# Задание 5

## Расчёт статистических характеристик последовательности

Познакомимся (или вспомним) с довольно важными понятиями при обработке данных, и попутно (☺) закрепим навыки использования полиморфизма в коде.

За все время своего существования Человечество изобрело огромное количество способов анализа последовательности данных. Все мы, конечно, рассматривать не будем – эту деятельность мы оставим науке Статистика. Но часть из этих достижений представляют собой практическую ценность.

Представим себе, что нам нужно замерить время выполнения некоторого кода. Например, с целью сравнения двух разных алгоритмов вычисления и выбора оптимального. Казалось бы – сделать это легко. Мы просто засекаем время до начала выполнения алгоритма, потом запускаем код, который нужно оценить по скорости, и по окончанию его выполнения останавливаем замер времени.

Пример кода:

```
#include <chrono> // библиотека для работы с метками времени

auto startTime = std::chrono::system_clock::now(); // текущее время
runAlgorithm(); // запуск алгоритма
auto endTime = std::chrono::system_clock::now(); // текущее время
auto milliseconds = std::chrono::duration_cast<
     std::chrono::milliseconds>(
          endTime - startTime
     ); // получаем значение разницы в миллисекундах
```

Библиотеку <chrono> мы ещё не изучали, и код может выглядеть немного страшно. Но суть простая – мы получаем две метки времени, а потом вычисляем разницу в миллисекундах.

Пример, с которым можно поэкспериментировать приложен в материалах – см. `chrono_example.cpp`. В нем замеряются два разных алгоритма сортировки массива значений (std::vector). При этом перед запуском сортировки массив каждый раз случайно перемешивается с использованием генератора случайных чисел. Отдельно с функциональностью случайного перемешивания можно ознакомиться в примере `random_shuffle.cpp` из материалов.

Проблема такого рода замеров заключается в том, что наша операционная система (ОС) вовсе не планирует нам помогать в достижении поставленной задачи. Ей неведомо, что мы тут замерами с высокой точностью занимаемся – она может запустить обновление системы, спланировать поиск вирусов или любую другую ресурсоёмкую активность. Плюс распределение памяти для нашего приложения может работать от запуска к запуску более или менее удачно. В общем, есть множество причин, по которым наш код может работать не совсем стабильно.

Поэтому обычно практикуется подход, когда замеряется не единичный прогон той или иной функции, а выполняется несколько (несколько сотен или даже тысяч) прогонов и оценка статистики поведения. При этом рассматриваются сразу несколько статистических характеристик.

Например, мало информативно узнать, что среднее арифметическое всех замеров представляет собой значение (например) == 10.124 ms. Желательно посмотреть ещё минимальное и максимальное значения, а также получить оценку отклонений.

Вот статистикой мы и займёмся.

## Формулировка задания

На вход (стандартный ввод) приложению подаётся последовательность (заранее неизвестного размера) числовых значений. Приложение должно в ходе своей работы считать всю последовательность из стандартного ввода и вывести на экран набор следующих статистических характеристик:

- min – минимальное значение из последовательности

- max – максимальное значение из последовательности

- mean – арифметическое среднее, посчитанное на основе всех элементов последовательности

- std - среднеквадратическое отклонение

- [опционально] pct90 - 90-й процентиль

- [опционально] pct95 - 95-й процентиль

Пункты, помеченные как [опционально] представляю собой «задачу со звёздочкой» и могут не выполняться.

Остановка ввода последовательности предполагается путём передачи признака EOF (End Of File). В Windows это делается путём следующего набора команд: Ctrl+Z, Enter. В Linux это делается путём нажатия Ctrl+D.

Пример работы приложения:

```
> statistics
0 1 2 3 4 5 6 7 8 9 10
min = 0
max = 10
mean = 5
std = 3.162277
pct90 = 9
pct95 = 10
```

Стоит заметить, что результаты могут отличаться в зависимости от реализации. Так, например, некоторые реализации могут выдать значение pct95 = 9.5. Или значения std, отличные от того, что указано выше, за счёт иного округления результата.

## Уточнения по реализации

- расчёт каждого вида статистики должен представлять собой отдельный класс
- , представляющий собой наследника класса IStatistics
- и реализующий чисто виртуальную функции последнего

Макет приложения можно посмотреть в файле `statistics.cpp` из материалов.

## Итоговые требования

1. создать приложение расчёта статистики
2. для сборки использовать CMake
3. выгрузить результат на github.com в свой аккаунт

В Чат с преподавателем в Личном кабинете отправить:

1. ссылку на репозиторий с исходниками приложения