# Домашнее задание №4

## Общие требования

1. Наличие интерактивного диалогового интерфейса для проверки корректности разработанной программы.
2. Корректное завершение программы — как в случае штатного выхода, так и в случае невосстановимых ошибок (без утечек и без использования функций мгновенного завершения программы `exit`, `abort`, `std::terminate` и пр.).
3. Использование высокоуровневых подходов (например `std::copy` вместо `memcpy`, `std::string` вместо `char*`, исключений вместо кодов возврата и т.д.).
4. Использование стандартных библиотек и функций языка C++ предпочтительнее, чем использование библиотек и функций языка C (`std::copy` вместо `memcpy`, `std::abs` вместо `abs`, `cstring` вместо `string.h` и т.д.).
5. Логичная и удобная структура проекта, где каждая единица (файл/библиотека) обладает своей единой зоной ответственности (каждый класс — в своих файлах .h и .cpp, диалоговые функции и `main` — в своих).
6. Наличие средств автосборки проекта. Предпочтительны инструменты, работающие «поверх» Makefile, например: CMake, qmake и др. Использование Makefile напрямую не желательно, но допустимо.
7. Статический анализ кода с использованием средств, встроенных в IDE (например, в VS2019: Analyze -> Run Code Analysis, см. также Project -> Properties -> Configuration Properties -> Code Analysis -> Microsoft -> Active Rules), или внешних инструментов (Sonarqube + extensions, Clang Static Analyzer и др.). Обязательным является знакомство с инструментом, исправление всех замечаний или обоснование, почему конкретное замечание исправить нельзя.
8. Динамический анализ на утечки памяти, встроенный инструментарий в IDE / библиотеки (Пр., VS2019) или внешние инструменты (valgrind, Deleaker и т.п.). Отсутствие утечек памяти и прочих замечаний анализатора.
9. Не «кривой», не избыточный, поддерживаемый и расширяемый код (разумная декомпозиция, DRY, корректное использование заголовочных файлов и т.п.).
10. Стандарт языка C++20 или C++23.

## Требования задачи

1. Разработка модульных тестов для классов, полное (не менее 80%) покрытие методов классов модульными тестами.
2. Использование фреймворков для тестирования решения (gtest, catch2 и пр.). Тестирование встроенными средствами языка запрещено.
3. Структура решения: проект со статически линкуемой библиотекой с классом, проект консольного приложения для диалоговой отладки, проект для модульного тестирования.
4. Использование контейнеров из STL, умных указателей и т.п. возможно только для «профессионалов» в C++, подавляющему большинству не рекомендуется.
5. Документирование всех публичных методов класса с использованием doxygen.
6. Документация метода должна включать в себя как минимум описание всех аргументов метода, описание возвращаемого значения и описание всех исключений, которые могут быть выброшены в этом методе (для каждого указать тип исключения, и в каких случаях оно может возникнуть).
7. Корректность состояния класса, отсутствие избыточности, наличие необходимых конструкторов и деструктора, корректность сигнатуры методов, сохранение семантики перегружаемых операторов и корректность их сигнатуры, сохранение семантики работы с потоками ввода/вывода для перегружаемых операторов сдвига.

## Порядок выполнения

### Простой класс

Выполнить разработку простого класса и его методов, а также диалоговых функций для проверки класса. Простой класс должен обладать как минимум следующими методами:

- пустой конструктор (явный или неявный);
- инициализирующие конструкторы, перечисленные в задании;
- методы получения значений атрибутов класса (геттеры);
- методы изменения значений атрибутов класса (сеттеры);
- методы ввода и вывода состояния класса в входной/выходной поток;
- прочие методы, перечисленные в задании.

**Сложный класс**

Выполнить метод разработку сложного класса и его методов, а также диалоговых функций для проверки класса. В качестве вектора использовать статический массив. Сложный класс должен обладать как минимум следующими методами:

- конструктор по умолчанию (явный или неявный);
- инициализирующие конструкторы, перечисленные в задании;
- методы ввода и вывода состояния класса в входной/выходной поток;
- прочие методы, перечисленные в задании.

**Перегрузка операторов**

Модифицировать классы, перегрузив для них следующие операторы (внутри операторов целесообразно вызывать методы, реализованные в предыдущих пунктах):

- = — для копирования экземпляра сложного класса;
- >> — для ввода экземпляра простого класса из входного потока;
- << — для вывода экземпляра простого класса в выходной поток;
- >> — для ввода экземпляра сложного класса из входного потока;
- << — для вывода экземпляра сложного класса в выходной поток;
- прочие операторы, определённые в задании (в скобках перед описанием метода).

**Динамическая память**

Модифицировать сложный класс, используя динамическую память для вектора (использование STL и умных указателей для вектора недопустимо в учебных целях). Дополнить класс необходимыми методами:

- копирующий конструктор;
- перемещающий конструктор;
- перемещающий оператор “=”;
- деструктор.

**Прикладная программа**

Реализовать прикладную программу для проверки корректности разработанных классов.

**Тестирование**

В процессе выполнения каждого пункта задания (кроме прикладной программы) необходимо реализовать тесты для разрабатываемых классов. Тесты должны покрывать все методы разработанных в неко-

тором пункте классов. При переходе между пунктами тесты необходимо обновлять в соответствии с внесёнными изменениями.

### Документация

В процессе выполнения каждого пункта вести документацию разработанных классов (см. требования), особенно перед передачей кода другому студенту.

## Условие

### Простой класс

**Точка** — определяет точку на плоскости, представляется совокупностью двух координат $x$ и $y$, представленных числом с плавающей запятой.

Методы простого класса (помимо общих):

- создание экземпляра класса с инициализацией двумя координатами;
- создание экземпляра класса с инициализацией массивом из двух координат;
- вычисление расстояния между двумя точками;
- `+` — покоординатное суммирование двух точек;
- `-` — покоординатное вычитание двух точек;
- `/` — покоординатное деление точки на число;
- вращение точки на 90 градусов относительно начала координат по часовой стрелке.

### Сложный класс

**Фигура** — определяется количеством вершин и массивом вершин (точек) на плоскости.

Методы сложного класса (помимо общих):

- создание экземпляров класса с инициализацией значением единственной вершины;
- создание экземпляров класса с инициализацией количеством вершин и массивом координат вершин;
- определение “центра тяжести” многоугольника;
- `+=` — добавление вершины многоугольника;
- `[]` — получение вершины многоугольника, заданной её номером (возврат по ссылке);

- выполнение операции поворота многоугольника относительно заданной точки на определённый угол, кратный 90 градусам, против часовой стрелки;
- перемещение многоугольника в направлении и на величину вектора, построенного из точки {0,0} в заданную.