

### 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) 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.
- 4) Использование средств языка C++ для работы с динамической памятью – операторов `new` и `delete` (`malloc`, `calloc`, `realloc`, `free` запрещены).
- 5) Использование исключений для обработки ошибочных ситуаций (вместо кодов возврата).
- 6) 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.).
- 7) 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).
- 8) Наличие средств автосборки проекта (желательно CMake, qmake и прочие, работающие “поверх” Makefile; использование самописного Makefile нежелательно, но допустимо).
- 9) Не "кривой", не избыточный, поддерживаемый и расширяемый код (разумная декомпозиция, DRY, корректное использование заголовочных файлов и т.п.).
- 10) Стандарт языка C++20 (рекомендуется). Допустим C++17 (если почему-то нет C++20).

### **Требования задачи:**

1) Логичная структура решения, разделение на зоны ответственности (отдельные компоненты вынести в отдельные библиотеки), следование принципам SOLID.

2) Корректность состояния классов, 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.

3) Строгое следование схемам MVC или MVP при проектировании программы. То есть классы программы необходимо разделить на 3 категории (библиотеки):

- Классы внутренней логики программы (model);
- классы отображения (view);
- классы управления/представления (controller/presenter).

Допускаются объединение view и controller/presenter в один компонент.

4) Основной язык программирования — C++. Допускается использование Java или C# на ваш выбор.

### **Порядок выполнения работы:**

**UML.** Выполнить проектирование диаграммы классов реализуемой программы в нотации UML (рекомендуется использовать специализированный редактор, например, modelio) и разработку соответствующих диаграмме прототипов классов (хедеров). Допустима генерация UML-диаграммы из кода или кода из UML-диаграммы, однако в любом случае диаграмма классов и их прототипы должны полностью соответствовать друг другу.

**Реализация.** Выполнить реализацию всех классов, отвечающих за логику программы. Для проверки реализованных классов использовать тесты или простую проверочную main функцию.

**Прикладная программа.** Реализовать прикладную программу для работы с разработанными классами. Возможно выполнение пункта в 1 из 3 вариантов:

- диалоговая программа (макс. балл за задачу не более 80/100);
- псевдографическая программа (обязательно интерактивное навигирование при помощи клавиш без нажатия Enter, например, с использованием библиотеки ncurses) (макс. балл за задачу не более 90/100);
- графическая программа.

**Примечание:** Пункты задания от «UML» до «Прикладная программа» необходимо выполнять и сдавать строго в указанной последовательности. Приступать к выполнению следующего пункта до выполнения предыдущего крайне не рекомендуется, так как при обнаружении ошибок на более ранних этапах придётся переделывать всю программу целиком от первых и до последних этапов.