CMPSC 112
Introduction to Computer Science II
Fall 2016

Laboratory Assignment Nine: Performing Arithmetic Interpretation with a Stack

Introduction

Many real-world programs use the stack abstract data type (ADT) to solve a problem. For instance, the Java virtual machine (JVM) relies on a stack to store the activation records for the methods that are called during the execution of a program. Moreover, stacks are often implemented as part of the hardware used in many modern computers. In this laboratory assignment, you will learn how to use the stack ADT in a Java program. In particular, you will implement and test a `StackMachine` that provides a simple language supporting select arithmetic operations.

Learning About the Stack Abstract Data Type

Before you start to implement your `StackMachine`, you should take some time to learn more about the stack ADT. What is an example of a problem that can be solved using the stack? Your response to this question should include a brief introduction to the problem and a discussion of how the stack is incorporated into the solution. For instance, you could explain a program that you have already implemented that manipulated a stack. Along with reading Section 6.1 of your textbook to review the fundamental stack operations, you can study books, articles, and online sources to investigate a well-known problem that can be solved through the use of a stack. For instance, you could examine the algorithm that converts an expression in infix notation to postfix notation.

Implementing an Arithmetic Interpreter Using a Stack

In this assignment, you will write a Java program that constructs an instance of the `java.util.Stack` class. What are the key methods provided by `java.util.Stack`? What are the inputs and outputs of the key methods? What is the behavior of these methods? Please make sure that you review your textbook and consult the instructor if you do not understand the methods of the stack.

A program that only uses a single stack for storage sometimes is called a "stack machine". A stack machine normally supports a language describing the operations that it can perform. Consider the following language for a stack machine that enables several basic arithmetic computations:

Command	Meaning
int	push *int* on the stack
+	push a "+" onto the stack
s	push an "s" onto the stack
d	display contents of the stack
x	stop the stack machine
e	evaluate the top of the stack (see the following commentary)

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 the "e" is issued. For this laboratory assignment, the "e" command must follow these rules:

1. If a "+" 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.
2. If an "s" is on top of the stack, then the "s" is popped off the stack and the following two elements are swapped on the top of the stack.
3. If an integer is on the top of the stack or the stack is empty, then the stack is left unchanged.

The following examples illustrates the impact of the "e" command. Note that in this table, the top of the stack is on the left and the notation "..." denotes the rest of the stack's contents.

State of Stack Before "e"	State of Stack After "e"
+ 1 2 5 s ...	3 5 s ...
s 1 + + 10 ...	+ 1 + 10 ...
1 + 3 ...	1 + 3 ...

You must implement an interpreter for this arithmetic language. The input to your program is a series of commands, with one per line. The interpreter should prompt for commands with >>>>> or something else appropriate. You do not need to do any error checking. That is, you can assume that all commands are valid and that the appropriate number of arguments are on the stack for evaluation. However, you may implement a sensible exception handling system for extra credit. Finally, you should be aware of the fact that you will have to decide what types of variables are going to be pushed onto and popped off of the `java.util.Stack` maintained by the `StackMachine`; this choice may influence the types of casting that the methods will perform.

As previously mentioned, you should implement a `StackMachine.java` file that declares and uses an instance of `java.util.Stack`. Students who are interested in earning extra credit can use the examples in the book to implement their own `Stack` as an alternative to the `java.util.Stack`. Since your `StackMachine` must accept input from the user, you should leverage the `Scanner` class, as described in Section 1.6 of your textbook. Please see the instructor if you have any questions.

Summary of the Required Deliverables

This assignment invites you to submit a signed and printed version of the following deliverables:

1. A complete description of a problem that can be solved by using the stack abstract data type.
2. The properly commented version of `StackMachine.java` and any other Java files you create.
3. The output from five separate runs of the `StackMachine` program, demonstrating correctness.
4. A written report that provides a response to all of the questions posed in this assignment.
5. A reflective commentary on the challenges that you faced when completing this assignment.

Along with turning in a printed version of these deliverables, you should ensure that everything is also available in the repository that is named according to the convention `cs112F2016-<your user name>`. Please note that students in the class are responsible for completing and submitting their own version of this assignment. While it is acceptable for members of this class to have high-level conversations, you should not share source code or full command lines with your classmates. Deliverables that are nearly identical to the work of others will be taken as evidence of violating the Honor Code. Please see the instructor if you have questions about the policies for this assignment.