Programming Assignment 0: Stack

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 the 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 is pushed back on the stack.
- If 's' is on top of the stack, then the 's' is popped and the following two items are swapped on the stack.
- 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 . . .

You are to implement an interpreter for this language in Cool. Input to the program is a series of commands, one command per line. Your interpreter should prompt for commands with `>`. Your program need not 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()` after 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 it as well.

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

Please note: Your stack machine will be 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.

Sample session

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

```
%coolc stack.cl atoi.cl
%coolspim -file stack.s
SPIM Version 5.6 of January 18, 1995
Copyright 1990-1994 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README 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
```

Extra Credit

There is a chance that you will discover a bug in our Cool compiler. We will award extra credit for legitimate bug reports; to get credit, send a bug report on QM+ forum. Your report must include all of the needed Cool source and a transcript of a terminal session showing how to reproduce the bug (use the `script` command). There are a number of ways the compiler can potentially fail: the compiler may dump core, the generated code may be incorrect, the compiler may refuse to accept a legal program, it may accept an illegal program, etc. *Please be sure you have found a bug before submitting a report!* The module organizer is the final arbiter of what is a "bug" and what is a "feature". Credit usually will be awarded only to the first person to report a bug.

Getting and turning in the assignment

1. Set up an account on QMUL Github, if you haven't yet:
   http://docs.hpc.qmul.ac.uk/intro/github-only-account/.
   If you have an account, you need to login at least once, the easiest way is from the webpage
   https://github.research.its.qmul.ac.uk,
   before you can be added as a member of the github organization for ecs652
   https://github.research.its.qmul.ac.uk/orgs/ecs652.

2. On your machine, type the following commands:

   ```
   cd ~
   mkdir cool
   cd cool
   git clone -o distro https://github.research.its.qmul.ac.uk/ecs652/distro.git
   cd  distro
   git remote add origin https://github.research.its.qmul.ac.uk/ecs652/<your-user-name>.git
   git push -u origin master
   ```

   For this assignment, ⟨`your-user-name`⟩ is your git username, which is the same as you college username.

   These commands create everything you need to start working on the assignment. They also set up an upstream git repository that you should use to save your work, and finally submit it.

3. Add `bin` to the `PATH`. For example, on Linux with bash shell, type

   ```
   export PATH=~/cool/distro/bin:$PATH
   ```

4. To compile and run the template code, type

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

   Try it now – it should work, and print "Nothing implemented" (among a few other things).

5. Follow the instructions in the README file in pa0 directory.

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

7. To turn the assignment in, commit your final version of stack.cl and push to your upstream repository:

   ```
   git add stack.cl
   git commit stack.cl -m "Final version"
   git push
   ```

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