**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) Использование высокоуровневых подходов (например std::copy вместо memcpy, std::string вместо char*, исключений вместо кодов возврата 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 (CMake, qmake и прочие, работающие “поверх” Makefile).

7) 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 д.р.) (обязательно знакомство с инструментом, исправление всех замечаний или обоснование в комментарии 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.

11) Сдача работ проводится только в интегрированной среде разработки (IDE) или редакторе с настроенным LSP.

**Требования задачи:**

1) На выбор предоставляется две категории вариантов: простые и сложные. Сложные варианты обладают дополнительным множителем для баллов, получаемых за пункты задания «UML», «Реализация», «Прикладная программа», «Тестирование» и «Многопоточность». Множитель, указанный в варианте, является ориентировочным и может меняться по усмотрению преподавателя. При выборе варианта также обратите внимание на то, что для простых вариантов отсутствуют дополнительные задания «динамическая загрузка» и «плагины».

2) Разработка модульных тестов для классов, удовлетворительное (не менее 50%) покрытие методов классов модульными тестами.

3) Использование фреймворков для тестирования решения (gtest, catch2 и пр.). Тестирование встроенными средствами языка запрещено.

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

5) Документирование всех публичных методов шаблонного и расширяемого классов (для простых вариантов — всех классов) 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 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.

7) 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.

8) Динамический анализ (thread sanitizer и т.п.) многопоточной программы на отсутствие состязаний (race condidion). Наличие состязаний в финальной версии программы не допускается.

9) Основной язык программирования — C++. **По согласованию с преподавателем**, принимающим лабораторные работы, допускается использование Java или C#.

**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.

**Шаблон**. Выполнить реализацию указанного в задании контейнера в виде шаблонного класса. Разработанный шаблонный класс должен обладать основными методами (вставка, поиск, удаление и т.д.) и предоставлять полноценный интерфейс доступа при помощи итераторов. Класс итератора должен соответствовать выбранной категории (random access, bidirectional, forward или прочие). Выбор категории итератора необходимо обосновать. В учебных целях, при реализации своего шаблонного контейнера, запрещается использовать контейнеры STL. Использовать умные указатели можно.

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

- диалоговая программа (до 2 баллов);
- псевдографическая программа (обязательно интерактивное навигирование при помощи клавиш без нажатия Enter, например, с использованием библиотеки ncurses) (до 4 баллов);
- графическая программа (до 6 баллов).

**Тестирование**. Разработать тесты для классов логики. Написание unit тестов для интерфейса пользователя (контроллера/представления и отображения) не требуется.

**Документация**. Составить документацию для расширяемого и шаблонного классов (см. требования). Для простых вариантов вместо этого составить документацию для всех разработанных классов логики. Документировать классы пользовательского интерфейса не требуется.

**Многопоточность**. Модифицировать программу таким образом, чтобы указанный в задании алгоритм выполнятся параллельно в несколько потоков. Необходимо использовать предоставляемые языком примитивы синхронизации для избежания состязаний. Сравнить скорость выполнения одной и той же операции в однопоточном и многопоточном режимах в зависимости от объёма данных (построить график).

**Динамическая загрузка***. Обеспечить динамическую загрузку для наследников расширяемого (см. в задании) класса из динамических библиотек (.so/.dll). При запуске программа должна загружать все библиотеки, представленные в некотором каталоге, считывать их

метаинформацию и использовать при дальнейшей работы. Перенести всех реализованных наследников расширяемого класса в динамические библиотеки в качестве примера.

**Плагин***. Обменяться своим кодом с другими студентами и выполнить реализацию своего плагина для кода другого студента. Плагин представляет из себя новог наследника расширяемого класса, собранного в виде динамической библиотеки. Модифицировать код другого студента не допускается.

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

# Вариант 22 Сетевой сервис

Разработать приложение, позволяющее организовать работу по учёту передачи информации в сети: почта, файл или гипертекст. Все серверы в сети имеют сетевой адрес (IP-адрес, например, 194.67.66.175) и сетевое имя (www.mephi.ru). Информация о предоставляемом сервисе хранится в специальном описателе.

Описатель *почты* – признак «приём или передача»; сетевой адрес отправителя или получателя; дата и время связи; объём информации (в MB).

Описатель *файла* – признак «приём или передача»; сетевой адрес отправителя или получателя; дата и время связи, продолжительность связи (в минутах), объём информации (в MB).

Описатель *сети* - сетевой адрес отправителя, дата и время связи, продолжительность связи (в минутах), объём выходного и входного трафика (в MB).

Информация обо всём сетевом сервисе сведена в **просматриваемую таблицу**[1] – «таблицу связи», каждый элемент которой содержит сетевой адрес абонента и указатель на описатель сервиса. Абонент может выступать как получателем, так и отправителем информации.

Полная информация о сетевом сервисе хранится в описателе сервера, который содержит: собственный сетевой адрес и сетевое имя сервиса, тариф оплаты минуты связи, тариф оплаты передачи одного MB и «таблицу связи».

Обеспечить выполнение следующих операций:

- Для описателя сервера:
  - получить (вернуть в качестве результата) собственный сетевой адрес; получить тариф минуты связи; получить тариф передачи одного MB;
  - включить элемент в таблицу по сетевому адресу абонента;
  - найти элемент в таблице по сетевому адресу абонента и времени услуги (дата и время сеанса связи);
  - показать содержимое таблицы.
- Для любого сетевого сервиса:
  - получить (вернуть в качестве результата) тип сервиса;
  - получить тип связи («приём» или «передача»);
  - получить время (дату и время связи) оказанной услуги;
  - получить сетевой адрес отправителя или получателя;
  - получить время связи; получить объём информации;
  - рассчитать стоимость сервиса (сумма платы по объёму информации и по времени связи).
- Для приложения:
  - записать информацию об оказанных услугах для указанного абонента;
  - вывести информацию об оказанных услугах для указанного абонента (суммарный объём и время связи по каждому типу сервиса);
  - рассчитать стоимость оказанных услуг всех видов сервиса для указанного абонента;
  - рассчитать баланс принятой и отправленной информации для всех абонентов сети, используя класс-итератор[2].

1. Шаблонный класс — просматриваемая таблица.
2. Указанную операцию реализовать в многопоточном режиме. Каждая группа сервисов обрабатывается в отдельном потоке.