Цели практической работы:

1. Создание и работа с WAT-модулями.
2. Интеграция WAT с JavaScript.
3. Применение условных операторов и операторов цикла в WAT.
4. Компиляция и выполнение WAT-кода.

Часть 1: Создание WAT-модуля

Задание 1: Напишите WAT-модуль, который реализует функцию для вычисления факториала числа.

Инструкции:

1. Создайте новый файл с расширением `.wat`, например, `factorial.wat`.
2. Определите функцию `factorial`, которая принимает один параметр `n` (целое число).
3. Реализуйте вычисление факториала с использованием циклов.
4. Экспортируйте функцию `factorial`.

Варианты:

1. **Простой вариант:** Вычисление факториала числа до 5.
2. **С проверкой на ноль:** Если `n` равно 0, возвращать 1.
3. **Использование рекурсии:** Реализуйте факториал с использованием рекурсии.
4. **Обработка больших чисел:** Убедитесь, что функция работает для больших чисел (например, до 12).

Задание 2: Реализуйте WAT-модуль для проверки, является ли переданное число четным.

Инструкции:

1. Создайте новый файл `.wat`, например, `is_even.wat`.
2. Определите функцию `is_even`, которая принимает один параметр `x` (целое число).
3. Реализуйте проверку, является ли `x` четным, и возвращайте 1, если да, и 0, если нет.
4. Экспортируйте функцию `is_even`.

Варианты:

1. **Базовая проверка:** Прямое деление и проверка на остаток.
2. **Использование битовых операций:** Проверка с помощью битовых операций.
3. **Работа с отрицательными числами:** Убедитесь, что функция корректно обрабатывает отрицательные числа.
4. **Проверка на ноль:** Обработка случая, когда `x` равно 0.

Часть 2: Создание файла JavaScript для взаимодействия с WAT

Задание 3: Напишите JavaScript-код, который загружает WAT-модуль и вызывает функцию `factorial`.

Инструкции:

1. Создайте новый файл `.js`, например, `main.js`.
2. Используйте WebAssembly API для загрузки и компиляции WAT-модуля.
3. Вызовите функцию `factorial` с различными аргументами и выведите результаты в консоль.

Варианты:

1. **Простой вызов:** Вызов функции `factorial` с числом 5.
2. **Массив значений:** Вызов функции `factorial` с массивом чисел и вывод результатов.
3. **Обработка ошибок:** Добавьте обработку ошибок для случаев, когда модуль не загружается или функция вызывает исключение.
4. **Взаимодействие с пользователем:** Получение числа от пользователя и вызов функции `factorial`.

Задание 4: Напишите JavaScript-код для использования функции `is_even` из WAT-модуля.

Инструкции:

1. Создайте новый файл `.js`, например, `check_even.js`.
2. Используйте WebAssembly API для загрузки WAT-модуля.
3. Вызовите функцию `is_even` и выведите результат в консоль.

Варианты:

1. **Простой вызов:** Вызов функции `is_even` с числом 10.
2. **Случайные числа:** Вызов функции `is_even` с случайными числами.
3. **Обработка ошибок:** Обработка возможных ошибок при загрузке или вызове функции.
4. **Взаимодействие с пользователем:** Запрос числа у пользователя и проверка четности этого числа.

Часть 3: Использование условных операторов и операторов цикла

Задание 5: Напишите WAT-модуль, который находит наибольший элемент в массиве чисел.

Инструкции:

1. Создайте новый файл `.wat`, например, `find_max.wat`.
2. Определите функцию `find_max`, которая принимает массив чисел и возвращает наибольший элемент.

3. Используйте цикл и условные операторы для нахождения максимального элемента.
4. Экспортируйте функцию `find_max`.

Варианты:

1. **Массив фиксированного размера:** Массив фиксированного размера, например, 5 элементов.
2. **Динамический массив:** Массив переменной длины, передаваемый как параметр.
3. **Обработка пустого массива:** Возвращайте специальное значение или ошибку для пустого массива.
4. **Отрицательные значения:** Обработка массива, содержащего отрицательные значения.

Задание 6: Реализуйте WAT-модуль, который вычисляет сумму чисел в массиве до тех пор, пока сумма не станет больше 100.

Инструкции:

1. Создайте новый файл `.wat`, например, `sum_until_limit.wat`.
2. Определите функцию `sum_until_limit`, которая принимает массив чисел и суммирует их до тех пор, пока сумма не станет больше 100.
3. Используйте циклы и условные операторы для вычисления суммы.
4. Экспортируйте функцию `sum_until_limit`.

Варианты:

1. **Фиксированный массив:** Массив фиксированного размера, например, 5 элементов.
2. **Массив переменной длины:** Массив, длина которого передается как параметр.
3. **Обработка отрицательных чисел:** Убедитесь, что функция корректно работает с отрицательными значениями.
4. **Вывод промежуточных сумм:** Вывод промежуточных сумм в процессе вычисления.

Часть 4: Компиляция и выполнение WAT-кода

Задание 7: Скомпилируйте WAT-модуль в WebAssembly и выполните его в браузере.

Инструкции:

1. Используйте `wat2wasm` для компиляции WAT-файла в бинарный формат WebAssembly.
2. Создайте HTML-файл, который загружает и выполняет скомпилированный модуль.
3. Убедитесь, что модуль правильно загружается и функции вызываются корректно.

Варианты:

1. **Компиляция и выполнение одного модуля:** Компиляция и выполнение простого WAT-модуля.
2. **Множественные модули:** Компиляция и использование нескольких WAT-модулей в одном проекте.
3. **Интерактивная веб-страница:** Создание веб-страницы с пользовательским интерфейсом для взаимодействия с WebAssembly.
4. **Тестирование:** Написание тестов для проверки корректности работы функций WebAssembly.

Задание 8: Настройте проект с использованием Emscripten для компиляции C/C++ кода в WebAssembly и его выполнения.

Инструкции:

1. Установите Emscripten и настройте его окружение.
2. Напишите простой C/C++ код и используйте Emscripten для его компиляции в WebAssembly.
3. Создайте HTML-файл для загрузки и выполнения скомпилированного модуля.

Варианты:

1. **Простой C код:** Компиляция и выполнение простого C-кода (например, функция `main`).
2. **Использование параметров:** Передача параметров из JavaScript в скомпилированный модуль.
3. **Обработка ошибок:** Обработка возможных ошибок при компиляции и выполнении.
4. **Интерактивный интерфейс:** Создание веб-страницы с элементами управления для взаимодействия с C/C++ кодом через WebAssembly.

Задание 9: Разработайте JavaScript-код для вызова функций из WebAssembly модуля и обработки результатов.

Инструкции:

1. Создайте JavaScript-код для загрузки и выполнения WebAssembly модуля.
2. Вызовите функции и обработайте их результаты.
3. Выведите результаты в консоль или на веб-страницу.

Варианты:

1. **Базовый вызов функции:** Вызов одной функции и вывод результата в консоль.
2. **Множественные вызовы:** Вызов нескольких функций и отображение результатов на веб-странице.
3. **Асинхронная загрузка:** Использование асинхронных функций для загрузки и выполнения WebAssembly.
4. **Обработка ошибок:** Обработка и отображение ошибок при загрузке и вызове функций.

Задание 10: Создайте и протестируйте проект на Node.js, который загружает и выполняет WebAssembly модуль.

Инструкции:

1. Установите Node.js и необходимые модули для работы с WebAssembly.
2. Напишите Node.js скрипт для загрузки и выполнения WebAssembly модуля.
3. Проверьте выполнение и обработку результатов.

Варианты:

1. **Базовый проект:** Простое выполнение одного WebAssembly модуля.
2. **Модуль с функциями:** Выполнение WebAssembly модуля с несколькими функциями и вывод результатов.
3. **Обработка ошибок:** Обработка ошибок при загрузке и выполнении WebAssembly в Node.js.
4. **Интерактивный консольный интерфейс:** Создание консольного интерфейса для взаимодействия с WebAssembly модулем.