CS 320: Language Interpreter Design Part 1

Part 1 Due: November 11th 11:59pm EST

1 Overview

The project is broken down into three parts. Each part is worth 100 points.

You will submit a file named `interpreter.ml` which contains a function, `interpreter`, with the following type signature:

```
interpreter :  string -> string -> unit
```

2 Functionality

The function will take a program as an `input` string, consisting of the program lines, and will take in the output file path as the second parameter, and will return a `unit`, as your evaluated output will be logged into a file.

Because the function evaluates a stack-based programming language, a stack is used internally throughout the interpreter to keep track of intermediate evaluation results. Note that the stack will not be checked, but rather what is written in the output file — the stack and the output file are two different things.

The autograder will check your output file's content, and more information on the output file will be described below.

3 Part 1: Basic Computation
Due Date: November 11th 11:59pm EST

3.1 Grammar

For part 1 you will need to support the following grammar

3.1.1 Constants

<digit> ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

<int> ::= [−] <digit> {<digit>}

<char> ::= `a` | `b` | `c` | ... | `y` | `z` | `A` | `B` | `C` | ... | `Y` | `Z` (aka the alphabet)

<string> ::= "{<char>}"

<const> ::= <int> | <string>

3.1.2 Programs

<com> ::= `Quit` | `Push` <const> | `Pop` | `Add` | `Sub` | `Mul` | `Div` | `Swap` | `Neg` | `Concat`

<prog> ::= <com> <prog> | <com>

3.2 Errors

In part 1, when an error occurs during interpretation, evaluation must stop immediately and output the exact log `"Error"` into the provided output file (see `3.3.1` for more information about the output file).

3.3 Commands

Your interpreter should be able to handle the following commands:

3.3.1 Quit

This command causes the interpreter to stop and *does not have to be* on the last line of the program. When called, the created stack should be written out to an output file that is specified as the second argument to the main function, and no commands after the Quit call get evaluated.
Example 1

```
Push 1
Quit
```

should result in the following stack and the following contents in the output file:

```
1
```

Example 2

```
Push 1
Quit
Push 4
Quit
Push "candy"
```

should result in the following stack and the following contents in the output file:

```
1
```

Example 3

```
Push 1
Push 4
Push "candy"
```

should result in the stack

```
"candy"
4
1
```

and a file with nothing inside (this will be checked)

3.3.2 Push

All kinds of *const* are pushed to the stack in the same way. Resolve the constant to the appropriate value and add it to the stack.

The program

```
Push 12
Push "abc"
Push -6
Push ""
Quit
```

should result in the following stack and the following contents in the output file:

```
""
-6
"abc"
12
```

3.3.3 Pop

The command Pop removes the top element from the stack. If the stack contains less than 1 element, terminate evaluation with error.

Example 1

```
Push 33
Push ""
Push "a"
Pop
Quit
```

should result in the following stack and the following contents in the output file:

```
""
33
```

Example 2

```
Push 5
Pop
Pop
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

3.3.4 Add

Add consumes the top 2 values in the stack, and pushes their sum to the stack. If there are fewer than 2 values on the stack, terminate with error. If not all of the top 2 values on the stack are not integers, terminate with error.

Example 1

```
Push 5
Push 7
Add
Push 3
Add
Quit
```

should result in the following stack and the following contents in the output file:

```
15
```

Example 2

```
Push "hello"
Add
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

Example 3

```
Push "bottle"
Push "water"
Add
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

3.3.5 Sub

Sub consumes the top 2 values on the stack, and pushes the difference between the top value and the second top value to the stack.

If there are fewer than 2 values on the stack, terminate with error. If the top 2 values on the stack are not integers, terminate with error.

Example 1

```
Push "test"
Push 1
Push 10
Sub
Quit
```

should result in the following stack and the following contents in the output file:

```
9
"test"
```

Example 2

```
Push "test"
Push 1
Push 3
Push 4
Push 10
Sub
Quit
```

should result in the following stack and the following contents in the output file:

```
6
3
1
"test"
```

Example 3

```
Push "choco"
Push 10
Sub
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

3.3.6 Mul

Mul consumes the top 2 values in the stack, and pushes their product to the stack.

If there are fewer than 2 values on the stack, terminate with error. If the top 2 values on the stack are not integers, terminate with error.

Example 1

```
Push 5
Push 7
Mul
Quit
```

should result in the following stack and the following contents in the output file:

```
35
```

Example 2

```
Push 2
Push "laddoo"
Mul
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

Example 3

```
Push "coffee"
Push "cream"
Mul
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

3.3.7 Div

Div consumes the top 2 values on the stack, and pushes the quotient between the top value and the second top value onto the stack.

If there are fewer than 2 values on the stack, terminate with error. If the second value is 0, terminate with error. If the top 2 values on the stack are not integers, terminate with error.

Example 1

```
Push 2
Push 10
Div
Quit
```

should result in the following stack and the following contents in the output file:

```
5
```

Example 2

```
Push "samosa"
Push 10
Div
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

Example 3

```
Push 5
Push 5
Sub
Push 10
Div
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

3.3.8 Swap

Swap takes the top 2 elements in the stack and swaps their order. If there are less than 2 items in the stack, terminate with error

Example 1

```
Push "hello"
Push "world"
Swap
Quit
```

should result in the following stack and the following contents in the output file:

```
"hello"
"world"
```

Example 2

```
Push "320"
Push "cas"
Push "cs"
Swap
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

Example 3

```
Push 0
Swap
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

3.3.9 Neg

Neg would negate the top element on the stack. If the top element in the stack is not an integer, terminate with error. If the stack is empty, terminate with error.

Example 1

```
Push 2
Push 10
Neg
Quit
```

should result in the following stack and the following contents in the output file:

```
-10
2
```

Example 2

```
Push 5
Push "kenny"
Neg
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

Example 4

```
Neg
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

3.3.10 Concat

Concat consumes the top 2 values on the stack, and pushes the concatenation between the top value and the second top value onto the stack.

If there are fewer than 2 values in the stack, terminate with error. If the top 2 elements in the stack are not of *string* type, then terminate with error.

Example 1

```
Push "lemon"
Push "laddoo"
Concat
Quit
```

should result in the following stack and the following contents in the output file:

```
"laddoolemon"
```

Example 2

```
Push 3
Push "peanut"
Concat
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```

Example 3

```
Push "chocolate chip"
Push "cookie"
Concat
Quit
```

should result in the following stack and the following contents in the output file:

```
"Error"
```