Colon

Основы

Реализованный функционал:

1) **Арифметические операции:**
    - в модуле AritmeticOperations написана реализация операций +,-,*,/,mod

    - Реализован через монадический стек

    - Обобщая реализацию можно сказать что мы просто достаем из стека 2 значение и совершаем необходимую операцию(+,-,*,/,mod) и результат добавляем в стек
2) **Манипулирование над стеком:**
    - В модуле StackOperations написана реализация операций push,pop,dup,swap,rot,over
    - В реализации мы просто выполняем определенные действия над стеком
3) **Сравнения**
    - В модуле CompairingOperations написана реализация операций сравнения >,<,=
    - Так же как и в вышеописанных модулях, в реализации мы достаем из стека 2 элемента и результат выполненной операции кладем в стек (-1,0)

Для всех команд создали тип Command и в него добавили все необходимые команды. Они описаны в модуле CommandExecutor.
В нем же и реализована каждая команда.

4) **Ввод-Вывод**
    - Реализация операций . , emit, key, cr описаны в модулях CommandExecutor, а реализация команды CR написана в модуле StringParser(не успел еще перенести).

- В реализации CR мы проверяем есть ли CR в нашей строке, если да, то разбиваем строку на до CR и после удаляем CR и повторяем ту же логику для оставшейся строки(рекурсия). Реализация Key,Emit, . все так же, достаем значение из стека(либо из консоли) и выполняем соответствующую операцию.

5) **Строки ." " и парсинг комментариев**
   - Реализация описана в модуле StringParser.
   - Строки: Во входной строке проверяем начинается ли она с . ", если да то выводим содержимое внутри
   - Комментарии: проверяем начинается ли строка с #, если да то просто пропускаем ее.
6) **Условный оператор и цикл DO I LOOP.**
   - Реализация описана в модуле CommandExecutor.
   - IF ELSE: Проверка верхнего элемента стека и выполнение различных ветвей в зависимости от значения
   - DO LOOP: Выполнение цикла пока в стеке не появится 0.

Реализация и взаимодействие

Парсинг команд

- В модуле MainParser использовал Megaparsec для преобразования текста в команды типа Command. Для IF ELSE, DO LOOP написал свои парсеры.

Исполнение

- Как писал выше есть отдельный модуль исполнения команд CommandExecutor. Каждая команда реализована в executeCommand
- Команды взаимодействуют с другими командами. Например большинство из команд используют pop, push. Conditional и Do обрабатывают другие команды переданные как Program (список команд)

Обработка ошибок:

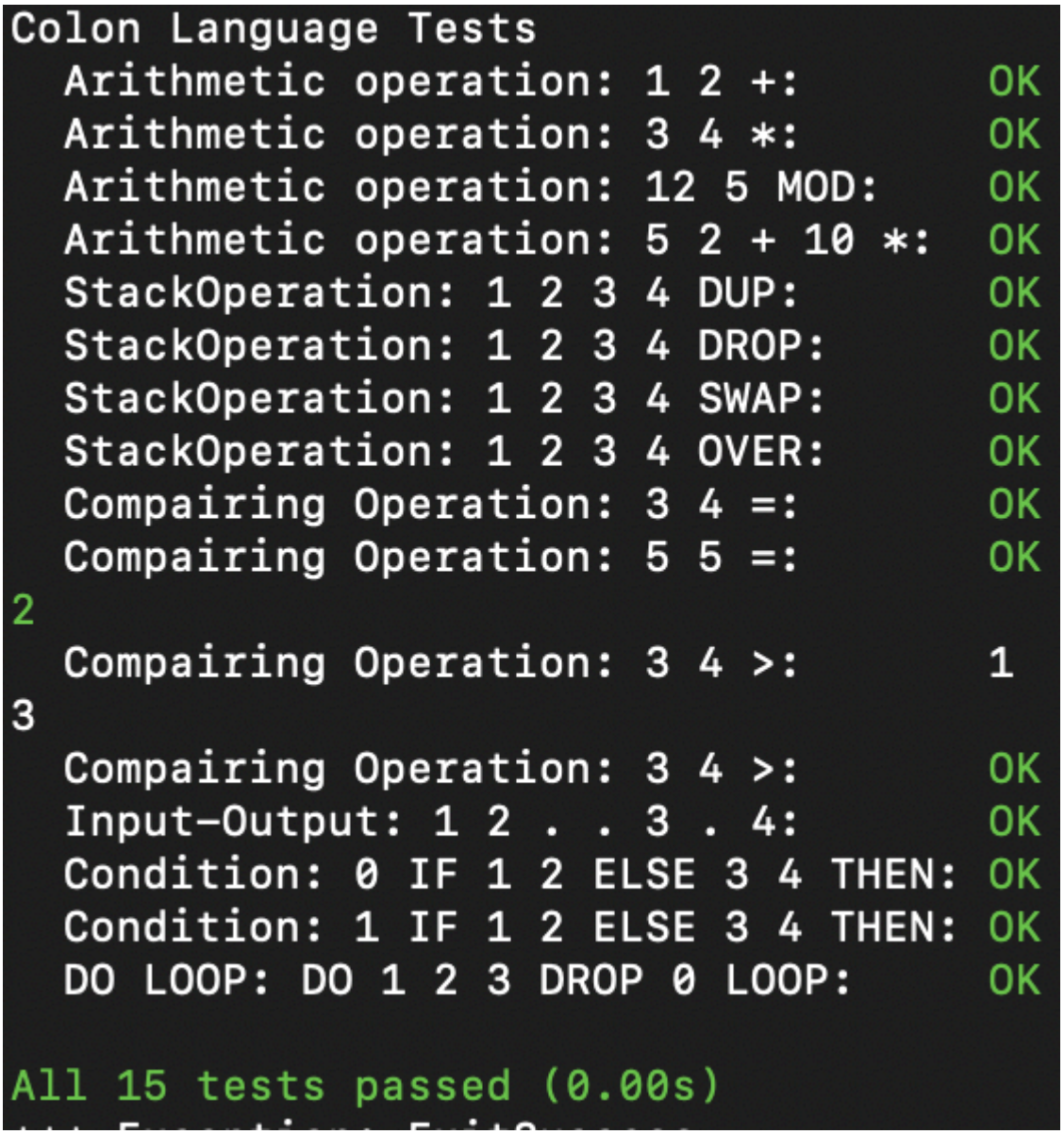
- Парсер возвращает детализированные ошибки при ошибках

Тестирование

Для тестов использовал библиотеки Tasty, HUnit, QuickCheck
Тестами покрыто:

1) Арифметические операции
2) Ввод вывод
3) Манипуляции над стеком
4) Операции сравнения
5) IF ELSE
6) DO LOOP

Результаты:



Личные выводы

Во-первых, узнал синтаксис haskell и вообще что такое функциональное программирование.

Во-вторых, научился работать с парсером Megaparsec.
В-третьих, научился писать тесты с использованием библиотек

Возникшие проблемы и сложности:

Было очень тяжело привыкнуть к синтаксису haskell и вообще к ФП. Так же были проблемы с использованием и согласованием типов между StateT и IO.
Ну и в самом начале приходилось хардкодить для парсинга, но в итоге все решилось использованием сторонних библиотек.

На следующий этап беру:

- конструкция `case of endof endcase` (3 если с парсингом)
- арифметические операции над числами с плавающей запятой (2)
- проверка комментариев про стек (3)
- массивы (2)
- пошаговая отладка, получение текущего стека, памяти, словаря и т.п. (3)
- цикл `BEGIN UNTIL` (1)
- конкурентное программирование: `RX`, `G!` (3)
- multi-exit loops (4 если с парсингом)