

## 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
```

has a *Makefile*, such that running `make` in the directory called *arith*.

- This file, when executed as `./arith`, takes an input AST, and writes the result to `stdout`.
- There should be no filename conflicts.
- You may write your test cases by hand.

You may execute all tests in the `test.sh` script. This should make you and the TA feel confident about the only ones you are tested on. Since you are not tested on the `custom` cases, please keep track of your code using `git`.

If you commit frequently and conscientiously, you can be sure you have a good version of your code. Please be careful of your code (and try to resolve the conflicts with confidence).

Please submit a zip file that contains:

- All the code you've written for the assignment.
- A pdf report that contains screenshots of your code and test results.
- A txt file of your git commit logs.

Both your code and your report will be tested. Your code and your report will be exactly the same.

Good luck!

---

Rubric (2)

Figure 1: ARITH cases

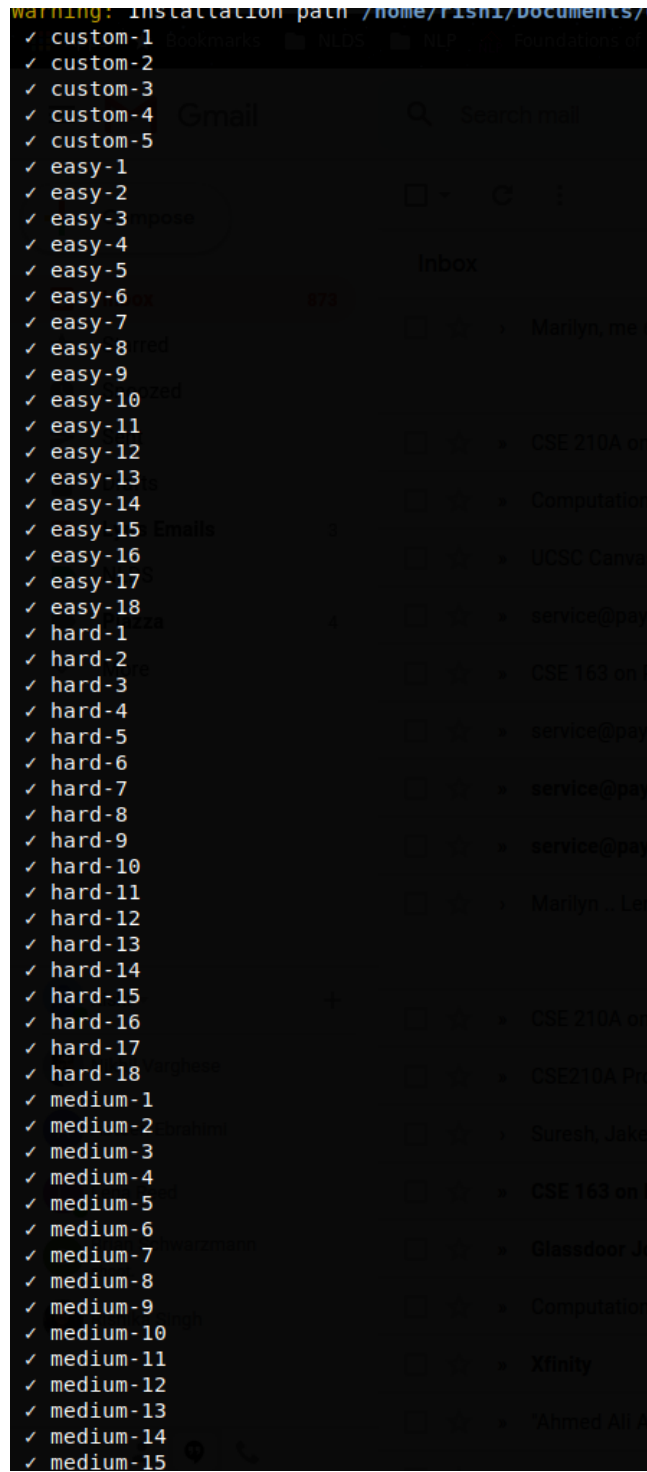


Figure 2: WHILE cases

```

skip, {x := 1}
(hw4) → cse210A-asgtest-hw4-whiless git:(master) x ./test.sh
echo 'python whiless-python/while_small_step.py' > while ; chmod +x while-ss
✓ easy-1
✓ easy-2
✓ easy-3
✓ easy-4
✓ easy-5
✓ easy-6
✓ easy-7
✓ easy-8
✓ easy-9
✓ easy-10
✓ easy-12
✓ easy-13
✓ easy-14
✓ easy-15
✓ easy-16
✓ easy-17
✓ hard-1
✓ hard-2
✓ hard-3
✓ hard-4
✓ hard-5
✓ hard-6
✓ hard-7
✓ hard-8
✓ hard-9
✓ hard-10
✓ hard-11
✓ hard-12
✓ hard-13
✓ hard-14
✓ hard-15
✓ hard-16
✓ hard-17
✓ hard-18
✓ hard-19
✓ hard-20
✓ medium-1
✓ medium-2
✓ medium-3
✓ medium-4
✓ medium-5
✓ medium-6
✓ medium-7
✓ medium-8
✓ medium-9
✓ medium-10
46 tests, 0 failures

```

Figure 3: WHILE-SS cases