Programming Assignment I

1 Overview of the Programming Project

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 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. 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 `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, display that command, etc. If you wish, you may use the classes defined in `atoi.cl` in the `[cool root]/examples` directory to perform string to

integer conversion. If you find any other code in `[cool root]/examples` that you think would be useful, you are free to use it as well.

We wrote a solution in approximately 200 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.

2 Sample Session

The following is a sample compile and run of our solution (substitute the root directory of your workspace for `[cool root]`).

```
% [cool root]/bin/coolc stack.cl atoi.cl
% [cool root]/bin/spim -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: [cool root]/lib/trap.handler
>1
>+
>2
>s
>d
s
2
+
1
>e
>e
>d
3
>x
COOL program successfully executed
```

3 Files and Directories

To get started with the programming assignments, download the starter code from the OpenClassroom website and extract it to a convenient directory on your local machine. Make sure you download the tarball that matches your particular machine architecture. You may also download the pieces of this assignment individually from the Resources page, but we strongly recommend that you download and use the complete tarball as is.

Once you have a working copy of the programming assignment source tree, change into the directory for the current assignment,

```
[cool root]/assignments/PA1/
```

Some of the files in the directory will be read-only (using symbolic links). You should not edit these files. In fact, if you make and modify private copies of these files, you may find it impossible to complete the assignment. See the instructions in the `README` file.

4 Testing

You can test your stack machine by comparing its output to that of our reference implementation using `make test`. 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.