CS-GY 6413/CS-UY 3943 — Programming Assignment 1

Jeff Epstein

Introduction

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.

The tools we use in this class presuppose a Linux environment. In particular, you'll need to be able to run Linux binary programs. Before starting, please make sure that you're using a suitable programming environment:

- I recommend using the NYU Vital cloud infrastructure. Please refer to the Vital guide on Classes.
- You may install a Linux virtual machine on your own computer. Please refer to the VirtualBox tutorial on Classes.
- You may use the Windows Subsystem for Linux. Please refer to the official documentation.
- You may work directly on your own computer, if you have an appropriate operating system. I recommend a recent Debian or Ubuntu distribution.

To learn the Cool language, please read the Cool manual, specifically the sections "Getting started" and "Informal language description."

Collaboration policy Collaboration of any kind is prohibited on this assignment. All submitted work must represent the effort of a single student.

Assignment

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, exit program

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.

removed
- 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 the provided `atoi.cl` file to perform string to integer conversion.

Example The following is a sample compile and execution. Make sure you understand what's happening in this execution before you start coding.

```
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /home/COOL/cool/refimpl/trap.handler
>1
>+
>2
>s
>d
s
2
+
1
>e
>e
>d
3
>x
COOL program successfully executed
```

Prints out stack immediately after call. Notice that s does not swap until e is called

1
2
+
d

call +

exit

Setup and testing

Download two files: `pa1.zip`, which contains starter code specifically for this assignment; and `refimpl.zip`, which contains a reference implementation of the Cool compiler. Extract the content of both zip files into one directory.

Running the `make` command from that directory will cause the `stack.cl` file to be compiled with the reference compiler and subsequently run with the test input data in the file `stack.test`. Please note, however, that this test data is not sufficient to guarantee that your implementation is correct: you must devise your own test cases.

Your stack machine will be tested by comparing its output to that of the reference implementation. Therefore, your stack machine should not produce any output aside from whitespace, '`>`' prompts, and the output of a 'd' command. Prior to submitting, please remove any output commands that you used for debugging.

Rules

Your code should compile and run cleanly with the provided `Makefile`.

Do not add any additional files to the program. Modify only those files indicated in `README`.

Use good coding style, including informative variable names, consistent indentation, and clear structure. Your code should be robust and generally free of bugs.

You are not obligated to use comments. However, confusing, unclear, and uncommented code may result in a reduced grade. The grader must be able to understand your code.

Submission

Test your code thoroughly before you submit.

In addition to your code, you must answer the questions at the end of the provided `README` file. Write your answers directly in the `README` file.

Run the command `make zip` and check the content of the resulting zip file to make sure that it contains all the code you wrote. Submit only the zip file on Gradescope.

Acknowledgments

This sequence of assignments is based on work by Alex Aiken and Westley Weimer.