Модуль 5. Дополнительные темы C++. Вопросы.

1. **Паттерны проектирования с использованием шаблонов**

   a. **Класс `any` из стандартной библиотеки**
   Класс `any`. Функция `any_cast`.

   b. **Класс `variant` из стандартной библиотеки**
   Функции для работы с `variant`:

   - `get`
   - `holds_alternative`
   - `visit`

   Для чего можно применять `variant`? Динамический полиморфизм при использовании класса `variant`.

   c. **Type erasure**
   Паттерн Type erasure (Стирание типа). Реализация своего класса `any` при помощи паттерна Type erasure.

2. **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`.

1. **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`.

2. **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.

3. **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.

4. **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.

5. **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.

6. **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`.

7. **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?

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