CS164 Programming Language and Compilers **Spring 2016**

Programming Assignment I

Assigned: January 25, 2016 **Due:** February 2, 2016 at 11:59pm

Instructions: 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 either one or two.

1 Stack Machine in Cool (50 points)

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 followed by a newline per line and nothing else, 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` file included in the assignment archive file to perform string to integer conversion. If you find any code in `examples` directory of the archive file 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.

Sample session

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

```
%python coolc.py stack.cl atoi.cl
%python runmips.py stack.s
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

>1
>+
>2
>s
>d
s
2
+
1
>e
>e
>d
3
>x
COOL program successfully executed
```

Getting the assignment

Everything you need is included in the archive file (PA1.zip) available from the course website. Please read the directions in the `README` file for more details.

Using your own machine

If you want to use your own machine for the development, you will need to install JDK 7, Apache Ant 1.7 (or newer), and Python 2.7. You can get the packages from the following links:

- JDK 7: `http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`
- Apache Ant: `http://ant.apache.org/bindownload.cgi`
- Python 2.7.11: `https://www.python.org/downloads/`

After installing a package, please put the path to the bin folder of the installed package in your `PATH` variable. In other words, `java`, `javac`, `ant`, and `python` commands should run on the shell prompt without the exact path to the corresponding bin folder. You also need to set `JAVA_HOME` and `ANT_HOME` environment variables.

Mac and Linux. You can also use a package manager for your system, such as `apt-get` and `brew`, to install a necessary package. Just make sure that you are using the correct version. Package managers do not set

`JAVA_HOME` and `ANT_HOME` environment variables for you. So, please make sure to set these variables.

Windows. Please check the following guides to setup a development environment on a Windows machine:

- *JDK installation*: `http://docs.oracle.com/javase/7/docs/webnotes/install/windows/jdk-installation-windows.html`
- *Setting the JAVA_HOME Variable*: `https://confluence.atlassian.com/doc/setting-the-java_home-variable-in-windows-8895.html`
- *Python on Windows*: `https://docs.python.org/2/using/windows.html`
- *Installing Apache Ant*: `http://ant.apache.org/manual/install.html`

Turning in the assignment

1. Make sure your code is in `stack.cl` and that it compiles and works. A copy of `build.xml` and `atoi.cl` will be present when we test your submission, so all we need from you is `stack.cl` (45 points) and `README` (5 points).
2. Answer the 3 required questions in the `README` file, and include any other relevant comments here.
3. Upload everything you need to submit to your instructional account if you are using your own development machine.
4. Make sure everything is in a directory called `PA1` (in your instructional account.)
5. Please make sure to register your account by executing `register` command before executing the submission script.
6. Run the command `submit PA1` (case is important) from your `PA1` directory (in your instructional account.)

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.

Hints

- Read the Cool Manual - it describes the Cool language thoroughly.
- Play around with the example Cool programs (in `examples` directory) if you are having difficulty learning the syntax.
- CoolToJS (linked on the course page) is useful to test, but it does not fully implement Cool correctly.

2 Find a Bug! (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, post the bug report on Piazza. Your report must include all of the needed Cool source and a transcript of a terminal session showing how to reproduce the bug (consider using the `script` command). There are a number of ways the compiler can potentially fail: the compiler may throw an exception, 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.