Programming Assignment 0: "Stack"

For all "general" questions, please ask on our QM+ forum. These include questions about setting up and running COOL, and questions about the COOL language or this specification.

Do NOT share your actual assessment solution code on the forum, for obvious reasons. In the worst case, doing so will considered collusion in assessed work. You CAN use the forum if you take care with the phrasing of your question, possibly by abstracting your actual code in some way.

1 Overview

This assignment asks you to write a short Cool program. The purpose is to acquaint you with the Cool language and to give you experience with some of the tools used in this course. This assignment will *not* be done in a team; you should turn in your own individual work. All future programming assignments will be done in small teams.

A machine with only a single stack for storage is a *stack machine*. Consider the following very primitive language for programming a stack machine:

Command	*Meaning*
int	push the integer *int* on the stack
+	push a '+' on the stack
s	push an 's' on the stack
e	evaluate the top of the stack (see below)
d	display contents of the stack
x	stop

The 'd' command simply prints out the contents of the stack, one element per line, beginning with the top of the stack. The behavior of the 'e' command depends on the contents of the stack when 'e' is issued:

- If '+' is on the top of the stack, then the '+' is popped off the stack, the following two integers are popped and added, and the result of the addition is pushed back on the stack.
- If 's' is on top of the stack, then the 's' is popped and the following two items on the stack are swapped.
- If an integer is on top of the stack or the stack is empty, the stack is left unchanged.

The following examples show the effect of the 'e' command in various situations; the top of the stack is on the left:

stack before	*stack after*
+ 1 2 5 *s* . . .	3 5 *s* . . .
s 1 + + 99 . . .	+ 1 + 99
1 + 3 . . .	1 + 3 . . .

Note that 'e', 'd', and 'x' are not pushed onto the stack themselves.

You are to implement an interpreter for this language in Cool. The input to the program is a series of commands, one command per line. Your interpreter should prompt for commands with `>` at the start of each line. Your program does not need to do any error checking: you may assume that all commands are valid and that the appropriate number and type of arguments are on the stack for evaluation. You may also assume that the input integers are unsigned. Your interpreter should exit gracefully; do not call `abort()` when receiving an `x`.

You are free to implement this program in any style you choose. One approach is to define a class `StackCommand` with a number of generic operations, and then to define subclasses of `StackCommand`, one for each kind of command in the language. These subclasses define operations specific to each command, such as how to evaluate that command or display that command. If you wish, you may use the classes defined in `atoi.cl` to perform string to integer conversion. If you find any other code in `distro/examples` that you think would be useful, you are free to use that as well.

Our solution is less than 200 lines of Cool source code, including extensive comments. This information is provided as a rough measure for the amount of work involved in this assignment—your own solution might be substantially shorter or longer.

Please note: Your stack machine will be automatically tested by comparing its output to that of our reference implementation. Therefore, your stack machine should not produce any output aside from whitespace (which our testing harness will ignore), '`>`' prompts, and the output of a 'd' command. Prior to submitting, please remove any output commands that you used for debugging.

Example session

The following is a sample compile and run of our solution.

```
%coolc stack.cl atoi.cl
%coolspim -file stack.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /home/abc123/cool/distro/lib/trap.handler
>1
>+
>2
>s
>d
s
2
+
1
>e
>e
>d
3
>x
COOL program successfully executed
```

2 Getting and turning in the assignment

2.1 Initial Setup

1. Set up a (free) GitHub account if you do not have one already: https://github.com/
2. Set up the *initial* code project as a Git repository on your local machine, and push it to GitHub as a **private** individual repository. E.g.,

   - First, unzip the CW distro zip on your local machine. The initial code project is the `ECS652U-main-template` subdirectory. Change the name of the project directory to `ECS652U-cw-<myqmid>` with `<myqmid>` substituted for your QM username. *Before* modifying anything in the code project, first initialise it as a *local* Git repository.
     E.g., to do all the above on the command line

     ```
     unzip ECS652U-cw-distro.zip
     cd ECS652U-cw-distro
     mv ECS652U-main-template ECS652U-cw-<myqmid>
     cd ECS652U-cw-<myqmid>
     git init .
     git add *
     git commit -m "Initial"
     ```

     Henceforth, we will refer to (the fully qualified path to) your `ECS652U-cw-<myqmid>` directory as `$COOL_HOME`. We advise defining this as an environment variable.
   - Second, create a new **private** individual repository on GitHub, and follow the instructions for linking and pushing your local repository to the GitHub one. Verify this is all working correctly before proceeding with the actual assessment.

   It is crucial that you set your GitHub repository to **private** (double and triple check it!). In the worst case, failure to do so may amount to (inadvertant) assessment collusion (which would be a great pain and hassle for *everyone* all round).
3. Add the following two directories to your `PATH` environment variable.

   - Add `$COOL_HOME/bin/.orig/.Darwin` if you are on a Mac, or `$COOL_HOME/bin/.orig/.Linux` otherwise.[1] This is for the reference COOL compiler (`coolc`) and SPIM tools (`coolspim`).
   - Add `$COOL_HOME/bin`. This is for the build, execution and testing scripts (e.g., `buildme`, `testme`) that you will need to use for all A0–A3.

   E.g., using Bash on Linux/MacOS, the following command adds `<newpathelement>` (substitute this with each of the above) to the front of your existing PATH. (You can also add this to your config file so it is done automatically every time you open a shell.)

   ```
   PATH="<newpathelement>:$PATH"
   ```
4. After unzipping, the COOL executable files may not have executable permissions set by default. Set executable permissions for each file in the above two directories. E.g., on Linux/WSL

   ```
   cd $COOL_HOME/bin
   chmod u+x buildme filter runme testme
   cd $COOL_HOME/bin/.orig/.Linux
   chmod u+x *
   ```

   Similarly for `.Darwin` on a Mac.

[1] For Windows, follow a Linux setup using WSL 2 (it works well for me using Ubuntu/WSL 2).

2.2 A0 Instructions and Submission

1. To compile and run the template code for this assignment, run

   ```
   cd $COOL_HOME/assignments/pa0
   coolc stack.cl atoi.cl
   coolspim -file stack.s < stack.test
   ```

   Try it straight away—it should "work", and print "Nothing implemented" (among a few other things).

2. Also check the instructions in the `README` file in the `pa0` directory. (Some of the instructions are a repeat of the above.)

3. **Make sure all your code is in `stack.cl` and that it compiles and works as intended.** A copy of `atoi.cl` will be present when we test your submission, so all we need is your `stack.cl`. You can commit other files into your Git repository (e.g., tests you have created), but our marking script will ignore them.

4. Although `stack.cl` is the only file that will be marked, you will submit a zip archive of your *entire* `ECS652U-cw-<myqmid>` code repository, **including the (possibly "hidden") `.git` directory**. Failure to include the `.git` directory will count as an incomplete submission.

   For marking, we will automatically use the latest commit on the `main` branch of your repository (regardless of the commit message) at the deadline.