

## Test scripts for CSE201A

Please merge your homework with the respective test script repository. You should include a *Makefile* such that running `make` in the root directory of your submission produces a file in the root directory. The file, when executed, should read the strings via `stdin` and output the required content via `stdout`.

These test scripts are prepared based on Sohum Banerjee's work.

### Instructions

#### Assignment 1

**Assignment 1: ARITH** was implemented in Golang. No additional dependencies are required since all parsing is done using handwritten parser combinators.

The go installation instructions can be followed from <https://golang.org/doc/install>

To build the executable, simply run

```
make
```

The implementation of ARITH was largely inspired by Armin Heller's posts: \*  
<https://medium.com/@armin.heller/parser-combinator-gotchas-2792deac4531> \*  
<https://medium.com/@armin.heller/parser-combinator-gotchas-2792deac4531>

The key differences are that the original blog post evaluates the expressions inline instead of building a parse tree and in the assignment, some additional combinators are implemented.

#### Assignment 2

**While** was implemented in Haskell using the stack build tool. The stack tool is able to select the right version of GHC automatically (as specified in the `stack.yaml` file), so no additional work is needed to setup the compiler.

Stack can be set up following the instructions from here: <https://docs.haskellstack.org/en/stable/README/>

The implementation of while was inspired by "Write You A Haskell" tutorial by Stephen Diehl. The section on Parsers covered how to implement components of an arithmetic expression parser. Parts of the paper "Applicative program with effects" helped me understand applicative functors to be used in this code.

- [http://dev.stephendiehl.com/fun/002\\_parsers.html](http://dev.stephendiehl.com/fun/002_parsers.html)
- <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.114.1555&rep=rep1&type=pdf>

The Boolean expressions, statement expressions, ternary operator, variable assignment and dereferencing and the interpreter were fully implemented by me.

## Assignment 4

**Small Step While** official submission was implemented in Python using ANTLR4 as a build tool for extra credit. The only dependency required is the `antlr4-python3-runtime`. The requirements are provided in a `requirements.txt` file.

The implementation of the ANTLR grammar `While.g4`, the python code `while_small_step.py`, `WhileNode.py`, `WhileStructVisitor.py`, `WhileEvalVisitor.py` are all done by me. The rest code that was automatically generated by ANTLR.

There is also a Haskell implementation included in the main repo made initially built using stack. The stack tool is able to select the right version of GHC automatically (as specified in the `stack.yaml` file), so no additional work is needed to setup the compiler.

Stack can be set up following the instructions from here: <https://docs.haskellstack.org/en/stable/README/>

The definition of Show for pretty printing the AST was based off Prof. Flanagan's code in <https://canvas.ucsc.edu/courses/32489/assignments/122288>

The implementation of the evaluation, printing functions etc were fully by me.

## Test Case Coverage

```
(base) → cse210A-asgtest-hw1-arith git:(master) x ./test.sh
cd arith-go ; go build ; cp arith ../arith
✓ custom-1
✓ custom-2
✓ custom-3
✓ custom-4
✓ custom-5
✓ easy-1
✓ easy-2
✓ easy-3
✓ easy-4
✓ easy-5
✓ easy-6
✓ easy-7
✓ easy-8
✓ easy-9
✓ easy-10
✓ easy-11
✓ easy-12
✓ easy-13
✓ easy-14
✓ easy-15
✓ easy-16
✓ easy-17
✓ easy-18
✓ easy-19
✓ easy-20
✓ medium-1
✓ medium-2
✓ medium-3
✓ medium-4
✓ medium-5
✓ medium-6
✓ medium-7
✓ medium-8
✓ medium-9
✓ medium-10

35 tests, 0 failures
```

Figure 1: ARITH cases

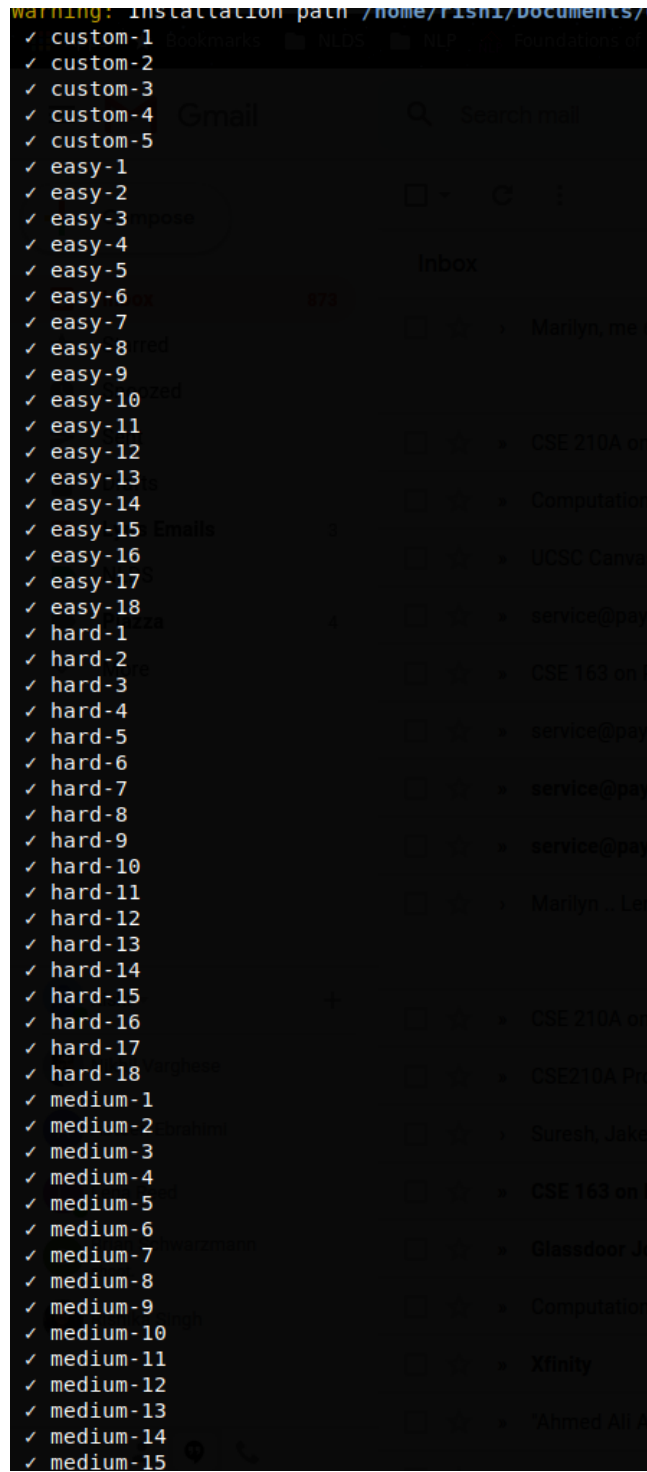


Figure 2: WHILE cases

```

(nlds) → cse210A-asgtest-hw4-whileless git:(master) x ./test.sh
echo 'python whileless-python/while_small_step.py' > while-ss ; chmod +x while-ss
✓ custom-1   E custom.txt 14
✓ custom-2   E custom.txt 14
✓ custom-3   E easy bats 15
✓ custom-4   E easy bats 15
✓ custom-5   E hard bats 16
✓ easy-1     E harness bash 14
✓ easy-2     E harness bash 14
✓ easy-3     E medium bats 15
✓ easy-4     E medium bats 15
✓ easy-5     x while-ss 16
✓ easy-6     x while-ss 16
✓ easy-7     x whileless-python 17
✓ easy-8     x whileless-python 17
✓ easy-9     x gitignore 18
✓ easy-10    E git-logs.txt 19
✓ easy-12    E git-logs.txt 19
✓ easy-13    E Makefile 20
✓ easy-14    E Makefile 20
✓ easy-15    E README.pdf 21
✓ easy-16    E README.pdf 21
✓ easy-17    E requirements.txt 22
✓ hard-1     E requirements.txt 22
✓ hard-2     E test.sh 23
✓ hard-3     E test.sh 23
✓ hard-4     E while-ss 24
✓ hard-5     E while-ss 24
✓ hard-6     x libexec 25
✓ hard-7     x libexec 25
✓ hard-8     x arith-tests.png 26
✓ hard-9     x arith-tests.png 26
✓ hard-10    E git-logs.txt 19
✓ hard-11    E README.md 27
✓ hard-12    E README.md 27
✓ hard-13    x while-ss-tests.png 28
✓ hard-14    x while-ss-tests.png 28
✓ hard-15    x while-tests.png 29
✓ hard-16    x while-tests.png 29
✓ hard-17    x whileless-python_results.png 30
✓ hard-18    x whileless-python_results.png 30
✓ hard-19    x whileless-python_results.png 30
✓ hard-20    x whileless-python_results.png 30
✓ medium-1   E 30
✓ medium-2   E 30
✓ medium-3   E 30
✓ medium-4   E 30
✓ medium-5   E 30
✓ medium-6   E 30
✓ medium-7   E 30
✓ medium-8   E 30
✓ medium-9   E OUTLINE 31
✓ medium-10  E OUTLINE 31
51 tests, 0 failures 30

```

Figure 3: WHILE-SS cases