Project 2

2023/10/27

100 Possible Points

Attempt 1

In Progress
NEXT UP: Submit Assignment

Add Comment

Unlimited Attempts Allowed
2023/10/7

Details

CS 3304 Project #2

In this assignment, you are going to implement an interpreter for (a subset of) the language Forth using Haskell. We begin by describing Forth and then some guidelines on implementing your interpreter.

What is FORTH?

Forth is a cute language meant for control and embedded systems applications. It comes with a lot of functionality, but for this assignment, you will be implementing only a small subset of the entire language. You can read more about Forth at, e.g., the pForth website (http://www.softsynth.com/pforth/).

The central entity in Forth is a stack. You push things onto the stack and do operations from there. A Forth program is a bunch of "words" with spaces between them. For instance, the following is a simple Forth program:

```
23 7 91
```


All it does is to push the three above numbers onto the stack. 91 is on top and 23 is at the bottom (last in, first out).

Here's another Forth program:

```
23 7 91 .
```

The only difference between this and the previous is the "dot" (.) at the end of it. The "dot" pops (removes) the top element off the stack and prints it. After this, only two numbers are remaining (23 and 7) on the stack. 7 is on the top.

The next command we will encounter is ".S" (dot-S). Consider the program:

```
23 7 91 .S
```

This causes as output "23 7 91". dot-S prints the entire stack (topmost element is printed rightmost) but doesn't remove anything.

So, we can put things onto the stack and remove them. The following program leaves the stack empty, but causes as output "91 7 23".

```
23 7 91 . . .
```

We can also do arithmetic operations on the stack. "+" takes the top two elements off the stack, adds them, and pushes the result back. Similarly for "-", "*", and "/". So,

```
4 5 + .
```

causes "9" to be printed and leaves the stack empty.

```
4 5 - .
```

causes "-1" to be printed and leaves the stack empty.

Submit Assignment

```
77 DUP .S
```

causes “77 77” to be printed (and the stack now contains these two integers). Similarly, SWAP swaps the topmost two elements on the stack.

```
8 7 SWAP .S
```


prints “7 8”. DROP drops the topmost element from the stack (unlike “dot”, it doesn’t print anything). OVER causes a copy of the second element on the stack to leapfrog over the first. So,

```
8 9 OVER .S
```

causes “8 9 8” to be printed (and r ROT takes the third element from the stack and moves it to the top.

```
7 8 9 ROT .S
```

prints “8 9 7” (and this also remain mples,

```
11 22 33 SWAP DUP
```

causes “11 33 22 22” to be on the stack (doesn’t print anything). Similarly,

```
11 22 33 ROT DROP
```


causes “22 33” to be on the stack (doesn’t print anything). For your assignment, you only have to support the above commands (words). Furthermore, you can assume that only integers can be pushed onto the stack. A syntactically correct FORTH program is simply a string that contains any combination of integers and FORTH words, separated by white space. When printing using “dot” or “dot-S” you can assume that all printings are on the same line, with spaces in between the elements printed.

Command Summary

- `.` pops the top of the stack and displays it.
- `.S` displays the stack with the top on the right
- + * / - performs the math operator on the top two stack operands and pushes the result back on the stack
- `DUP` duplicates the top element on the stack
- `SWAP` switches the first and second two elements on the stack
- `DROP` drops the topmost element from the stack
- `OVER` copies the second element in the stack and pushes it on top
- `ROT` takes the third element on the stack and puts it on top

An Interpreter in Haskell

The operation of the interpreter is to read a set of lines from a file named named as the first commandline argument (which will be in the current directory), where each line is a string denoting a syntactically correct FORTH program. The interpreter will process this program and return a tuple (a,b) which is printed onto a file named as the second commandline argument. “a” is a string that contains the result of all the printings that happen during the execution of the program. “b” is a string that identifies what is remaining on the stack (again topmost element to the right). For each line in the input, there is a corresponding line in the output.

For instance, for the input

```
11 22 33 ROT DROP .
```

the output is

```
("33","22")
```

(yes you have to print those brackets and quotes). For the input

```
11 1 3 4 + 5 7 DROP . .S
```

the output is

```
("5 11 1 7","11 1 7")
```

If something illegal is attempted, your interpreter should print

```
(illegal,illegal)
```

This can happen, for instance, when dot/DUP/DROP is attempted on an empty stack, the arithmetic operations/SWAP/OVER are attempted on a stack that contains less than two elements, or when ROT is attempted on a stack that contains less than three elements.

If nothing is printed by the program, or if the stack is empty, you should substitute an empty string ("") in the appropriate place in the returned tuple.

Submit Assignment

```
3 2 . .
```

produces

```
("2 3","")
```


Implementation Guidelines

Your implementation must adhere to the tenets of functional programming. Specifically, your program must utilize recursion and higher-order functions to the fullest extent. It m ts with the exception of input/output.

You may not use any imperative fe the language does have some impurities). You may use `let` s however as they do not run contrary to the spirit of functi y no iteration is permitted (use recursion).

Adopt an iterative development ap terpreter to first handle simple pushing and popping off stack, and then proeed to add the more complex FORTH wo

You might want to check out the H https://hackage.haskell.org/package/base-4.14.1.0/docs/Prelude.html) to see if there are any useful functions and datatypes provided. See also the code from the lecture summaries. (These will be coming soon, i.e. next week at the latest)

Submitting Your Program

Your Haskell implementation is to be organized in a single file. The file will use a .hs extension. Haskell is installed on rlogin. I'm pretty sure you can get easy to install Haskell installers for all major OSs.

All documentation and identifying information should be placed in comments within the single source file. The comments at the beginning of the file should identify the assignment and give your full name. Every Haskell function should be preceded by comments explaining its purpose, the meaning of each parameter, and the general strategy of its implementation if applicable.

Your program will be **submitted electronically** to Canvas. It will be scored 75 points for correctness and the TAs will assess the remaining 25 points based on the following requirements.

Requirements

1. Your program must take 2 commandline arguments.
   a. The first is the name of the input file to use.
   b. The second is the name of the output file to use.
2. You must use a recursive functional approach.
   - Do not use iteration, with the exception of line process. See the homework 5 main for ideas.
3. You must comment your code and each function as described above.
4. This is an individual assignment.
5. You must follow the Implementation Requirement as stated below.

Implementation Requirement

Do not reverse the list and process it "backwards". Simply process the head and then process the tail. This is how functional programming wants to operate and if you attempt to force a procedural approach your going to have a hard time.

Sample Files

samples.zip (https://canvas.vt.edu/courses/176190/files/29188107?wrap=1) (https://canvas.vt.edu/courses/176190/files/29188107/download?download_frd=1)

Grading

You will receive at most 75 points from the grader. The rest of the 25 points will come from meeting the stated requirements. These points will be assigned after the due by the TAs.

Choose a submission type

Upload More

Submit Assignment

Choose a file to upload
File permitted: HS

or

Canvas Files

Submit Assignment