Модуль 2. Вопросы.

1. **Move семантика**

   a. **Перемещение**
   Что понимается под копированием объекта в C++? Что происходит при копировании? Что понимается под перемещением объекта в C++? Что происходит при перемещении? Стандартная функция `std::move`. В чём преимущества перемещения над копированием? Перемещение объекта в функцию, если функция принимает объект по значению. Как происходит перемещение объектов встроенных типов и объектов классов? Что происходит при перемещении объекта класса `std::vector`? Перемещение объекта при возврате из функции и взаимодействие такого перемещения с RVO.

   b. **lvalue-выражения и rvalue-выражения**
   Что такое выражение? Тип выражения и категория выражения. Что такое lvalue-выражение? Что такое rvalue-выражение? Приведите примеры lvalue и rvalue выражений.

   c. **lvalue-ссылки и rvalue-ссылки**
   Что такое lvalue-ссылки, а что такое rvalue-ссылки, в чём разница? Зачем нужно разделение выражений на lvalue и rvalue. Перегрузка по категории выражения. Уметь написать функцию, которая печатает категорию переданого ей выражения. Какую категорию имеет выражение, состоящее только из одного идентификатора – rvalue-ссылки? Что на самом деле делает функция `std::move`?

   d. **Особые методы, связанные с перемещением**
   Конструктор перемещения и оператор присваивания перемещения. Создание класса, с пользовательским конструктором перемещения и пользовательским оператором перемещения. Правило пяти. Идиома Move-and-Swap.

   e. **Умный указатель `std::unique_ptr`**
   Зачем нужен умный указатель `std::unique_ptr`? В чём его преемущество по сравнению с обычными указателями? Реализация `std::unique_ptr`. Шаблонная функция `std::make_unique`. Перемещение объектов типа `unique_ptr`. Передача таких указателей в функции. Нужно уметь писать класс, аналогичный классу `std::unique_ptr`. Циклические ссылки и `std::unique_ptr`.

   f. **Умный указатель `std::shared_ptr`**
   Зачем нужен умный указатель `std::shared_ptr`? В чём его преемущество по сравнению с обычными указателями и с `std::unique_ptr`? Шаблонная функция `std::make_shared`. Как схематически устроен указатель типа `std::shared_ptr`. Циклические ссылки и `std::shared_ptr`. Умный указатель `std::weak_ptr`.

2. **Сборка**

   a. **Раздельная компиляция**
   Что такое файл исходного кода и исполняемый файл? Этап сборки программы: препроцессинг, ассемблирование, компиляция и линковка. Что такое заголовочные файлы (header-файлы)? Что делает директива препроцессора `#include`? Что такое единица трансляции? Компиляция программы с помощью `g++`. Опции компиляции `-E`, `-S` и `-c`. Что такое раздельная компиляция и в чём её преемущества?

   b. **Библиотеки**
   Что такое библиотека? Виды библиотек: header-only библиотеки, open-source библиотеки, статические библиотеки, динамические библиотеки. В чём различия между этими видами библиотек? В чём преимущества и недостатки каждого из видов библиотек? Как подключить библиотеки к своему проекту?

   c. **Статические библиотеки**
   Как создать статическую библиотеку? Как подключить статическую библиотеку? Опции компилятора `-I`, `-L` и `-l`. Характерные расширения файлов статических библиотек на Linux и Windows.

   d. **Динамические библиотеки**
   В чём главная разница между статическими и динамическими библиотеками? Как создать динамическую библиотеку? Как подключить динамическую библиотеку? Характерные расширения файлов динамических библиотек на Linux и Windows.

   e. **Bash-скрипты. Основы Make. Основы CMake**
   Использование Bash-скриптов(на Linux) или Bat-скриптов(на Windows) для оптимицации сборки программы. Основы Make и использование этой утилиты для сборки простейшей программы. Основы CMake и использование этой утилиты для сборки простейшей программы.

3. **Событийно-ориентированное программирование**

   a. **Библиотека SFML. Отрисовка на экран**
   Класс `sf::RenderWindow`. Классы `sf::CircleShape` и `sf::RectangleShape` и основные методы этих классов (в частности метод `setOrigin`). Системы координат SFML (координаты пикселей, глобальная система координат, локальные системы координат). Методы `mapPixelToCoords` и `mapCoordsToPixel` класса `sf::RenderWindow`. Основной цикл программы. Двойная буферизация. Методы `clear`, `draw` и `display` класса `sf::RenderWindow`.

   b. **Событийно-ориентированное программирование**
   Понятие событий. Событийно-ориентированное программирование. Очередь событий. Цикл обработки событий.

   c. **Событийно-ориентированное программирование в библиотеке SFML**
   Класс `sf::Event` и цикл обработки событий в SFML. События SFML: `Closed`, `Resized`, `KeyPressed`, `KeyReleased`, `MouseButtonPressed`, `MouseButtonReleased`, `MouseMoved`.

   Функции `sf::Keyboard::isKeyPressed` и `sf::Mouse::isButtonPressed`. Чем использование этих функций отличается от использования событий?

4. **Наследование**

   a. **Основы наследования**
   Наследование в языке `C++`. Добавление новых полей и методов в наследуемый класс. Модификатор доступа `protected`. Публичное и приватное наследование. Имеют ли друзья базового класса доступ к приватным полям класса-наследника? Порядок вызовов конструкторов при создании экземпляра класса-наследника. Как сделать так, чтобы вызывалась необходимая перегрузка конструктора базового класса при создании экземпляра класса-наследника.

   b. **Перегрузка и переопределение методов в классе наследнике**
   Перегрузка методов в базовом и наследуемом классе. Как проходит отбор перегрузки? Переопределение методов в классе-наследнике. Разница между перегрузкой и переопределением. Вызов методов базового класса из класса наследника.

   c. **Приведение типов**
   Присваивание объекта класса наследника объекту базового класса (`base = derived`). Срезка. Строение объекта класса-наследника. Размер объекта класса-наследника. Empty base optimisation. Присваивание указателя на объект класса наследника указателю базового класса (`pbase = pderived`). Иерархия наследования. Использование `static_cast` для перемещения по иерархии наследования. В каких случах это может привести к неопределённому поведению?

   d. **Множественно наследование**
   Строение объекта класса наследника при обычном (не виртуальном) множественном наследовании. Сдвиг указателей при присваивании в случае множественного наследования. Ромбовидное наследование. Как в языке C++ решается проблема ромбовидного наследования?

5. **Полиморфизм**

   a. **Основы полиморфизма**
   Статический полиморфизм в языке C++ и других языках. Динамический полиморфизм и его примеры в других языках(например, в языке Python). Для чего нужен полиморфизм?

   b. **Основы динамического полиморфизма в языке C++**
   Указатели на базовый класс, хранящие адрес объекта наследуемого класса (`Base* pbase = &derived`). Методы какого класса будут вызываться, если мы будем вызывать их через такой указатель? Виртуальные функции. Виртуальный деструктор. Ключевые слова `override` и `final`. Уметь написать пример использования полиморфизма (например, вектор указателей типа `Base*`). Приватность и виртуальные функции.

   c. **Абстрактные классы**
   Чистая виртуальная функция. Абстрактный класс. Интерфейс. Наследование от интерфейса. Ошибка pure virtual call.

   d. **dynamic_cast**
   Полиморфные типы. Использование `static_cast` для приведения типов и указателей на типы в иерархии наследования. Когда использование `static_cast` может привести к неопределённому поведению? Оператор `dynamic_cast`. Чем он отличается от `static_cast` и в каких случая он используется? Что

происходит если `dynamic_cast` не может привести тип (рассмотрите случай приведения указателей и случай приведения ссылок)?.

e. **Реализация механизма виртуальных функций**
Скрытое поле - указатель на таблицу виртуальных функций. Сколько таблиц виртуальных функций хранится в памяти при работе программы? Как устроены таблицы виртуальных функций?

6. **Паттерны проектирования**

a. **Основы**
Что такое паттерны проектирования? Зачем нужно использовать паттерны проектирования. UML-диаграммы. Отношения между классами: наследование, композиция, агрегация, ассоциация и дружественность.

b. **Стратегия**
Паттерн стратегия. Какие преемущества даёт использование этого паттерна? Пример использования этого паттерна.

c. **Машина состояний**
Паттерн состояние. Паттерн машина состояний. Какие преемущества даёт использование этих паттернов перед использованием объекта перечисляемого типа? Пример использования этих паттернов.

7. **Обработка ошибок**

a. **Методы обработки ошибок.**
Классификация ошибок. Ошибки времени компиляции, ошибки линковки, ошибки времени выполнения, логические ошибки. Виды ошибок времени выполнения: внутренние и внешние ошибки. Методы борьбы с ошибками: макрос `assert`, использование глобальной переменной(`errno`), коды возврата и исключения. Преемущества и недостатки каждого из этих методов. Какие из этих методов желательно использовать для внутренних ошибок, а какие для внешних?

b. **assert**
Макрос `assert` и его применения для обнаружения ошибок.

c. **Коды возврата и класс `std::optional`**
Обработка ошибок с помощью кодов возврата. Примеры стандартных фуцнкий, использующих коды возврата. Класс `optional` из стандартной библиотеки. Методы класса `optional`:

- Конструкторы
- Методы `value`, `has_value`, `value_or`.
- Унарные операторы `*` и `->`
- Оператор преобразования к значению типа `bool`.

Для чего можно применять `std::optional`? Использование класса `optional` для обработки ошибок с помощью кодов возврата.

d. **Исключения.**
Зачем нужны исключения, в чём их преимущество перед другими методами обработки ошибок? Оператор `throw`, аргументы каких типов может принимать данный оператор. Что происходит после достижения программы оператора `throw`. Раскручивание стека. Блок `try-catch`. Что произойдёт, если выброшенное исключение не будет поймано? Стандартные классы исключений: `std::exception`, `std::runtime_error`, `std::bad_alloc`, `std::bad_cast`, `std::logic_error`. Почему желательно ловить стандартные исключение по ссылке на базовый класс `std::exception`? Использование `catch` для ловли всех типов исключений. Использование исключений в кострукторах, деструкторах, перегруженных операторах. Спецификатор `noexcept`. Гарантии безопасности исключений. Исключения при перемещении объектов. `move_if_noexcept`. Идиома `copy and swap`.