Теория:

1. **Небольшие различия между C и C++**
   Подключение библиотек языка C в языке C++. Библиотеки `cmath`, `cstdlib` и другие. Встроенный тип `bool`. Указатель `nullptr`. Разница между `nullptr` и `NULL`. Недостатки `NULL` из языка C. Присваивание указателя одного типа указателю другого типа в языках C и C++. Разница между функцией `abs` из библиотеки `stdlib.h` языка C и функцией `abs` из библиотеки `cmath` языка C++. Аналогичная разница для других математических функций. Функции с аргументами по умолчанию.

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

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

4. **Перегрузка операторов**
   Перегрузка операторов в языке C++. Перегрузка арифметических операторов. Перегрузка унарных операторов. Перегрузка операторов как методов класса. Перегрузка оператора присваивания. Перегрузка оператора присваивания сложения. Реализация оператора сложения с помощью оператора присваивания сложения (`+=`) и других подобных операторов. Перегрузка операторов ввода вывода `<<` и `>>` с `cin` и `cout`. Перегрузка оператора взятия индекса. Перегрузка операторов инкремента и декремента. Перегрузка оператора стрелочка (`->`). Перегрузка операторов `new` и `delete`. Перегрузка оператора вызова функции.

5. **Классы. Инкапсуляция**
   Что такое объектно-ориентированное программирование. Основные принципы ООП: инкапсуляция, композиция, наследование и полиморфизм. Классы. Поля и методы класса. Константные методы класса. Модификаторы доступа `private` и `public`. Указатель `this`. Различие ключевых слов `struct` и `class` в языке C++. Конструкторы и деструкторы. Список инициализации членов класса. Какие поля можно инициализировать с помощью списка инициализации, но нельзя инициализировать обычным образом. Перегрузка конструкторов. Конструктор по умолчанию. Конструктор копирования. Делегирующий конструктор. Ключевое слово `explicit`. Перегрузка оператора присваивания. Конструкторы и перегруженные операторы, создаваемые по умолчанию. Друзья. Ключевое слово `friend`.

6. **Динамическое создание объектов в Куче**
   Создание экземпляров класса в стеке и куче в языке C++. Использование операторов `new` и `delete`. Основные отличия `new` и `delete` от `malloc` и `free`. Операторы `new[]` и `delete[]`. Создание массива объектов в Куче с вызовом конструкторов по умолчанию у каждого объекта. Оператор `placement new` и ручной вызов деструктора.

7. **Реализация строки. Строки std::string**
   Реализация своей строки с выделением памяти в Куче. Методы такой строки: конструктор, принимающий строку в стиле C, конструктор копирования, конструктор по умолчанию, деструктор, оператор присваивания, оператор сложения, оператор присваивания сложения(`+=`). Стандартная строка `std::string`. Преимущества строки `std::string` по сравнению со строкой в стиле `C`.

8. **Шаблоны.**
   Шаблонные функции. Использование шаблонных функции в языке C++. Шаблоны классов. Инстанцированием шаблона. Вывод шаблонных аргументов функций и классов. Специализация шаблона. Частичная специализация шаблона.

9. **Реализация динамического массива**
   Реализация своего шаблонного класса динамического массива. Конструкторы, деструктор, оператор присваивания, итераторы. Использование `std::initializer_list` для реализации одного из конструкторов вектора.

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

11. **STL. Итераторы**
    Идея итераторов. В чём преимущество итераторов по сравнению с обычным обходом структур данных. Операции, которые можно производить с итератором. Обход стандартных контейнеров с помощью итераторов. Константные и обратные итераторы. Методы `begin`, `end`, `cbegin`, `cend` и другие. Итератор `std::back_insert_iterato` Использование функции `std::copy` для вставки элементов в контейнер. Итератор `std::ostream_iterator`. Функции `std::advance`, `std::next` и `std::distance`. Категории итераторов (Random access, Biderectional, Forward, Output, Input).

12. **STL. Контейнер `std::list`**
    С помощью какой структуры данных реализован. 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 удаляются элементы списка во время прохода по нему.

13. **STL. Другие последовательные контейнеры.**
    Контейнер `std::array`, как реализован, операций, которые можно с ним провести и их вычислительная сложность. Контейнер `std::deque`, как реализован, операций, которые можно с ним провести и их вычислительная сложность. Когда происходит инвалидация итераторов deque? Контейнер `std::valarray`, как реализован, операций, которые можно с ним провести и их вычислительная сложность. Контейнеры адаптеры `std::stack`, `std::queue` и `std::priority_queue`.

14. **STL. Упорядоченные ассоциативные контейнеры**
    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. Можно ли изменить элемент множества? Контейнер `std::map` – словарь. Его основные свойства. Методы `insert`, `operator[]`, `erase`, `find`, `count`, `lower_bound`, `upper_bound` и их вычислительная сложность. Как изменить ключ элемента словаря? Контейнеры `multuset` и `multimap`. Как удалить из `multimap` все элементы с данным ключом. Как удалить из `multimap` только один элемент с данным ключом? Когда происходит инвалидация итераторов множества и словаря? Пользовательский компаратор для упорядоченных ассоциативных контейнеров.

15. **STL. Неупорядоченные ассоциативные контейнеры**
    Контейнер `std::unordered_set` – неупорядоченное множество. Его основные свойства. С помощью какой структуры данных он реализован. Основные методы этого контейнера и их вычислительная сложность. Контейнер `std::unordered_map` – словарь. Его основные свойства и методы и их вычислительная сложность. Как изменить ключ элемента словаря? Контейнеры `std::unordered_multiset` и `std::unordered_multimap`. Как удалить из `unordered_multimap` только один элемент с данным ключом? Когда происходит инвалидация итераторов неупорядоченных множества и словаря. Пользовательский компаратор и пользовательская хеш-функция для неупорядоченнных ассоциативных контейнеров.

16. **Инициализация, ключевое слово auto и другое**
    Инициализация. Default initialization. Value initialization. Direct initialization. Direct list initialization. Copy initialization. Copy list initialization. Ключевое слово `auto`. Range-based циклы. Пользовательские литералы. `std::initializer_list`. Structure bindings. Copy elision. Return value optimization.

17. **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`.

18. **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.

19. **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. Скрипты `bash`. `Make`-файлы.

20. **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.

21. **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. Вызов переопределённого метода класса родителя. Object slicing. Множественное наследование. Ромбовидное наследование.

22. **Полиморфизм.**
    Полиморфизм в `C++`. Указатели на базовый класс, хранящие адрес объекта наследуемого класса. Виртуальные функции. Таблица виртуальных функций. Ключевые слова `override` и `final`. Виртуальный деструктор. Чистые виртуальные функции. Pure virtual call. Абстрактные классы и интерфейсы.

23. **Move-семантика.**
    Глубокое копирование и поверхностное копирование. Копирование объекта. Перемещение объекта. Стандартная функция `std::move`. В чём преимущества перемещения над копированием? Перемещение объекта при возврате из функции. Что такое выражение? lvalue-выражения и rvalue-выражения. Приведите примеры lvalue и rvalue выражений. Зачем нужно разделение выражений на lvalue и rvalue. rvalue-ссылки. В чём разница между lvalue-ссылками и rvalue-ссылками? Конструктор перемещения и оператор присваивания перемещения. Правило пяти.

24. **Умные указатели.**
    Недостатки обычных указателей. Умный указатель `std::unique_ptr`. Шаблонная функция `std::make_unique`. Основы move-семантики. Функция `std::move`. Перемещение объектов типа `unique_ptr`. Умный указатель `std::shared_ptr`. Работа с таким указателем. Шаблонная функция `std::make_shared`. Базовая реализация `std::shared_ptr`. Умный указатель `std::weak_ptr`.

25. **Приведение типов**
    В чём недостатки приведения в стиле `C`? Оператор `static_cast` и в каких случая он используется. Оператор `reinterpret_cast` и в каких случая он используется. Оператор `const_cast` и в каких случая он используется. Перегрузка оператора приведения. Использование `static_cast` для приведения типов и указателей на типы в иерархии наследования. Оператор `dynamic_cast` и в каких случая он используется. Что происходит если `dynamic_cast` не может привести тип, рассмотрите случай приведения указателей и случай приведения ссылок.

26. **Классы std::span и std::string_view**
    Класс `std::span`. Строение объектов этого класса, его размер. Конструкторы этого класса. Методы `first`, `last`, `subspan`. В чём преимущество передачи вектора в функцию, принимающую `std::span` по сравнению с функцией, принимающей `std::vector<T>&`. Класс `std::string_view`. Строение объектов этого класса, его размер. Конструкторы этого класса. Методы `remove_prefix` и `remove_suffix`. В чём преимущество передачи `string_view` в функцию. Опасность возврата `span` и `string_view` из функции.