CP2561 Project

Scott R. Young

June 29, 2023

Abstract

This project will cover material taught in this course and its prerequisites. Your objective is to make a functioning stack-based interpreter for a simple stack-based language based on the Forth programming language.

1 Operation

Your project, when compiled and run, will take a file-path as a command-line argument. It will confirm that the file contains a valid program. If it does not, it will print an error to the screen; if it does, it will execute the program. The programs will consist of a number of "Words" which are read in a left-to-right; top-to-bottom order.
Each "Word" is executed immediately, and each word fits into one of five categories: "Numbers", "Quotes", "Definitions", "Stack-Operations" and "I/O".

2 Your Task

Your task will be to:

- Implement the reading of a text file. You will turn the contents of the file into a List of strings, which do not contain any white-space, where the order of Strings in the list is a left-to-right; top-to-bottom ordering of the "Words"
- Starting with the first String in the list, your program will attempt to classify each word into one of the five categories. If it cannot classify a word, it will print an error and fail validation. It will, at this stage, return a list of "Words" rather than a list of Strings.
- Your program must then execute the "Words" in order. The types of "Words" your program must be able to execute are:
  - "Numbers": All number literals, both integers and reals, will be considered number words.

- "Quotes": Only one word is a quote and that is the ' character. This will tell the interpreter to put a String on the stack, which will be all the words after this quote until the next quote word.
- "Definitions": Only one word is a definition word, and that is the : character. This words will allow users to define their own words. Attempting to use a word before it is defined is an error, but once a word is defined it should be valid to use in the program. The first word after a definition is the new word defined by the definition, and must not be a "Number", "Quote", "Stack-Operation" or "I/O" word. Every word after the new word, until another definition word has been reached, is the code to be executed when this newly defined word is used in the future.
- "Stack-Operations": These words will modify the stack by either duplicating, removing, or reordering the contents of the stack.
- "I/O": These words will either read a "Word" from user input onto the stack, or print the value on the top of the stack to standard output.

3 Predefined words

The following words are predefined, meaning that they are either a "Definition", "Quote", "Stack-Operation" or "I/O" word.

- Definition
  - : — begin defining a word, the next word will be the new word to recognize, every word after that until the next : will be its definition.
- Quote
  - ' — Begins or ends a string. The string will be the concatenation of every word between the ' words, delimited by spaces. When the second ' in a pair is reached the string will be pushed onto the stack.

- I/O
  - in – Reads input from standard input as a string and pushes it on the stack.
  - out – Prints the value on the top of the stack to standard output
- Stack-Operations
  - dup – Duplicate the top entry of the stack and place it on top of the stack.
  - swap – Swap the top two entries on the stack
  - pop – Remove the top element of the stack
  - + – Pop the top two elements off the stack, if either is a String then concatenate the elements as strings. If both elements are numbers then add the numbers. Push the result onto the stack.
  - * – Pop the top two elements off the stack, if either is a String then treat the top element as a regex and the next as a String. Use the regex to find the index of the first match in the string, and push the String from that index onwards onto the stack. If both elements are numbers then multiply the numbers and push the result onto the stack.
  - - – If the top element is a number, negate it. If it is a string, reverse it. Push the result on the stack.

4 Example Programs

Here are a list of valid programs your interpreter must be able to execute, as well as the expected results.

• 2 3 4 * + out	• 14
• : fifteen 3 3 4 * + : fifteen fifteen + out	• 30
• ' Hello World! ' out	• Hello World!
• in out	• This program will echo whatever the user types in.
• 5 - out	• -5
• ' Hello ' ' World! ' + out	• Hello World!
• ' Hello World!' ' e ' * out	• ello World!
• ' Hello World!' - out	• !dlroW olleH

5 Limitations

- You are limited to the techniques that have been demonstrated in this course through its assignment, handouts, lecture notes and the course textbook.