри них выводятся «как есть» в результирующую строку. Путаница кавычек или отсутствие некоторых пробелов сделает SVG некорректным.

Пусть начальная реализация графического вывода гистограммы всегда выводит пустое изображение фиксированного размера:

```
void
show_histogram_svg(const vector<size_t>& bins) {
    svg_begin(400, 300);
    svg_end();
}
```

Заменим вызов `show_histogram_text(bins)` вызовом `show_histogram_svg(bins)`.

Создать файл изображения можно так (можно скачать marks.txt):

```
C:\lab03-histogram\bin\Debug> lab03-histogram.exe <marks.txt >marks.svg
```

Эту команду нужно вводить в консоли, открытой в папке с exe-файлом проекта. Если проект называется не `lab03-histogram`, название файла нужно написать свое. Просмотреть, какие файлы есть в текущем каталоге, можно командой `dir`.

Чтобы быстро просмотреть из консоли содержимое `marks.svg` (разметку), можно использовать команду:

```
C:\lab03-histogram\bin\Debug> type marks.svg
```

В браузере `marks.svg` открывается из меню *Файл → Открыть…* Отобразится пустая страница (пустой рисунок). Можно нажать *Ctrl+U* или пункт *«Исходный код страницы»* в контекстном меню любого места страницы, чтобы увидеть результирующий код SVG (контекстное меню — это то, что появляется по щелчку правой кнопкой мыши в любом пустом месте страницы).

Функции вывода элементов SVG

Для вывода подписей к столбцам напишем функцию вывода текста в SVG, которая принимает координату по горизонтали (`left`), координату нижнего края текста по вертикали (`baseline`) и сам текст:

```
void svg_text(double left, double baseline, string text);
```

Из введения в SVG выше известно, что она должна выводить строку такого формата:

```
<text x='20' y='35'>anything you want</text>
```

На C++ точно такую строку можно вывести следующим образом (`svg_text()`) должна размещаться выше `show_histogram_svg()`:

```
void
svg_text(double left, double baseline, string text) {
    cout << "<text x='20' y='35'>anything you want</text>";
}
```

Чтобы вместо координаты `x` (числа 20) выводить значение `left`:

```
    cout << "<text x='" << left << "' y='35'>anything you want</text>";
```

Обратите внимание на то, что сохранены одинарные кавычки внутри строк C++ (в двойных кавычках) — они необходимы по правилам SVG.

7. Закончите реализацию `svg_text()`, чтобы подставлять значение координаты `baseline` и текст надписи `text`.

Для проверки выведем высоту первого столбца гистограммы:

```
void
show_histogram_svg(const vector<size_t>& bins) {
    svg_begin(400, 300);
    svg_text(20, 20, to_string(bins[0]));
    svg_end();
}
```

Функция `to_string()` преобразует значения разных типов в строки.

Чтобы проверить модифицированную программу, её нужно запустить повторно:

```
C:\lab03-histogram> bin/Debug/lab03-histogram.exe <marks.txt >marks.svg
```

После повторного запуска программы открытый в браузере файл можно обновить клавишей *F5*. На странице должен быть виден текст «2». Если в файле ошибка, он не отобразится. В этом случае нужно просмотреть код страницы на предмет ошибок в SVG, исправить код программы, перезапустить ее и проверить результат.

8. Напишите функцию для вывода прямоугольника в SVG:

```
void svg_rect(double x, double y, double width, double height);
```

Для проверки выведем первый столбец гистограммы справа от подписи к нему:

```
    svg_rect(50, 0, bins[0] * 10, 30);
```

9. Убедитесь, что вывод первого столбца гистограммы работает.

Исключение «магических констант»

Текст выводится в координатах *(20, 20)*, смещение левого края столбца *50* (оно же — ширина подписей), а высота столбца *30*. Эти числа подобраны так, чтобы элементы гистограммы не накладывались друг на друга, но они разбросаны по коду и их смысл неясен.

Прямо внутри функции `show_histogram_svg()` заведем константы:

```
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;
```

10. Замените «магические числа» константами и проверьте работу программы.

Вывод гистограммы

Логика вывода гистограммы следующая: каждая корзина выводится так же, как первая, но к вертикальной координате добавляется смещение — высота столбца:

```
double top = 0;
for (size_t bin : bins) {
    const double bin_width = BLOCK_WIDTH * bin;
    svg_text(TEXT_LEFT, top + TEXT_BASELINE, to_string(bin));
    svg_rect(TEXT_WIDTH, top, bin_width, BIN_HEIGHT);
    top += BIN_HEIGHT;
}
```

11. Реализуйте вывод гистограммы и проверьте работу программы.

Оформление гистограммы. Значения параметров по умолчанию

Черная гистограмма не слишком эстетична или экономична при печати. За цвет линий в SVG отвечает атрибут `stroke`, а за цвет заливки — `fill`. Можно задать один из стандартных цветов или выбрать цвет в формате `#RRGGBB` из палитры. Пример прямоугольника с красными границами и бледно-розовой заливкой:

```
<rect x='50' y='0' width='30' height='30' stroke='red' fill='#ffeeee'/>
```

12. Доработайте функцию `svg_rect()` для указания цвета линий и заливки:

```
void svg_rect(double x, double y, double width, double height,
        string stroke, string fill);
```

13. Измените цвета вывода и проверьте работу программы.

Значения параметров по умолчанию

Цвета элементов нужно задавать не всегда, а в текущей реализации у функции `svg_rect()` шесть параметров, два из которых отвечают за цвет. Можно было бы перегрузить функцию:

```
void svg_rect(double x, double y, double width, double height);

void svg_rect(double x, double y, double width, double height,
        string stroke, string fill);
```

Однако в этом случае пришлось бы в каждой функции писать похожий код. В данном случае выгоднее применить значения параметров по умолчанию:

```
void svg_rect(double x, double y, double width, double height,
        string stroke = "black", string fill = "black");
```

Значения по умолчанию указываются только один раз в объявлении функции (если объявление и определение отделены). Новая версия `svg_rect()` работает так:

```
svg_rect(0, 0, 100, 200);                   // svg_rect(0, 0, 100, 200, "black", "black");
svg_rect(0, 0, 100, 200, "red");            // svg_rect(0, 0, 100, 200, "red",   "black");
svg_rect(0, 0, 100, 200, "blue", "#aaffaa");
```

14. Добавьте значения параметров по умолчанию функции `svg_rect()`.

15. Сделайте коммит и отправите изменения на GitHub.

16. **(Домашнее задание.)** Реализуйте для гистограммы в SVG масштабирование, как было сделано для текста в ЛР № 1. Опубликуйте коммит.

Часть 3. Модульное тестирование

Ранее для тестирования программы применялось перенаправление ввода и вывода. Такое тестирование называется функциональным. Однако функционального тестирования недостаточно для больших программ. Предположим, большая программа перестала выдавать правильный результат — как найти место ошибки? Необязательно это последнее изменение: возможно, другая часть и ранее работала неверно в некоторых случаях, и последняя правка лишь привела к такому случаю.

Модульное тестирование (unit testing) проверяет не работу всей программы, а работу отдельных ее компонент, например, отдельных функций. Модульные тесты пишутся программистами для собственного кода (функциональные тесты могут писаться или проводятся вручную другими специалистами).

Модульный тест — это отдельная программа, которая изолированно проверяет части кода основной программы. Если желательно протестировать части сложного алгоритма, эти части должны быть оформлены в виде отдельных функций (говорят: код должен быть *тестируемым)*.

Задача

Написать модульный тест для функции поиска минимума и максимума.

Указания

Выделение тестируемой функции в модуль

Функция `find_minmax()` находится в основном (и единственном) модуле `main.cpp`. Для использования функции в других программах нужно вынести ее код в отдельный модуль. Модулем в практике C++ называется заголовочный файл (`*.h`) с объявлениями функций и файл реализации (`*.cpp`) с их определениями.

Создание заголовочного файла

17. При помощи меню *File* → *New* → *File…* добавьте к проекту заголовочный файл *(C/C++ header)*. Он должен быть расположен в каталоге проекта:

    Filename with full path: `C:\lab03-histogram\histogram.h` Должна стоять галочка в пункте *Add file to active project*,

Заготовка файла включает «стража включения» (см. лекции):

```
#ifndef HISTOGRAM_H_INCLUDED
#define HISTOGRAM_H_INCLUDED

#endif // HISTOGRAM_H_INCLUDED
```

Можно заменить его на более простой вариант `#pragma once` или писать код в этом файле между `#define` и `#endif`.

18. Перенесите функцию `find_minmax()` в `histogram.h`.

Помимо самой функции в начало `histogram.h` (под `#pragma once`) нужно добавить подключение библиотеки векторов и стандартного пространства имен:

```
#include <vector>

using namespace std;
```

Если теперь попытаться собрать программу, это не удастся: в файле `main.cpp` функция `find_minmax()` теперь отсутствует. Необходимо подключить файл `histogram.h` (в самом начале файла `main.cpp`):

```
#include "histogram.h"
```

Обратите внимание на использование кавычек вместо угловых скобок: кавычки означают, что путь к файлу `*.h` написан относительно файла с `#include`, а угловые скобки предписывают искать включаемый файл по определенным в настройках путям.

19. Скомпилируйте программу и проверьте, что она работает.

Создание файла реализации

20. При помощи меню *File* → *New* → *File…* добавьте к проекту файл реализации *(C/C++ source)*. Необходимые настройки:

    Filename with full path: `C:\lab03-histogram\histogram.cpp`
    Add file to active project in build target(s) — нажать *All*, либо проставить флажки в пунктах *Debug* и *Release*

21. В `histogram.cpp` подключите `histogram.h`.

22. Перенесите определение функции `find_minmax()` в `histogram.cpp`.

23. Оставьте в `histogram.h` от функции `find_minmax()` только объявление:

    ```
    void find_minmax(const vector<double> numbers, double& min, double& max);
    ```

Если при выделении функции `find_minmax()` были использованы указатели, а не ссылки, объявление нужно соответствующим образом скорректировать.

24. Соберите программу и проверьте ее работу.

Создание проекта для модульных тестов

25. При помощи меню *File* → *New* → *Project…* создайте новый проект типа *Empty project.* (НЕ *Console application*!) Необходимые настройки:

    Project title: `lab03-test`
    Folder to create project in: `C:\lab03-histogram`
    Project filename: `lab03-test.cbp`
    Resulting filename: `C:\lab03-histogram\lab03-test.cbp`

26. Дважды щелкните по проекту `lab03-test`, чтобы сделать его активным.

27. Из контекстного меню проекта `lab03-test` выберите пункт *Add files…* и добавьте к проекту `histogram.h` и `histogram.cpp`. В открывшемся диалоге *Select the targets this file should belong to:* проставьте все флажки.

28. При помощи меню *File* → *New* → *File…* добавьте к проекту `lab03-test` файл реализации *(C/C++ source)* `test.cpp`. Необходимые настройки:

Filename with full path: `C:\lab03-histogram\test.cpp`
Add file to active project in build target(s) — нажать *All*, либо проставить флажки в пунктах *Debug* и *Release*

Написание модульных тестов

Файл `test.cpp` представляет собой полноценную тестирующую программу, в которой подключен тестируемый модуль, библиотека с `assert()` (см. лекцию) и присутствует функция `main()`:

```
#include "histogram.h"

#include <cassert>

int
main() {
}
```

Первый тест будет проверять простой случай массива положительных 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);
}

int
main() {
    test_positive();
}
```

29. Соберите программу-тест, запустите ее и убедитесь, что она завершается без ошибок, то есть тест проходит.

30. Замените проверку `max == 3` на `max == 4`. Соберите и запустите программу, чтобы наблюдать образец вывода при провале теста. Верните правильную проверку.

31. Добавьте новые тесты (имена придумайте) на такие случаи:

    - массив из трех отрицательных чисел;
    - массив из трех одинаковых чисел;
    - массив из одного числа.

    Убедитесь, что все тесты проходят.

32. Добавьте тест на обработку пустого массива и проверьте, проходит ли он.

Если `find_minmax()` в своем начале обращается к `numbers[0]`, программа будет завершаться аварийно, так как для пустого массива `numbers[0]` нет.

Обратите внимание: тщательное написание модульных тестов позволило выявить случай, при котором код не работает. Выходом может быть такое поведение: в начале `find_minmax()` проверять длину массива (`numbers.size()`), и если она нулевая, возвращаться из функции без изменения `min` и `max`. Тест тогда может проверять, что в результате вызова `min` и `max` не меняются. Возможно, стоит также учесть случай пустого массива в основной программе.

33. Внесите исправления в `histogram.cpp` (как минимум), пересоберите основную программу и тест, убедитесь, что они работают.

34. Сделайте коммит со всеми новыми файлами и отправите изменения на GitHub.

Важно: перед коммитом не забудьте сохранить каждый проект *(File → Save project)*.

35. **(Самостоятельно)** Выделите в модуль все функции для работы с SVG, опубликуйте коммит.

Домашнее задание

Результат выполнения нужно оформить отдельным коммитом на GitHub. Доработку следует делать с использованием функции, для который нужно добавить unit-тесты из не менее двух существенно отличающихся случаев.

Пример 1. По заданию нужно вместо запросить у пользователя цвет заливки столбцов, а если введен красный, отказаться его использовать и запросить повторно. Можно выделить функцию `bool check_color(string color)`, которая принимает введенную пользователем строку цвета и возвращает `true`, если цвет подходящий (не красный). Для нее можно придумать положительный тест, что `check_color("blue") == true` и отрицательный, что `check_color("red") == false`.

Пример 2. По заданию требуется добавить гистограмме заголовок *Гистограмма* по центру. Можно выделить функцию, которая принимает ширину и высоту изображения и вычисляет координаты надписи. Тестовые случаи — горизонтальное и вертикальное изображение (ширина больше или меньше высоты соответственно).

В вариантах примеры приведены для текстовых гистограмм, но решение должно выдавать SVG.

Вариант 1

Дайте пользователю возможность задавать произвольную ширину столбца гистограммы вместо 400. Считайте некорректной ширину менее 70, более 800 или менее трети количества чисел, умноженных на ширину блока (`BLOCK_WIDTH`) — предлагайте пользователю ввести ширину заново с указанием причины.

Вариант 2

Задавать автоматически яркость заливки каждого столбца гистограммы в градациях серого в зависимости от высоты столбца. Чем больше столбец, тем темнее заливка.

Сделать это можно, передавая цвет в параметр fill в формате `"#RGB"` (red, green, blue). `"#111"` — самый темный, `"#222"` — чуть менее темный, …, `"#EEE"` — практически белый, `"#FFF"` — белый. В лабораторной работе использовать диапазон цветов от `"#111"` для самого большого столбца до `"#999"` для самого маленького столбца. Поскольку используются градации серого, расчет сводится к вычислению только одного значения и дублированию этого значения в качестве цвета каждого из каналов (Red, Green, Blue). Для расчета цвета *i*-го столбца `bins[i]` использовать формулу `(10 - (bins[i] * 9) / max_count)`. По ней мы получаем значение цвета одного канала (от 1 до 9), который затем записываем три раза.

Пример с текстом вместо SVG:

`1|▮` — цвет `#999`
`5|▮▮▮▮▮` — цвет `#111`
`3|▮▮▮` — цвет `#555`

Вариант 3

Измените высоту изображения `IMAGE_HEIGHT = 700`. Дайте пользователю возможность задавать высоту столбца гистограммы. Если итоговая высота гистограммы больше `IMAGE_HEIGHT`, рассчитывать высоту столбца как `(IMAGE_HEIGHT / bins_count)`.

Вариант 4

Подсчитайте процент элементов, попавших в столбец, как целое двузначное число с `%` в конце и отображайте этот процент после столбца гистограммы с выравниванием:

```
 3|***    25%
 5|*****  42%
 4|****   33%
```

Вариант 5

Отображайте гистограмму зеркально по аналогии с заданием этого варианта в лабораторной работе № 1.

Вариант 6

После запроса количества столбцов запросите цвет для каждого столбца.

Вариант 7

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

Вариант 8

Запрашивать у пользователя размер шрифта. За размер шрифта отвечает атрибут `font-size`. Считать 12 значением по умолчанию. Не позволять вводить значения менее 8 и более 32. В этом случае предлагайте пользователю ввести значение заново с указанием причины.

Вариант 9

Запросите у пользователя ширину одного «блока» гистограммы `BLOCK_WIDTH`. Не позволяйте вводить ширину блока менее 3px и более 30px. В этом случае предлагайте пользователю ввести ее заново с указанием причины.

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.

Вариант 11

Добавьте рамку вокруг гистограммы, используя пунктирные линии. Для отрисовки пунктирной линии можно использовать стандартный элемент `<line>`, установив в нем атрибут `stroke-dasharray = '10 10'`

Вариант 12

Добавьте на ось подписей границы столбцов по аналогии с заданием этого варианта в лабораторной работе 1.

Вариант 13

Отображайте гистограмму вертикально с подписями снизу.

```
  *
  *
* *
* *
* * *

3 5 1
```

Предусмотреть расчет `IMAGE_HEIGHT` таким образом, чтобы вся гистограмма вмещалась в область рисунка.

Вариант 14

Разделяйте каждый столбец пунктирными линиями длиной `IMAGE_WIDTH`. Запрашивайте у пользователя шаблон пунктира. Шаблон пунктира задается в атрибуте `stroke-dasharray` блока `<line>` в виде `stroke-dasharray = '20 10'`, где 20 означает длину черточки, 10 - длину промежутка. У пользователя нужно запросить как длину черточки, так и длину промежутка.

Вариант 15

Добавьте горизонтальную шкалу под гистограммой по аналогии с заданием лабораторной работы 1. Шкалу нужно разбить на интервалы, размер которых вводит пользователь. Допустимы размеры от 2 до 9 `BLOCK_WIDTH`, при некорректном вводе печатайте сообщение со словом «ERROR» и завершайте работу программы. Под нулевой, первой и последней отметкой шкалы требуется напечатать соответствующие числа. Шкала должна быть во всю ширину гистограммы.

Вариант 16

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

Вариант 17

Задавать автоматически прозрачность заливки каждого столбца гистограммы в зависимости от высоты столбца. Чем больше столбец, тем темнее заливка. Сделать это можно, передавая процент прозрачности в параметр `fill-opacity` в формате `"0.7"`. 1 соответствует отсутствию прозрачности, 0 соответствует полной прозрачности (отсутствию цвета)

Для расчета прозрачности каждого i-го столбца `bins[i]` использовать формулу `(bins[i]) / max_count)`.

Пример:

`1|▮` — прозрачность 0.2
`5|▮▮▮▮▮` — прозрачность 1.0
`3|▮▮▮` — прозрачность 0.6

Вариант 18

Позволять пользователю делать оформление текста - подчеркивание, надчеркивание, ~~зачеркивание~~ текста. За оформление шрифта отвечает атрибут `text-decoration`. Сделать `'none'` значением по умолчанию. Допустимые значения: `none`, `underline`, `overline`, `line-through`. Проверять введенной пользователем значение, и если оно не соответствует допустимым, запрашивать значение заново, выдавая предупреждение.

Козлюк Д. А., Мохов А. С., Василькова П. Д. для кафедры Управления и информатики НИУ «МЭИ», 2021 г.