Теория:

1. **Параллелизм и конкурентность.**
   Что такое параллелизм и что такое конкурентность? Что такое процесс и что такое поток? Организация параллелизма с использование процессов и с использованием потоков. В чём преемущества и недостатки этих подходов. Зачем нужен параллелизм?

2. **Потоки. Класс `std::thread`**
   Что такое поток? Создание нового потока в языке `C++` с использованием объекта класса `std::thread`. Методы `join` и `detach`. Что произойдёт если выбросится исключение (в новом потоке, или в потоке, который создаёт новый поток)? Передача аргументов в функцию потока. Возврат данных из нового потока. Передача владения потоком. Создание произвольного количества потоков. Идентификация потоков.

3. **Состояние гонки и мьютексы.**
   Что такое разделяемые данные? Что такое состояние гонки (race condition)? Проблематичные и безобидные состояния гонки. Что такое гонка данных (data race) и к чему она приводит? Защита разделяемых данных с помощью мьютекса. Класс `std::mutex`. Методы `lock`, `unlock` и `try_lock`. В чём недостатки класса `std::mutex`? Класс `std::lock_guard`. В чём преимущество `std::lock_guard` перед `std::mutex`? Класс `std::unique_lock`. В чём преимущества и недостатки `std::unique_lock` перед `std::lock_guard`? Взаимоблокировка (deadlock). Решение проблемы взаимоблокировки с помощью стандартной функции `std::lock`.

4. **Механизмы синхронизации.**
   Условные переменные. Класс `std::condition_variable` и как им пользоваться? Методы `wait`, `notify_one` и `notify_all`. Ложные пробуждения (spurious wake). Запуск асинхронной задачи с помощью функции `std::async`. Возврат значения из фоновой задачи с помощью объекта класса `std::future`. Класс задачи – `std::packaged_task`. Зачем могут понадобиться объекты класса `std::packaged_task`? Методы класса `std::packaged_task`: `get_future`, `operator()`. Передача объекта класса `std::packaged_task` в другие функции и потоки. Класс `std::promise`. Методы класса `std::promise`: `get_future`, `set_value` и `set_exception`.

5. **Потокобезопасные стек и очередь с блокировками.**
   Что такое потокобезопасная структура данных? Являются ли стандартные контейнеры `STL` потокобезопасными? Стандартный класс `std::stack` и его методы `push`, `top` и `pop`. Почему в стандартной библиотеке языка `C++` стек реализован так, как он реализован? Что такое потокобезопасная структура данных с блокировками? Написание своего потокобезопасного стека с блокировками. Реализация методов `push` и `pop` такого стека. Потокобезопасная очередь с блокировками. Реализация методов `push`, `try_pop` и `wait_and_pop` такой очереди.

6. **Атомарные типы и операции над ними.**
   Атомарные переменные. В чём отличие атомарных переменных от обычных переменных? Класс `atomic_flag` и его методы `clear` и `test_and_set`. Атомарные типы `atomic<T>` и методы `load`, `store` и `compare_exchange`. Упорядочение доступа к памяти. Упорядочения `memory_order_seq_cst`, `memory_order_acquire`, `memory_order_release` и `memory_order_relaxed`. Реализация спинлока (простейшего мьютекса) на основе атомарной переменной.

7. **Потокобезопасные стек и очередь без блокировок.**
   Неблокирующие структуры данных. Структуры данных, свободные от блокировок. Структуры данных, свободные от ожидания. Реализация потокобезопасного стека без блокировок (без устранения утечек памяти). Реализация потокобезопасной очереди без блокировок (без устранения утечек памяти).

8. **Управление памятью в структурах данных без блокировок.**
   Реализация потокобезопасного стека без блокировок (без утечек памяти). Метод подсчёта количества потоков, выполняющих `pop`. Метод указателей опасности (hazard pointers). В чём преемущества и недостатки каждого из методов. Реализация потокобезопасной очереди без блокировок (без утечек памяти).

9. **Пул потоков.**
   Что такое пул потоков? Реализация пула потоков на языке C++. Ожидание задачи, переданной пулу потоков. Предотвращение конкуренции за очередь работ. Занимание работ.

10. **Основы Cmake.**
    Что такое Cmake и для чего он нужен? Основы работы с CMake. Структура CMake-проекта. Файл CMakeListis.txt. Генерация файлов проекта для данной среды. Как скомпилировать проект с помощью CMake? Что такое таргет (target)?
    Что делают следующие команды CMake:

    - `cmake_minimum_required`
    - `project`
    - `set`
    - `option`
    - `message`
    - `add_executable`
    - `add_library` и её опции `STATIC`, `SHARED` и `MODULE`
    - `target_link_libraries` и её опции `PUBLIC`, `PRIVATE` и `INTERFACE`
    - `target_compile_options`
    - `add_subdirectory`
    - `find_package`

11. **Переменные Cmake.**
    Переменные Cmake. Как создавать переменные и как их использовать в CMake? Какие типы переменных есть в CMake? Переменные среды. Кешированные переменные. Чем кешированные переменные отличаются от обычных переменных? Поле `type` при создании кешированной переменной и какие значения оно может принимать. Изменение кешированных переменных. Задание кешированных переменных внутри CMake-скрипта, в командной строке и путём изменения файла CmakeCache.txt.
    Переменные:

    - `CMAKE_SOURCE_DIR`
    - `CMAKE_BUILD_DIR`
    - `CMAKE_CURRENT_SOURCE_DIR`
    - `CMAKE_CURRENT_BUILD_DIR`

12. **Boost Asio.**