

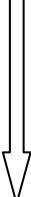
An Expression Evaluator for Postfix Hexidecimal Notations

You are to write a Python program that computes the value of bitwise manipulated expressions provided in postfix hexadecimal notation. For example, given the string "8<>1><|~" (infix: "~(><8|<>1)"), your calculator will compute "f" as the answer. All strings provided will be valid. All numbers will be single hexadecimal digits (0-f).

Note (1): Shifts are all unsigned shifts. For example, the result of "8>" will be a hex "4"; the result of "8<" will be a hex "0".

Note(2): All expressions and intermediate computations are restricted to 4 bits only. Hence, the result of "3~" will result in a hex "c".

The bit manipulation operators are:

|     |                       |   |                          |   |
|-----|-----------------------|---|--------------------------|---|
| "~" | bitwise NOT           | } | <b>RIGHT</b> associative | Highest precedence<br><br>Lowest precedence |
| ">" | bitwise shift right 1 |   |                          |   |
| "<" | bitwise shift left 1  |   |                          |   |
| "&" | bitwise AND           |   | <b>LEFT</b> associative  |   |
| "^" | bitwise XOR           |   | <b>LEFT</b> associative  |   |
| " " | bitwise OR            |   | <b>LEFT</b> associative  |   |

- Your calculator will use a stack to compute/store all intermediate computations.
- You will implement your own push and pop stack operations.
- The **ONLY** library or built-in methods that you can use are *len* and the "casting" methods, e.g. *int* and *hex*.
- Obtain the python 3.4 interpreter from [www.python.org](http://www.python.org). Additional elaboration and requirements will be forthcoming in class.

**Your program must conform to the structure and specs given on the following page.**

Python file

```
PF1
PF2
tos
    Push (stack, element)
    Pop (stack)
    return element
BitTwiddle (expression)
    stack definition/allocation
    Shift (direction, expr)
    return value
    Not (expr)
    return value
    And (expr1, expr2)
    return value
    Xor (expr1, expr2)
    return value
    Or (expr1, expr2)
    return value
    :
    process expression
    :
BitTwiddle (PF1)
BitTwiddle (PF2)
BitTwiddle ('f9>&3~8^|c~b| |')
```

- PF1 and PF2 are initialized (as strings) to the following postfix expressions, respectively
    - "f>f<1&|" (infix: ">f|<f&1")
    - "1<f>&>8|9|3~7&^" (infix: ">(<1&>f)|8|9)^(~3&7)")
  - "tos" is the *global* top-of-stack variable
  - the "stack" is defined/allocated in BitTwiddle
  - note procedure nesting
  - 'process expression' examines the expression character at a time
    - If character is an operator, stack is popped, appropriate operation is called and result is pushed back onto stack
    - If character is not an operator then corresponding hex value is pushed onto stack
- When expression evaluation is complete, stack[0] is printed.
- BitTwiddle is invoked using PF1, PF2 and the following string: "f9>&3~8^|c~b| |"**
- (infix: "((f&>9)|(~3^8)|(~c|b))")
- You can write additional helper functions if you so