MINIA UNIVERSITY
FACULTY SCIENCE
Department of Computer Science
Data Structures Using Python

Exercises #5
ArrayStack Class Application

<u>Part A:</u>

Using the *ArrayStack Class*, write a function that implements the following algorithm, which accepts an infix expression, and converts it to postfix (Reverse Polish Notation (RPN)) form, then returns it.
For example, if the function accepts the infix expression:

(6 - (2 + 3)) ∗ (3 + 8 / 2) + 2

then the function should return its postfix form: 6 2 3 + - 3 8 2 / + ∗ 2 +

Note that this algorithm uses a character stack.

```
Algorithm Infix_To_Posfix
Begin
   Create an object S of ArrayStack;
   While items remain in infix expression Do
   Begin
        Get next item;
        If item is an operand then
                Add operand to postfix expression;
        Else if item is a left parenthesis then
                Push left parenthesis onto S;
        Else if item is a right parenthesis then
                Repeat
                        Pop operator from S;
                        If operator is not a left parenthesis then
                                Add operator to postfix expression;
                Until operator is a left parenthesis;
        Else    // item is an operator
                While not Empty (S) and top of S has priority ≥ operator Do
                Begin
                        Pop operator from S;
                        Add popped operator to postfix expression;
                End While;
                Push operator;
        End If
   End While
   While not Empty (S) Do
   Begin
        Pop operator from S;
        Add popped operator to postfix expression;
   End While
   Return postfix expression
End.
```

P.T.O.

Part B:
Using the *ArrayStack Class*, write a function that implements the following algorithm, which accepts an RPN (postfix) expression, and evaluates it, then returns the result.
For example, if the function accepts the postfix expression:

6 2 3 + - 3 8 2 / + ∗ 2 +

then the function should return its value: 9
Note that this algorithm uses an integer stack.

```
Algorithm Evaluate_Postfix
Begin
   Create an object S of ArrayStack;
   While items remain in postfix expression Do
   Begin
        Get next item;
        If item is an operand then
                Push item onto S;
        Else    // item is an operator
                Pop operand into Op2;
                Pop operand into Op1;
                Case operator type Of
                        '+' : Result = Op1 + Op2;
                        '-' : Result = Op1 – Op2;
                        '∗' : Result = Op1 ∗ Op2;
                        '/' : Result = Op1 / Op2;
                        '%' : Result = Op1 % Op2;
                End Case;
                Push Result onto S;
        End If
   End While
   Pop Result from S;
   Return Result;
End.
```

Part C:
Write a main program that:

- Reads an arithmetic expression in infix form.
- Calls Function Infix_To_Posfix() to convert the input infix expression to postfix form.
- Calls Function Evaluate_Posfix() to evaluate the postfix expression returned from Function Infix_To_Posfix().
- Displays the input infix expression, the postfix expression returned from Function Infix_To_Posfix(), and the value returned from Function Evaluate_Posfix().