Stack

د. عامر جعفر صادق د. هاله حسن محمود م.م امنه هيثم عبد اللطيف

Q1) stacking, taking the top element of the stack, determining whether the stack is empty, and displaying the elements of the stack

Program

```
class Stack:
    def __init__(self):
        self.stack =[]
    def push(self,value): # push method
        self.stack.append(value)
        return True
    def pop(self): #stack out method
                # Determine if the stack is empty
        if self.stack:
                        # Get the stack element and return
            item = self.stack.pop()
            return item
        else:
            return False
    def top(self): #View stack top element method
        if self.stack:
            return self.stack[-1]
        else:
            return False
    def length(self): #View stack length method
        return len(self.stack)

    def view(self): #View stack element method
        return ','.join(self.stack)
s = Stack()
s.push('1')
s.push('2')
s.push('3') #Push three elements 1, 2, 3 on the stack
item = s.pop() # pop an element from the stack
print(s.top()) #View the top element of the stack
print(s.length()) #View the stack length
print(s.view()) #View stack elements
```

Output

```
2
2
1,2
```

Q2) Find maximum in stack in O(1) without using additional stack

Program

```
class Block:

    # A block has two elements
    # as components  (i.e. value and localMax)
    def __init__(self, value, localMax):
        self.value = value
        self.localMax = localMax

class Stack:
    def __init__(self, size):

        # Setting size of stack and
        # initial value of top
        self.stack = [None] * size
        self.size = size
        self.top = -1

    # Function to push an element
    # to the stack
    def push(self, value):

        # Don't allow pushing elements
        # if stack is full
        if self.top == self.size - 1:
            print("Stack is full")
        else:
            self.top += 1

            # If the inserted element is the first element
            # then it is the maximum element, since no other
            # elements is in the stack, so the localMax
            # of the first element is the element itself
```

```
        if self.top == 0:
            self.stack[self.top] = Block(value, value)

        else:

            # If the newly pushed element is less
            # than the localMax of element below it,
            # Then the over all maximum doesn't change
            # and hence, the localMax of the newly inserted
            # element is same as element below it
            if self.stack[self.top - 1].localMax > value:
                self.stack[self.top] = Block(
                    value, self.stack[self.top - 1].localMax)

            # Newly inserted element is greater than
            # the localMax below it, hence the localMax
            # of new element is the element itself
            else:
                self.stack
                self.stack[self.top] = Block(value, value)

        print(value, "inserted in the stack")

# Function to remove an element
# from the top of the stack
def pop(self):

    # If stack is empty
    if self.top == -1:
        print("Stack is empty")

    # Remove the element if the stack
    # is not empty
    else:
        self.top -= 1
        print("Element popped")
```

```
    # Function to find the maximum
    # element from the stack
    def max(self):

        # If stack is empty
        if self.top == -1:
            print("Stack is empty")
        else:

            # The overall maximum is the local maximum
            # of the top element
            print("Maximum value in the stack:",
                  self.stack[self.top].localMax)

# Driver code

# Create stack of size 5
stack = Stack(5)
stack.push(2)
stack.max()
stack.push(6)
stack.max()
stack.pop()
stack.max()
```

Output

```
2 inserted in stack
Maximum value in the stack: 2
6 inserted in stack
Maximum value in the stack: 6
Element popped
Maximum value in the stack: 2
```

Q3) Reverse a string using stack

Program

```
# Python program to reverse a string using stack
 # Function to create an empty stack.
# It initializes size of stack as 0
def createStack():
    stack=[]
    return stack

# Function to determine the size of the stack
def size(stack):
    return len(stack)

# Stack is empty if the size is 0
def isEmpty(stack):
    if size(stack) == 0:
        return true

# Function to add an item to stack .
# It increases size by 1
def push(stack,item):
    stack.append(item)

#Function to remove an item from stack.
# It decreases size by 1
def pop(stack):
    if isEmpty(stack): return
    return stack.pop()

# A stack based function to reverse a string
def reverse(string):
    n = len(string)

    # Create a empty stack
    stack = createStack()
```

```
    # Push all characters of string to stack
    for i in range(0,n,1):
        push(stack,string[i])

    # Making the string empty since all
    #characters are saved in stack
    string=""

    # Pop all characters of string and
    # put them back to string
    for i in range(0,n,1):
        string+=pop(stack)

    return string

# Driver program to test above functions
string="Data Structure Lab 2023-2024"
string = reverse(string)
print("Reversed string is " + string)
```

Output

Reversed string is 4202-3202 baL erutcurtS ataD