Модуль 2. Вопросы.

1. **Универсальные ссылки**
   Правила свёртки ссылок. Универсальные ссылки, чем они отличаются от lvalue и rvalue ссылок? Реализация функции `std::move`. Идеальная передача. Функция `std::forward`.

2. **Раздельная компиляция**
   Что такое файл исходного кода и исполняемый файл. Этап сборки программы: препроцессинг, ассемблирование, компиляция и линковка. Директивы препроцессора `#include` и `#define`. Компиляция программы с помощью `g++`. Header-файлы. Раздельная компиляция. Преимущества раздельной компиляции. Статические библиотеки и их подключение с помощью компилятора `gcc`. Динамические библиотеки и их подключение.

3. **Событийно-ориентированное программирование и библиотека SFML**
   Библиотека SFML. Класс `sf::RenderWindow`. Системы координат SFML (координаты пикселей, глобальная система координат, локальные системы координат). Методы `mapPixelToCoords` и `mapCoordsToPixel`. Основной цикл программы. Двойная буферизация. Понятие событий. Событийно-ориентированное программирование. События SFML: `Closed`, `Resized`, `KeyPressed`, `KeyReleased`, `MouseButtonPressed`, `MouseButtonReleased`, `MouseMoved`. Очередь событий. Цикл обработки событий.

4. **Наследование.**
   Наследование в языке `C++`. Добавление новых полей и методов в наследуемый класс. Вызов конструкторов наследуемого класса. Модификатор доступа `protected`. Переопределение методов. Чем отличается переопределение от перегрузки? Ключевые слова `override` и `final`. Чистые виртуальные функции. Абстрактные классы и интерфейсы. Срезка объектов. Множественное наследование. Виртуальное множественное наследование.

5. **Полиморфизм.**
   Полиморфизм в `C++`. Указатели на базовый класс, хранящие адрес объекта наследуемого класса. Виртуальные функции. Реализация механизма виртуальных функций. Таблица виртуальных функций. Виртуальный деструктор. Статический и динамический типы. Хранение объектов разных динамических типов в векторе. Оператор `dynamic_cast`. В каких случая он используется? Что происходит если `dynamic_cast` не может привести тип? Рассмотрите случай приведения указателей и случай приведения ссылок. Использование `static_cast` для приведения типов и указателей на типы в иерархии наследования.

6. **Функциональные объекты**
   Указатели на функции в алгоритмах STL. Функторы. Стандартные функторы: `std::less`, `std::greater`, `std::equal_to`, `std::plus`, `std::minus`, `std::multiplies`. Основы лямбда-функций. Стандартные алгоритмы STL, принимающие функциональные объекты. Тип обёртка `std::function`. Шаблонная функция `std::bind`.

7. **Лямбда-функций**
   Лямбда-функций. Объявление лямбда-функций. Передача их в другие функции. Преимущества лямбда-функций перед указателями на функции и функторами. Использование лямбда функций со стандартными алгоритмами `std::sort`, `std::transform`, `str::copy_if`. Лямбда-захват. Захват по значению и по ссылке. Захват всех переменных области видимости по значению и по ссылке. Объявление новых переменных внутри захвата.

8. **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: `assert`, использование глобальной переменной, коды возврата и исключения. Преемущества и недостатки каждого из этих методов. Какие из этих методов желательно использовать для внутренних ошибок, а какие для внешних?

9. **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.

10. **Реализация вектора.**
Реализация своего вектора mipt::Vector<T> (аналога std::vector<T>). Нужно также предусмотреть итераторы этого вектора: mipt::Vector<T>::iterator, а также константные и обратные итераторы.
Методы такого вектора:

    - Конструктор по умолчанию
    - Конструктор, принимающий количество элементов
    - Конструктор, принимающий количество элементов и значение элемента
    - Конструктор от std::initializer_list.
    - Конструктор копирования
    - Конструктор перемещения
    - Деструктор
    - Оператор присваивания копирования
    - Оператор присваивания перемещения
    - Оператор взятия индекса (operator[])
    - Метод at, аналог метода at класса std::vector
    - Методы size, capacity, empty, reserve, resize, shrink_to_fit.
    - Методы push_back, emplace_back, pop_back.
    - Методы для работы с итераторами begin, end, rbegin, rend.

    Безопасность относительно исключений у такого вектора.

11. **Система типов языка C++.**
Система типов языка C++. Встроенные типы, массивы, структуры, объединения, перечисления, классы, указатели, ссылки, функциональные объекты (функции, указатели и ссылки на функции, функторы, лямбда-функции), указатели на члены класса, битовые поля. Вывод типа выражения с помощью decltype. Различие вывода с помощью decltype, auto и вывода шаблонных аргументов. Разложение типов (type decay) и когда он происходит.

12. **Приведение типов**
В чём недостатки приведения в стиле C? Оператор static_cast и в каких случая он используется. Операторы reinterpret_cast и const_cast и в каких случая они используется.

13. **Классы std::any, std::optional и std::variant**
Класс std::any. Функция std::any_cast. Класс std::optional. Методы класса std::optional:

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

    Для чего можно применять std::optional?

    Класс std::variant. Функции для работы с std::variant:

    - std::get
    - std::holds_alternative
    - std::visit

    Для чего можно применять std::variant?

14. **Вычисления на этапе компиляции.** constexpr
Вычисление на этапе компиляции. В чём преимущества вычисления на этапе компиляции по сравнению с вычислением на этапе выполнения. Ключевое слово constexpr. Что означает constexpr при объявлении переменной? Что означает constexpr при определении функции? Разница между const и constexpr. Ключевые слова consteval и constinit. static_assert.

15. **Вычисления на этапе компиляции. Шаблонное метапрограммирование.**
Полная специализация шаблона. Частичная специализация шаблона. Что такое шаблонные метафункции и зачем они нужны? Использование специализации шаблона для написание следующих метафункций:

    - `IsInt` - проверяет, является ли тип `T` типом `int`.
    - `IsIntegral` - проверяет, является ли тип `T` целочисленным типом.
    - `IsPointer` - проверяет, является ли тип `T` указателем.
    - `IsSame` - проверяет, являются ли 2 типа `T1` и `T2` одинаковыми.
    - `RemovePointer` - если тип `T` является указателем, то возвращает тип того, на что такой указатель указывает (то есть убирает одну "звёздочку"у типа).
    - `IsHasBegin` - проверяет, есть ли у типа `T` метод begin.

    Что такое концепты, как их использовать и зачем они нужны?