Теория:

1. **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.

2. **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.

3. **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.

4. **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`.

5. **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. Вывод типа выражения с помощью `decltype`. Различие вывода с помощью `decltype`, `auto` и вывода шаблонных аргументов. Разложение типов (type decay) и когда он происходит.

6. **Идеальная передача.**
   Правила свёртки ссылок. Унивесальные ссылки. Что делает стандартная функция `std::forward`? Что такое идеальная передача (perfect forwarding) и зачем она нужна? Какие стандартные функции используют идеальную передачу?

7. **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 нужно использовать для внутренних ошибок, а какие для внешних.

8. **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`.

9. **Вычисления на этапе компиляции.** `constexpr`
   Метапрограммирование с помощью макросов. Метапрограммирование с помощью шаблонов. Вычисление на этапе компиляции с помощью `constexpr`. Что означает `constexpr` при объявлении переменной. Что означает `constexpr` при определении функции. Разница между `const` и `constexpr`. `consteval` и `constinit`.

10. **Типы отношений между объектами.**
    Композиция, ассоциация и наследование. Реализация этих типов отношений на языке `C++`. UML-диаграмы для представления иерархии классов.

11. **Паттерны Мост и Стратегия.**
    Паттерн Мост. Разница между паттернами Мост и Стратегия. Привести примеры использования данных паттернов.

12. **Паттерны Состояние и Конечный автомат.**
    Паттерн Состояние. В чём разница между паттернами Состояние и Стратегия. Паттерн Конечный автомат. В чём преимущество при использовании паттерна Конечный автомат для описания изменения состояний объекта по сравнению с использованием перечислений. Привести примеры использования данных паттернов.

13. **Паттерны Фабричный метод и Абстрактная фабрик.**
    Паттерн Фабричный метод. Когда используется этот паттерн и зачем он нужен? Фабричный метод как виртуальный конструктор. Паттерн Абстрактная фабрика. Привести примеры использования данных паттернов.

14. **CMake: основы синтаксиса**
    Что тако CMake? Зачем нужен CMake и его преимущество перед обычными системами сборки? Файл `CMakeLists.txt`. Язык CMake. Функция `message`. Типы CMake как строки. Конкатенация строк. Переменные CMake и функция `set`. Обычные и кэшированные переменные. Файл `CMakeCache.txt`. Условный переход `if` в CMake. `foreach` в CMake. Функии и макросы в CMake.

15. **CMake: сборка проектов**
    Использование CMake для сборки проектов. Функции `cmake_minimum_required`, `project`. Таргеты в CMake. Функции `add_executable` и `add_library`. Использование функции, основанных на работе с таргетами, такие как `target_include_directories` и `target_compile_options`. Функция `target_link_libraries` и что она делает. Почему не стоит использовать функции `include_directories` и `link_libraries`? Функция `add_subdirectory`.