CSE2031

Lab 3
Summer 2016

In this lab, you will implement a simple RPN calculator. Since this is your third lab, and you haven't seen pointers and string manipulation yet, your calculator is a very simple one without any input validation or error checking. Later on, you will write a more sophisticated RON calculator. For now, we assume that the input is given to you without exactly as stated.

Reverse Polish Notation

RPN is a mathematical notation in which the operator follows all of its operands. https://en.wikipedia.org/wiki/Reverse_Polish_notation

For example, in the normal notation we are used to, adding 2 numbers looks like

3 + 4 the first operand (3) followed by the operator (+) then the second operand (4) the operator takes these 2 operands and does the operation (7).

In RPN, the operator follows the operands, we assume that operators and operands are put in a stack, operands are pushed in a stack, then the operator pops (removes) its operands from the stack, performs operation, and pushes the result on to the stack.

Assume the top of the stack is to the right, we push on the stack the numbers 4, 3, 6, and 8, the stack looks like

4,3,6,8 (8 is at the top of the stack)

The next element is the '+' operator, it needs two operands to add them. It pops the top 2 elements in the stack (6 and 8) adds them (14) and pushes the 14 back on to the stack. Now the stack looks like

4,3,14

If the next input is a number (8) it will be pushed on the stack, now the stack is

4,3,14,8

The next input is '*' it removes 14 and 8, multiply them (112) and pushes the result on the stack, now the stack looks like
4,3,112

The next input is '/' then it removes 112, and 3. Divides 112/3 = 37 (integer division) and pushes back the result on the stack, now the stack looks like

4,3,37

That means the input is either an integer, or a character. You don't know before hand what the input will be. That means, you cannot use %s or %d in your scanf. The solution is easy, there is a function called

```
int atoi(string)
```

That is ASCII to int, string is an array of character. That function takes a string, and returns the binary number corresponding to the string.

For example,

```
atoi("1234")   returns   1234
atoi("0")      returns   0
atoi("As&*")   returns   0
```

The last atoi returns a 0, in this case 0 means invalid integer. So, when you receive a 0, it means the input was a 0, or invalid. We cannot tell (for now, later on we will learn how to do this). For now, I will not test invalid input, so a 0 means the integer ZERO.

C code

Practice

Write a C program for a RPN calculator. The calculator reads its input from the **standard input**, and displays the result (if requested) on the **standard output**. It should do the following:

- If the input is an integer, the integer is stored on the stack.
- The stack has a maximum length of 50 elements.
- The allowed inputs are integers, '+', '-', '*', '/', 'p', and 'q'
- If the input is an operator (+, =, * or /), then the top 2 elements are popped out of the stack, the operation is performed, and the result is pushed back on the stack.
- If the input is 'p', the calculator prints on the screen the top of the stack. The top should be printed as integer in 10 spaces right justified followed by a new line.
- If the input is 'q', the program terminates.

Sample input: 3 4 – 5 + 6 * 7 / p 7 7 / p q
Sample output: 5 1

Binary adder:
The input is three strings, the first and third are binary strings of size 32 max. The second string is either "+" or "-".

The first and third strings are binary representation of an integer number. Your program will perform addition or subtraction in binary and display the result in binary. Assume there is no overflow or underflow.

For example, the inputs are
01010010
+
00110101
_____
1000111

Zeros to the left may be omitted, for example
11010
+
11
that means
11010
+
00011
_____
11101

When you display the answer, display only the non-zero bits followed by a newline.

Two's complement:
The program reads a string that represents a binary number and displays the two's complement.

The input could be up to 32-bit number (a string of up to 32 characters of 1's and 0's only similar to the input in the previous exercise).

Display the number as 32-bit number, or you can omit the 0's or 1's to the left (you must have at least one 0 or one 1 as the leftmost digit in order to differentiate between positive and negative numbers).