

### **Общие требования:**

- 1) Проект на git'e.
- 2) Наличие интерактивного диалогового интерфейса для проверки корректности разработанной программы.
- 3) 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.).
- 4) Проверка корректности ввода (работа только через потоки C++, т.е. без `scanf/printf/fscanf/fprintf`). В случаях ошибок формата ввода запрос повторного ввода данных. В случае невосстановимых ошибок ввода-вывода завершение программы.
- 5) Использование средств языка C++ для работы с динамической памятью – операторов `new` и `delete` (`malloc`, `calloc`, `realloc`, `free` запрещены).
- 6) Использование исключений для обработки ошибочных ситуаций (вместо кодов возврата).
- 7) Предпочтительно использование стандартных библиотек и функций языка C++ вместо библиотек и функций языка C (`std::copy` вместо `memcpy`, `std::abs` вместо `abs`, `cstring` вместо `string.h` и т.д.).
- 8) 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).
- 9) Наличие средств автосборки проекта (желательно CMake, `qmake` и прочие, работающие "поверх" Makefile; использование самописного Makefile нежелательно, но допустимо).
- 10) Статический анализ кода, встроенный инструментарий в IDE (пр. VS2019: Analyze->Run Code Analysis, см. также Project -> Properties -> Configuration Properties -> Code Analysis -> Microsoft -> Active Rules) или внешние инструменты (Sonarqube + extensions, Clang Static Analyzer и др.) (обязательно знакомство с инструментом, исправление всех замечаний или обоснование в комментариях почему конкретное замечание исправить нельзя).
- 11) 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.
- 12) Не "кривой", не избыточный, поддерживаемый и расширяемый код (разумная декомпозиция, DRY, корректное использование заголовочных файлов и т.п.).
- 13) Стандарт языка C++20 (рекомендуется). Допустим C++17 (если почему-то нет C++20).

### **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 (дождитесь задачи №3).
- 5) Документирование всех публичных методов класса с использованием doxygen. 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).
- 6) 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 работы:**

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

- конструктор по умолчанию (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 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\*.** Передать разработанный класс другому студенту, получить взамен чужой класс и реализовать прикладную программу на основе полученного класса. Внесение существенных изменений в чужой класс не допускается. Возможно исправление чужих багов и внесение изменений в интерфейс класса, если вам не повезло с качеством полученного кода или нужно получать больше информации из класса.

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

**Ревью\*.** В процессе выполнения пунктов «Простой класс» - «Динамическая память», имеется возможность записаться на ревью, в процессе которого вы отдадите свой код случайному студенту и взамен получите код от 3-х других студентов. Ваша задача в процессе ревью проанализировать код ваших коллег и оставить к нему замечания, которые по вашему мнению помогут коллегам улучшить их работу. Взамен вы получите 3 ревью на свою работу, которые помогут вам улучшить её. Ревью будет выполняться в несколько волн, на каждую из которых вы можете отправить работу по любому из пунктов (отправка незаконченной части некоторого пункта нежелательна, но допустима; лучше отправляйте логически завершённый кусок кода).

**Примечание:** крайне рекомендуется делать и сдавать все пункты строго по очереди. Однако, если вы уверены в своих силах, допускается делать и сразу финальный вариант программы.