Programming Assignment I
Due Tuesday, Sept. 6

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 with a partner; you should turn in your own individual work. All future programming assignments will be done in teams of two. Remember that there is a 25% penalty for each day or part of a day that an assignment is late.

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.

You are free to implement this program in any style you choose. However, in preparation for building a Cool compiler, we recommend that you try to develop an object-oriented solution. One approach is to define a class `Stack` with a number of generic operations, and then to define subclasses of `Stack`, one for each kind of command in the language. These subclasses define operations specific to each command,

such as how to evaluate that command, display that command, etc. If you wish, you may use the classes defined in `atoi.cl` in the `~cs164/examples` directory to perform string to integer conversion.

We wrote a solution in approximately 130 lines of Cool source code. 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.

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

```
%coolc stack.cl atoi.cl
%spim -file stack.s
SPIM Version 5.4 of Jan. 17, 1994
Copyright 1990-1994 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README a full copyright notice.
Loaded: /home/n/cs164/lib/trap.handler
>1
>+
>2
>s
>d
s
2
+
1
>e
>e
>d
3
>x
COOL program successfully executed
```

Getting and turning in the assignment. Create a working directory and `cd` into it. From there, type

```
% make -f ~cs164/assignments/PA1/Makefile
```

This command creates several files you will need in the directory. Follow the directions in the README file. The README file explains how to turn in your assignment when you are finished; it also contains a few questions you are to answer as part of the assignment.

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 to `mjacoby@cs`. 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 course staff are the final arbiters of what is a "bug" and what is a "feature". Credit usually will be awarded only to the first person to report a bug.