bigexp2 = input("Question 2: Insert an expression to be solved, no spaces ")  
bigexp3 = input("Question 3: Insert an expression to be solved, no spaces ")  
  
  
########Question one stack class #########  
class StackQ1:  
 # create variables for use and manipulation  
 def \_\_init\_\_(self):  
 self.stackL = []  
 # variable to keep track of current list index size, for pop  
 self.s = -1;  
  
 # push function. adds the number to the end  
 def push(self, num):  
 self.stackL.append(num)  
 self.s = self.s + 1  
  
 # pop function. Removes and shows the last inserted variable  
 def pop(self):  
 if (self.s > -1):  
 print(self.stackL[self.s])  
 self.stackL.\_\_delitem\_\_(self.s)  
 self.s = self.s - 1  
 else:  
 # no numbers to print  
 print('No Stack Size')  
  
 # returns top number/last inserted item but doesn't remove it  
 def top(self):  
 if (self.s > -1):  
 print(self.stackL[self.s])  
 else:  
 print('No Stack Size')  
  
 # returns the minimum valued item in the list of numbers  
 def getMin(self):  
 if (self.s == -1):  
 print('No Minumum, because no Stack size')  
 else:  
 min = self.stackL[0]  
 for i in range(len(self.stackL)):  
 if self.stackL[i] < min:  
 min = self.stackL[i]  
 print(min)  
  
  
####Question three and two stack class #######  
class StackQ3:  
 # create variables for use and manipulation  
 def \_\_init\_\_(self):  
 self.stackL = []  
 # variable to keep track of current list index size, for pop  
 self.s = -1;  
  
 # push function. adds the number to the end  
 def push(self, num):  
 self.stackL.append(num)  
 self.s = self.s + 1  
  
 # pop function. Removes and shows the last inserted variable  
 def pop(self):  
 if (self.s > -1):  
 value = (self.stackL[self.s])  
 self.stackL.\_\_delitem\_\_(self.s)  
 self.s = self.s - 1  
 return value  
  
 # returns top number/last inserted item but doesn't remove it  
 def top(self):  
 if (self.s > -1):  
 return (self.stackL[self.s])  
  
  
class Main:  
 ### TESTING FOR QUESTION 1  
 print()  
  
 MinStack = StackQ1()  
 MinStack.push(7)  
 MinStack.push(3)  
 MinStack.pop()  
 MinStack.pop()  
 MinStack.pop()  
 MinStack.pop()  
 MinStack.getMin()  
 MinStack.push(3)  
 MinStack.push(-3)  
 MinStack.push(5)  
 MinStack.push(-10)  
 MinStack.push(0)  
 MinStack.top()  
 MinStack.getMin()  
  
 print()  
 ###QUESTION 2 ##################################################################################################  
 # two stacks, one for numbers and another for operators  
 nums2 = StackQ3()  
 ops = StackQ3()  
 exp2 = bigexp2  
  
 # function to see if the position is a number  
 def isNumber(self, pos):  
 if ((pos == '+') | (pos == '-') | (pos == '\*') | (pos == '/')):  
 return False  
 return True  
  
 # function for doing an operation  
 def doOp(self):  
 num1 = self.nums2.pop()  
 num2 = self.nums2.pop()  
 op = self.ops.pop()  
 numR = 0  
 if op == '+':  
 numR = num2 + num1  
 if op == '-':  
 numR = num2 - num1  
 if op == '\*':  
 numR = num2 \* num1  
 if op == '/':  
 numR = num2 / num1  
 self.nums2.push(numR)  
  
 #function to decide the priority of the operator, higher = more pressing/priority  
 def prec(self, op):  
 if (op == '+') | (op == '-'):  
 return 2  
 if (op == '\*') | (op == '/'):  
 return 3  
 if (op == '>') | (op == '<'):  
 return 1  
 if op == '$':  
 return 0  
 return 0  
  
 # function to do the main part of the program, dividing and completing parts of the expression  
 def repeatOps(self, refOp):  
 # while there is > 1 number in the number stack, and the priority of the current operation is lower or equal to the last known operator:  
 while ((len(self.nums2.stackL) > 1) & (len(self.ops.stackL) > 0)):  
 self.doOp()  
  
 def solve(self):  
 for i in range(len(self.exp2)):  
 if self.isNumber(self.exp2[i]):  
 self.nums2.push(int(self.exp2[i]))  
 else:  
 self.repeatOps(self.exp2[i])  
 self.ops.push(self.exp2[i])  
 self.repeatOps('$')  
 print("The value of the result of the expression is for Question 2 is: ", self.nums2.pop())  
  
  
 ###QUESTION 3 #########################################################################################  
  
 # get postfix notation from user and create a stack for this question  
 nums3 = StackQ3()  
 # expIn=input("Enter an expression in postfix notation, no spaces")  
 exp3 = bigexp3  
 # for the index range of the given expression...  
  
 for i in range(len(exp3)):  
 # if the symbol at i is an operation, pop the last two numbers and do the operation on them  
 if ((exp3[i] == '+') | (exp3[i] == '-') | (exp3[i] == '\*') | (exp3[i] == '/')):  
 num1 = nums3.pop()  
 num2 = nums3.pop()  
 op = exp3[i]  
 # seperate variable numR for testing and debugging the result  
 numR = 0;  
 # if statements for the operation  
 if op == '+':  
 numR = num2 + num1  
 if op == '-':  
 numR = num2 - num1  
 if op == '\*':  
 numR = num2 \* num1  
 if op == '/':  
 numR = num2 / num1  
 nums3.push(numR)  
 # if the character at i isn't an operation, it should be a number  
 else:  
 # put the number at the top of the stack, after changing it into an integer  
 nums3.push(int(exp3[i]))  
 # print the last number, also the only number, in the stack. This will be the result of the postfix notation  
 print("The Result of the Expression is for Number 3 is: ", nums3.top())  
  
  
print()  
#print("############################### Question 3 ###############################")  
print()  
main = Main()  
main.solve()