1. Разраб HTML-страницу с бинарным wasm-кодом, полученным в рез. компиляции трех C-функций: ***sum, mul, sub***. Бинарный код получить с пом WasmFiddle. Продем выполнение с пом браузера в автономном режиме (файл D:/1.html)

\* исп. **WASMFiddle** – инструмент, позв. посмотреть как раб. WebAssembly  
\* получили бин. код и вставили его в приложение (26-01.html)  
 (массив wasmCode) – это откомпилированные ф-и, byte-код   
\* создали модуль, инстанс  
\* с пом export получали инстанс, кот. вып. ф-ции

демонстрируем **26-01.html**

2. Устан компилятор emcc. Получит wasm-бинарный код функций sum, mul, sub. Разработать серверную HTML-страницу, кот. с пом JS считывает WASM-код с пом GET-запроса с сервера и исполняет его. Продем в браузере

**Установка emcc:** \* **git clone https://github.com/emscripten-core/emsdk.git**  
 (получ. копию git-репозитория)  
 => появ. папка emsdk – заходим в нее  
 \* git pull – фетчим посл. версию, т.е. извлек  
 \* emsdk install latest – скач. и активируем посл. SDK  
 \* emsdk activate – активируем для исп-я  
 \* emsdk\_env.sh – активируем PATH и ост. пер. среды

**p.c** - написали на С ф-ции:  
 \* emsripten\_keepalive – префикс, сод. правила передачи парам-ров => позв.   
 экспорт. ф-ции и использ. их извне  
 \* extern “c” – указ, что имена ф. дб написаны согласны правилам C

**Получить wasm-код:** **$emcc -03 -o p.wasm WASM=1 p.c**  
 - 03 – опция оптим-ции – указ. до какого ур-ня хотим преобр (лучшая оптим)  
 - о p.wasm – куда компилировать  
 - s wasm=1 – указ что хотим получить wasm код  
 p.c – откуда компилировать  
 => появился **p.wasm**

Разраб **HTML**, кот. с пом JS считывает WASM-код с пом GET с сервера и исполняет его  
**26-01.js** – по GET-запросу выдает html localhost:3000/26-02.html  
**26-02.html** – с пом. fetch будем считыв. wasm-код и работать с ф-ями

3. Разработать Node.js приложение 26-01, выполн функции sum, mul, sub из WASM-бинарного кода, полученного в пред. задании.   
**26-01.js**: в wasmCode запис. из файла, по get вызов ф-й **localhost:3000/**

**ТЕОРИЯ**

**Wasm (Web Assembly)** - бинарный формат исполняемого файла, кот м.исполнятся в js engine (для стековой виртуальной машины)  
 + код быстрее чем JS  
 + поддерж больш-вом браузеров  
 + выполн. в sandbox  
 + есть отладчики  
 + открытый стандарт

**Идея:** разраб. прогу – компилируем => получ бинарный код исполняемого файла, этот код переносим на сторону клиента и он исполняется на стороне Клиента (в JS Engine)

**Для чего?**\* мы отсылаем предкомпилированный код => та часть, кот. выполнялась бы Engine-ом уже выполнена => можно экономить  
\* компилируем не на лету, а в оффлайне (JS налету) => мы не связаны временем исполнения (отделим стадию компиляции и стадию исполнения => можем не экономить на времени компиляции и можем оптимизировать

Этот код можно получить 2 способами:  
1) Wasm Fiddle (внизу) – сильная оптимизац  
2) компилятор emcc – компил, кот комипилируем код из c++ в wasm

**Wasm Fiddle** – инструмент, позв. проверить как работает Web Assembly  
 + удобный  
 + помогает разработать 1ый код, посмотреть как устроен JS API  
 wasdk.github.io/WasmFiddle

**EMCC** – компилятор, кот комипилирует код из c++ в wasm