Programming Assignment I
Due Thursday, Sept. 22 at Midnight

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.

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 content 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 off 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 subclass 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 the command, etc. If you wish, you may use the classes defined in **atoi.cl** available in your working 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.

Setup your environment: login your account at **wozniak.csustan.edu**. Add the following line at the end of the file .bashrc in your home directory

```
export PATH=/home/CS4300/bin:$PATH
```

Then type the Unix command

```
$ source .bashrc
```

Create a working directory, say, PA1, and cd **into it**: Type

```
$ mkdir PA1
$ chmod 700 PA1
$ cd PA1
```

Get the assignment files:. From working directory, type

```
$ make -f /home/CS4300/assignments/PA1/Makefile
$ chmod 600 README stack.cl
```

This command creates several files you will need in the directory. Follow the instructions in the README file. Note: It also contains a few questions you are to answer as part of the assignment.

Compile and test your program: To compile and test your program, type

```
$ make compile
$ make test
```

You can also run your compiled assembly code **stack.s** directly for testing. For example:

```
$ make compile
$ spim 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/CS4300/lib/trap.handler
>1
>+
>2
>s
>d
s
2
+
1
>e
>e
>d
3
>x
COOL program successfully executed
```

Note: the assembly code for a working stack.cl is in file stack_ex.s in your working directory. You can compare the results from your program with results of stack_ex.s to test whether yours is operating correctly.

Turn in your assignments: to submit your assignments, type:

```
$ make submit
```

Please review the late submission policy. An assignment is late if any of the electronically submitted files are late, including the answers to README questions. You should take reasonable precautions to prevent the theft of your code. (i.e. don't leave printouts in public recycle bins, don't allow group or world access to your files, don't give your password to a classmate etc..)

The following lines show you how to create your working directory and get your programming files with security in mind.

```
$ mkdir PA1
$ chmod 700 PA1
$ cd PA1
$ make -f /home/CS4300/assignments/PA1/Makefile
$ chmod 600 README stack.cl
```