FIT1008 Introduction to Computer Science
Practical Session 7
Semester 2, 2014

Objectives of this practical session

To write a Python program that calculates integer arithmetic expressions involving the operators +, *, - and /, using Stacks and Reverse Polish Notation.

Testing

For this prac, you are required to write:

(1) a *function to test* each function you implement, and

(2) *at least two test cases* per function.

The cases need to show that your functions can handle both valid and invalid inputs.

Reverse Polish Notation

We will first assume that the arithmetic expression that we have to evaluate is written in Reverse Polish Notation (RPN). This notation was invented in the 1920's by Polish mathematician Jan Lukasiewicz as a different way of entering mathematical expressions by writing the operator after the operands instead of between them. It does not have the precedence problems that "infix" notation does and can therefore avoid the use of parenthesis.
The following table illustrates how normal "infix" expressions can be written in Reverse Polish notation:

Infix	Reverse Polish
a + b*c	a b c * +
a + (b*c)	a b c * +
(a + b)*c	a b + c *
a*b + c	a b * c +
a*(b + c)	a b c + *
(a + b)*(c - d)	a b + c d - *
(a + b)*(c - d)/(e + f)	a b + c d - * e f + /

Note that

- the variables appear in the same order
- the operators (+, *, -, /) come after the operands (variables) they are operating on

- no parentheses are needed

In this prac we will use a stack to evaluate the Reverse Polish integer expressions. The idea is as follows: as each integer is read from the screen, it is pushed onto the stack; when the characters +, -, * or / are read, two integers are popped off the stack, the operator is applied to them, and the result is pushed back onto the stack; when the end of the string is read this marks the end of the expression to be evaluated and the integer on top of stack is popped out and printed; if the stack contains anything else at this point, an error message is printed.
For example, the infix expression

$$(1 - 7) * (4 + 5)$$

should be entered via the screen as

```
1 7 - 4 5 + *
```

When 1 is read it is pushed onto the stack. Then 7 is read and also pushed onto the stack. Then, the character - is read, 7 and 1 are popped off, 7 is subtracted from 1, and the result -6 is pushed onto the stack. Next 4 is read and pushed onto the stack, and 5 is read and also pushed onto the stack. Then, the character + is read, 5 and 4 are popped off and their sum 9 is pushed onto the stack. Then, after the character * is read, 9 and -6 are popped off the stack, and their product -54 is pushed onto the stack. Finally, once the end of the string is read, -54 is popped off the stack and printed. Since the stack is empty, no error is given.

Task 1 [3 marks]

An easy way of implementing a Reverse Polish Notation evaluator is to use stacks.

(i) Write a Python class to implement a Stack ADT.

(ii) Write the following methods for your class:

`__len__()` which returns the number of elements stored in the stack.

`__str__()` which returnes a string composed of the integers stored in the stack separated by spaces and starting from the one at the top.

(iii) Write a Python program that implements a Stack, and allows a user to perform the following commands on the Stack using a menu:

Push: which should ask the user for an integer, check it is indeed an integer, if so push it onto the stack and, if not, give the appropriate error.

Pop: which should remove and print the integer at the top of the stack.

Print: which should print all the integers in the stack and doesn't change the stack.

Size: which should print the number of integers in the stack and doesn't change the stack.

Task 2 [3 marks]

Before you can write a program to evaluate expressions you need to be able to interpret the input.
Write a Python program that prompts the user for a character string, splits it into its substrings (using spaces as the delimiters), and prints out each substring together with a message indicating whether the substring is an integer, an operator (accepted operators will be addition, subtraction, multiplication, division, and equality, i.e., '+', '-', '*', '/'), or an invalid string.
For example, for the following input string:

```
12 3 + -8ha -98
```

the output would be:

```
  12 Integer
   3 Integer
   + Operator
-8ha Invalid string
 -98 Integer
```

Tip: You should divide your program into separate chunks of code. For instance, you could delegate the part of the code that takes a string and generates a description to its own separate `describe()` function. This also allows you to run tests on sub-functionality of your program.

Task 3 [4 marks]

Write a Python program that allows a user provide an integer expressions in Reverse Polish Notation. For each expression the user provides the program evaluates the expression and prints out the result.
Important: Make sure your program deals with invalid input, i.e., detects invalid input, gives the appropriate message and allows the program to continue. Invalid inputs includes strings that are too large for the stack, and those containing too many operands/operators.

Advanced Question [Bonus 2 marks]

So far our simple calculator reads expressions in Reverse Polish notation. In this question you will write a function that allows you to read integer infix expressions and prints out Reverse Polish expressions.
Dijkstra's shunting algorithm is a method of generating Reverse Polish notation from normal "infix" notation. We say that infix multiplication and division have a higher precedence than addition and subtraction. The former bind more tightly to their operands than the latter. The shunting algorithm uses a stack of characters and works as follows. When an INTEGER is read in, it is immediately printed. When an OPERATOR is read in, (let's call it the current Operator), then either:

- the stack is empty and current Operator is pushed onto the stack,
- the current Operator has higher precedence than the OPERATOR on top of the stack, and the current Operator is pushed onto the stack, or
- the current Operator has lower, or equal, precedence than the OPERATOR on top of the stack, and in this case those OPERATORS on the stack with greater, or equal, precedence than the current Operator are popped off the stack (and printed) until either the stack is empty, or the OPERATOR on top of the stack has lower precedence than the current Operator. Then the current Operator is pushed onto the stack.

Finally, when all the input is exhausted all the OPERATORS remaining on the stack are popped off and printed. For example, consider the following infix expression: 4*3+6/2 (which is equivalent to (4*3)+(6/2), evaluates to 15, and is written in Reversed Polish Notation as 4 3 * 6 2 / +). The shunting algorithm:

- reads a 4 and immediately prints it,
- reads a * and pushes it into the stack (since it is empty),
- reads a 3 and immediately prints it,
- reads a + and since + has lower precedence than the top of stack *, it pops *, prints it, and then pushes +,
- reads a 6 and prints it,
- reads a / and since it has higher precedence than the top of stack +, it pushes / into the stack,
- reads a 2 and immediately prints it
- since the input is exhausted, it pops / and prints it, and then pops + and prints it

Write a Python program that reads integer infix expressions and prints out Reverse Polish expressions to the screen.

Hall of Fame

In this question you will modify your calculator so that it can read normal "infix" notation. You will need to deal with parentheses. Parentheses isolate subexpressions. Dijkstra's shunting algorithm treats the subexpression as a piece of sub-input. When an opening '(' is encountered, it is pushed onto the stack and hides everything beneath. The next OPERATOR which is encountered is pushed onto the stack. When a closing ')' is encountered, it is treated as the end of a subexpression and any operators of the subexpression are popped off the stack (and printed) - up to the opening '('. Both parentheses are then discarded. The algorithm then continues as before.
Write a Python program that accepts infix notation with parentheses and computes the expression.