C++. Модуль 1. Вопросы.

1. **Ссылки, вектор и строка**

   a. **Пространства имён**
   Пространство имён: что такое и зачем нужно. Директива `using namespace`. `using`-объявление. Пространство имён `std`.

   b. **Ссылки**
   Что такое ссылки? Различие ссылок и указателей. Константные ссылки. Три типа передачи аргументов в функцию: передача по значению, передача по ссылке и передача по константной ссылке. Преимущества/недостатки каждого метода. Возврат ссылки из функции. Висячие ссылки.

   c. **Перегрузка функций**
   Сигнатуры функций. Перегрузка функций. Выбор функции при перегрузке.

   d. **Класс `std::string`**
   Стандартная строка `std::string`. Преимущества строки `std::string` по сравнению со строкой в стиле `C`. Конструкторы класса `std::string`. Работа с отдельными символами в строке. Библиотека `cctype`. Строение объектов класса `std::string`, его размер. Методы класса строки: `push_back`, `pop_back`, `insert`, `clear`, `erase`, `substr`, `find`, `starts_with`, `c_str`, `size`, `capacity`, `reserve`, `resize` и `shrink_to_fit`. Передача строк в функции. Конвертация чисел в строки и наоборот. Чтение в строку. Функция `std::getline`. Литералы типа `std::string` из пространства имён `std::string_literals`. Оптимизация малой строки (SSO).

   e. **Класс `std::vector`**
   Контейнер `std::vector`. Внутренее устройство вектора. Конструкторы вектора. Доступ к отдельным элементам в векторе. Методы вектора: `push_back`, `pop_back`, `begin`, `end`, `front`, `back`, `clear`, `empty`, `data`, `size`, `capacity`, `reserve`, `resize`, `shrink_to_fit`. Передача вектора в функции.

   f. **Приведение типов в C++**
   Опасность приведения в стиле C. Оператор приведения `static_cast`. Отличие приведения типов с помощью оператора `static_cast` от приведения типов в стиле C. То есть чем

   ```
   static_cast<type>(a)
   ```

   отличается от:

   ```
   (type)(a)
   ```

   Оператор приведения `reinterpret_cast`. Оператор приведения `const_cast`.

2. **Классы**

   a. **Классы**
   Объектно-ориентированное программирование. Класс. Поля и методы класса. Члены класса. Инкапсуляция. Константные методы класса. Указатель `this`. Сокрытие данных. Модификаторы доступа `private` и `public`. Различие ключевых слов `struct` и `class` при объявлении классов. Геттеры и сеттеры. Друзья классов. Вложенные классы. Использование `typedef` внутри класса.

   b. **Конструкторы и деструкторы**
   Когда вызываются конструкторы, а когда деструкторы? Можно ли перегружать конструкторы и деструкторы? Различные синтаксисы вызова конструктора (с использованием знака `=`, с использованием круглых скобок и с использованием фигурных скобок). Конструкторы и передача в функции/возврат из функций. Делегирующий конструктор. Идиома RAII.

   c. **Перегрузка операторов**
   Перегрузка операторов в языке C++. Перегрузка арифметических операторов. Перегрузка унарных операторов. Перегрузка операторов как свободных функций и как методов класса. Перегрузка оператора присваивания. Перегрузка оператора присваивания сложения. Перегрузка оператора индексирования (квадратные скобки). Перегрузка операторов инкремента и декремента. Перегрузка оператора стрелочка (`->`). Перегрузка оператора вызова функции (круглые скобки). Перегрузка операторов `<<` и `>>` с объектами типа `std::ostream` и `std::istream`.

   d. **Реализация своего класса строки**
   Уметь писать свой простейший класс строки. Методы такой строки:

   - Конструктор по умолчанию

- Конструктор, принимающий строку в стиле C (`const char*`)
- Конструктор копирования
- Деструктор
- Оператор присваивания
- Оператор присваивания сложения(`+=`).
- Оператор сложения. Реализация оператора сложения с помощью оператора присваивания сложения (`+=`).
- Операторы сравнения.
- Оператор индексирования.

e. **Раздельная компиляция**
Вынос определений функций из класса. Forward declaration. Вынос определений функций в другие `.cpp` файлы.

3. **Инициализация**

a. **Виды инициализации**
Что такое инициализация? Классификация типов на скалярные типы, агрегатные типы и нормальные классы. Виды инициализации: *default initialization*, *value initialization*, *direct initialization*, *copy initialization*. `explicit`-конструкторы.

b. **Инициализация полей класса**
Когда инициализируются поля класса? Инициализация полей класса по умолчанию. Список инициализации полей класса.

c. **Особые методы класса**
Конструктор по умолчанию. Конструктор копирования. Оператор присваивания копирования. Деструктор. Конструктор перемещения. Оператор присваивания перемещения. При каких условиях компилятор автоматически создаёт эти методы. Что делают особые методы, автоматически созданные компилятором? Правило пяти. Удалённые функции и методы, ключевое слово `delete`. Ключевое слово `default` для особых методов класса.

d. **Динамическое создание объектов в куче**
Создание/удаление объектов в куче с помощью операторов `new` и `delete`. Создание/удаление массива объектов в куче с помощью операторов `new[]` и `delete[]`. Основные отличия `new` и `delete` от `malloc` и `free`. Оператор `placement new`. Как оператор `new` возвращает ошибку при нехватки памяти?

e. **Статические поля и методы**
Статическое поле. Инициализация статического поля. Статические методы.

f. **Основы работы с исключениями**
Зачем нужны исключения, в чём их преимущество перед другими методами обработки ошибок? Оператор `throw`. Что происходит после достижения программой 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`.

4. **Шаблоны**

a. **Шаблоны функций**
Шаблоны функций. Автоматический вывод типа для шаблона функции. Ограничения, накладываемые на тип шаблона функции. Шаблоны функций и перегрузка. Шаблоны с нетиповыми параметрами. Два этапа компиляции шаблона. Инстанцирование шаблона. Зависимые имена в шаблонах.

b. **Шаблоны классов**
Шаблоны классов. Вывод шаблонных аргументов классов.

c. **Стандартные шаблонные классы**
Класс вектора `std::vector<T>`. Класс массива `std::array<T, Size>`. Чем `std::array` отличается от `std::vector`? Чем `std::array` отличается от массива языка C? Класс пары `std::pair` и как его применять. Поля `first` и `second`. Класс `std::optional<T>`. Методы класса `std::optional`: `value`, `has_value` `value_or`. Объект `std::nullopt`. Конструкторы класса `optional`: конструктор по умолчанию, конструктор от объекта типа `T`, конструктор от другого `std::optional`.

d. `auto`
Ключевое слово `auto`. Range-based циклы. Structured bindings. Использование `auto` для типа возвращаемого значения функции. Использование `auto` для параметров функций.

e. **Вариативные шаблоны**
Использование вариативных шаблонов.

5. **Контейнеры**

a. **Итераторы**
Идея итераторов. В чём преимущество итераторов по сравнению с обычным обходом структур данных? Операции, которые можно производить с итератором вектора. Обход стандартных контейнеров с помощью итераторов. Передача итераторов в функции. Константные и обратные итераторы. Методы `begin`, `end`, `cbegin`, `cend`, `rbegin` и `rend`.

b. **Класс списка `std::list`**
С помощью какой структуры данных реализован список `std::list`? Как устроен список, где и как хранятся данные в списке? Итератор списка `std::list<T>::iterator`. Операции, которые можно производить с итератором списка. Методы списка: `insert`, `erase`, `push_back`, `push_front`, `pop_back`, `pop_front`. Вычислительная сложность этих операций. Как удалить элементы списка во время прохода по нему? Контейнер `std::forward_list`.

c. **Класс двухсторонней очереди `std::deque`**
Как устроена двухсторонняя очередь, где и как хранятся данные в ней? Операций, которые можно с ней провести и их вычислительная сложность.

d. **Контейнеры-множества**
Контейнер `std::set` – множество. Его основные свойства. С помощью какой структуры данных он реализован? Методы `insert`, `erase`, `find`, `count`, `lower_bound`, `upper_bound` и их вычислительная сложность. Как изменить элемент множества? Контейнер `std::unordered_set` – неупорядоченное множество. Его основные свойства. С помощью какой структуры данных он реализован? Основные методы этого контейнера и их вычислительная сложность. Контейнеры `multuset` и `unordered_multuset`.

e. **Контейнеры-словари**
Контейнер `std::map` – словарь. Его основные свойства. Методы `insert`, `operator[]`, `erase`, `find`, `count`, `lower_bound`, `upper_bound` и их вычислительная сложность. Контейнер `std::unordered_map`. С помощью какой структуры данных этот контейнер реализован? Его основные свойства и методы и их вычислительная сложность. Как изменить ключ элемента словаря? Контейнеры `multumap` и `unordered_multumap`. Как удалить из `multimap` все элементы с данным ключом? Как удалить из `multimap` только один элемент с данным ключом?

f. *** Настройка множеств и словарей**
Пользовательский компаратор для упорядоченных ассоциативных контейнеров. Пользовательский компаратор и пользовательская хеш-функция для неупорядоченнных ассоциативных контейнеров.

g. *** Инвалидация итераторов**
Инвалидация итераторов вектора. Инвалидация итераторов списка. Инвалидация итераторов множества и словаря.

6. **Алгоритмы**

a. **Основные алгоритмы**
Библиотека `algorithm`. Стандартные шаблонные функции из этой библиотеки: `max_element`, `sort`, `reverse`, `count`, `find`, `all_of`, `any_of`, `none_of`, `fill`, `unique`, `remove`. Библиотека `numeric`. Стандартные функции из этой библиотеки: `iota` и `accumulate`. Как написать подобные алгоритмы самостоятельно?

b. **Output и Input итераторы**
Output итераторы. Итератор `std::back_insert_iterator`. Как перегружены операторы для этого итератора? Использование функции `std::copy` и этого итератора для вставки элементов в контейнер. Итератор `std::ostream_iterator`, как перегружены операторы для этого итератора. Input итераторы. Итератор `std::istream_iterator`, как перегружены операторы для этого итератора.

c. **Категории итераторов**
Различие между итератором вектора и итератором списка. Какие операции можно применять к итератору вектора, но нельзя применять к итератору списка? Категории итераторов (Input, Output, Forward, Bidirectional, Random access). Допустимые операции для каждой категории итераторов. Привести пример итератора из каждой категории. Почему нельзя сортировать контейнер типа `std::list` с помощью стандартной функции `std::sort`? Функции `std::advance`, `std::next` и `std::distance`.

d. **Функциональные объекты**
Тип функция. Тип указатель на функцию. Функтор. Различие между функцией и функтором. Стандартные функторы: `std::less`, `std::greater`, `std::equal_to`, `std::plus`, `std::minus`, `std::multiplies`. Лямбда-функции. Передача функциональных объектов в функции.

e. **Алгоритмы, принимающие функциональные объекты**
Стандартные функции, принимающие функциональные объекты: `for_each`, `sort`, `stable_sort`, `find_if`, `count_if`, `all_of`, `generate`, `copy_if`, `transform`, `partition`, `stable_partition`. Как написать подобные алгоритмы самостоятельно?

f. **Лямбда-захват**
Захват локальных переменных по ссылке и по значению. Опасность захвата по ссылке.

7. **Move семантика**

a. **Перемещение**
Операция копирования. Что происходит при копировании (разберите случаи копирования скаляра, агрегата и нормального класса)? Операция перемешения. Что происходит при перемещении (разбирите случаи перемещения скаляра, агрегата и нормального класса)? Стандартная функция `std::move`. Что происходит при перемещении объектов классов `std::string` и `std::vector`? В чём преимущества перемещения над копированием? Когда происходит перемещение? Перемещение объекта в функцию, если функция принимает объект по значению.

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. Зачем нужно разделение выражений на lvalue и rvalue? Передача lvalue и rvalue выражений в функции, принимающие по значению.

c. **lvalue-ссылки и rvalue-ссылки**
Что такое lvalue-ссылки, а что такое rvalue-ссылки, в чём разница? 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.

8. **Умные указатели**

a. **Ошибки при работе с динамической памятью**
Утечки памяти. Утечки памяти при бросании исключений. Двойное удаление.

b. **Умный указатель `std::unique_ptr`**
Применение класса `std::unique_ptr` Основные свойства `std::unique_ptr`. Методы `operator*`, `operator->`, `get`, `release`. Ошибочное использование `std::unique_ptr` при его инициализации с помощью обычного указателя. Шаблонная функция `std::make_unique`. Перемещение объектов типа `unique_ptr`. Передача таких указателей в функции.

c. *** Реализация класса `std::unique_ptr`**
Нужно уметь писать класс, аналогичный классу `std::unique_ptr`. Циклические ссылки и `std::unique_ptr`.

d. *** 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`.