Модуль 1. Вопросы.

1. **Перегрузка**

   a. **Пространства имён**
   Пространство имён: что такое и зачем нужно. `using`-объявление. Анонимное пространство имён.

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

   c. **Перегрузка функций**
   Сигнатуры функций в языках C и C++. Перегрузка функций. Манглирование имён. Ключевое слово `extern "C"`. Правила разрешения перегрузки функций.

   d. **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).

   e. **Стандартный ввод/вывод**
   Вывод на экран с помощью объекта `std::cout`. Считывание с экрана с помощью объекта `std::cin`. Перегрузка операторов ввода вывода `<<` и `>>`.

   f. **Явное приведение типов в C++**
   Оператор `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 `reinterpret_cast`. Оператор `const_cast`.

2. **Классы**

   a. **Инкапсуляция и сокрытие**
   Классы. Поля и методы класса. Константные методы класса. Указатель `this`. Модификаторы доступа `private` и `public`. Различие ключевых слов `struct` и `class` в языке C++.

   b. **Конструкторы и деструкторы**
   Когда вызываются конструкторы, а когда деструкторы? Можно ли перегружать конструкторы и деструкторы? Список инициализации членов класса. Какие поля можно инициализировать с помощью списка инициализации, но нельзя инициализировать обычным образом? Делегирующий конструктор. Идиома RAII.

   c. **Особые методы класса**
   Конструктор по умолчанию. Конструктор копирования. Деструктор. Перегруженный оператор присваивания. Что делают особые методы, создающиеся по умолчанию?

   d. **Остальное**
   `explicit`-конструкторы. Удалённые функции и методы, ключевое слово `delete`. Ключевое слово `default` для особых методов класса. Друзья. Ключевое слово `friend`.

   e. **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.

3. **Инициализация**

   a. **Класс `std::string`**
   Стандартная строка `std::string`. Преимущества строки `std::string` по сравнению со строкой в стиле `C`. Строение объектов этого класса, его размер. Конструкторы этого класса. Оптимизация малой строки (SSO). Литералы типа `std::string` из пространства имён `std::string_literals`.

   b. **Класс `std::string_view`**
   Что такое `std::string_view` и в чём преимущество по сравнению с `std::string`? Методы `remove_prefix` и `remove_suffix`. Опасность возврата `string_view` из функции.

   c. **Виды инициализации**
   Default initialization. Value initialization. Direct initialization. Direct list initialization. Copy initialization. Copy list initialization. `explicit`-конструкторы.

   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. **Copy elision**
   Copy elision. Return value optimisation.

4. **Шаблоны**

   a. **Шаблонные функции**
   Как написать шаблонную функцию и как её использовать? Вывод шаблонных аргументов.

   b. **Шаблоны классов**
   Инстанцирование шаблона. Вывод шаблонных аргументов классов. Полная специализация шаблона. Частичная специализация шаблона.

   c. **Класс `std::pair`**
   Класс пары и как его применять. Поля `first` и `second`. Уметь написать свой класс пары.

   d. **Класс `std::vector`**
   Контейнер `std::vector`. С помощью какой структуры данных реализован? Как устроен вектор, где и как хранятся данные? Размер и вместимость вектора, методы `resize` и `reserve`. Методы `push_back`, `pop_back`, `insert`, `erase` и их вычислительная сложность. Когда происходит инвалидация итераторов вектора?

   e. **Класс `std::array`**
   Как устроен вектор, где и как хранятся данные? Когда происходит инвалидация итераторов массива?

5. **Контейнеры STL**

   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. Константные и обратные итераторы. Методы `begin`, `end`, `cbegin`, `cend` и другие.

   b. **Классы `std::list` и `std::forward_list`**
   С помощью какой структуры данных реализованы? Как устроен список, где и как хранятся данные в списке? Методы списка: `insert`, `erase`, `push_back`, `push_front`, `pop_back`, `pop_front`. Вычислительная сложность этих операций. Когда происходит инвалидация итераторов списка? Как удалить элементы списка во время прохода по нему?

   c. **Класс `std::deque`**
   Как реализован? Операций, которые можно с ним провести и их вычислительная сложность. Когда происходит инвалидация итераторов deque? Контейнеры адаптеры `std::stack`, `std::queue` и `std::priority_queue`.

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. Когда происходит инвалидация итераторов множества? 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? Когда происходит инвалидация итераторов словаря? Пользовательский компаратор для упорядоченных ассоциативных контейнеров. Контейнеры `multumap` и `unordered_multumap`. Как удалить из `multimap` все элементы с данным ключом? Как удалить из `multimap` только один элемент с данным ключом? Пользовательский компаратор и пользовательская хеш-функция для неупорядоченнных ассоциативных контейнеров.

f. **Ключевое слово auto и другое**
Инициализация. Ключевое слово `auto`. Range-based циклы. Пользовательские литералы. Structured bindings.

6. **Алгоритмы STL**

a. **Категории итераторов**
Различие между итератором вектора и итератором списка. Какие операции можно применять к итератору вектора, но нельзя применять к итератору списка? 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`. Категории итераторов (Random access, Bidirectional, Forward, Output, Input). 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`.

b. **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?

c. **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. Лямбда-захват. Захват по ссылке и по значению. Опасность захвата по ссылке.

d. **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?

7. **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`.