**Department of Information Technology**

**UIT2201 Programming and Data Structures**

**2022 – 2023**

**Exercise — 09**

**Part A**

**1.Design and implement stack and queue data structure as wrapper around the Python List. Using stack and queue check whether a given number is a palindrome.**

**Code:**

class Stack:

    '''

    This class is used for creating the Simple Stack

    using the Wrapper class

    '''

    def \_\_init\_\_(self):

        '''This function is called constructor used for initializing'''

        self.list=[]

    def isempty(self):

        '''This function is to check wheather stack is empty'''

        return len(self.list)==0

    def insert(self,ele):

        '''This function is used for inserting the element'''

        return self.list.append(ele)

    def delete(self):

        '''This function is used for deleting the element'''

        return self.list.pop()

    def \_\_len\_\_(self):

        '''This function gives us the length of Stack'''

        return len(self.list)

    def str(self):

        '''This function displays the Stack'''

        return str(self.list)

class Queue:

    '''

    This class is used for creating the Simple Queue

    using the Wrapper class

    '''

    def \_\_init\_\_(self):

        '''This function is called constructor used for initializing'''

        self.list=[]

        self.front=0

        self.rear=0

    def isempty(self):

        '''This function is to check wheather Queue is empty'''

        return self.front==self.rear

    def enqueue(self,ele):

        '''This function is used for inserting the element'''

        return self.list.append(ele)

    def dequeue(self):

        '''This function is used for deleting the element'''

        return self.list.pop(0)

    def \_\_len\_\_(self):

        '''This function gives us the length of Queue'''

        return self.rear-self.front

    def str(self):

        '''This function displays the Queue'''

        return str(self.list)

def palindrome():

    '''This function tells the number is palindrome or not'''

    num=int(input("Enter Number: "))        #getting the number from user

    stack=Stack()                           #Creating instance for Stack

    queue=Queue()                           #Creating instance for Queue

    string=str(num)

    for i in string:

        stack.insert(i)                     #inserting element in the stack

        queue.enqueue(i)                    #inserting element in the queue

    result=0

    result=1

    for i in range(len(stack)):             #iterating the element

        if stack.delete()==queue.dequeue(): #checking the palindrome condition

            result=0

        else:

            result=1

    if result==0:

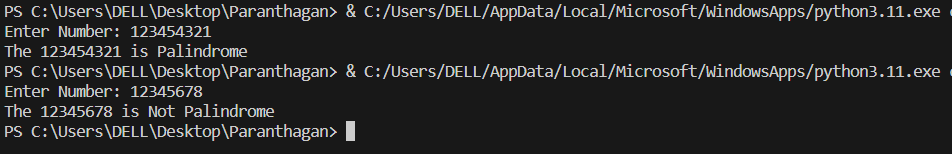
        print(f"The {num} is Palindrome")

    else:

        print(f"The {num} is Not Palindrome")

palindrome()                                #calling the function

**Output:**

****

**2.Design and implement data structure to maintain two stacks in a single array. All the basic stack operations should include an argument to select one of the stacks. For example, ‘push(0, item)’ should push item into the first stack, while ‘push(1, item)’ should push item into the second stack. Your stack methods should not raise stack full exception, unless every array cell is used .**

**Code:**

import ctypes       #Importing the ctypes module

class Exception:    #Creating class for Exception

    pass

class TwoStack:

    '''

    This class is used for creating the Double Stack

    using the Compact Array Method

    '''

    def \_\_init\_\_(self,capacity):

        '''This function is called constructor used for initializing'''

        self.capacity=capacity

        self.top1=0

        self.top2=capacity-1

        self.list=self.makearray(self.capacity)

    def makearray(self,capacity):

        '''This function is used for making new array'''

        return (capacity\*ctypes.py\_object)()

    def push(self,stack\_no,element):

        '''This function is used for inserting the element'''

        if self.isfull():

            raise Exception("Stack is Full")

        else:

            if stack\_no == 0:

                self.list[self.top1]=element

                self.top1+=1

            elif stack\_no == 1:

                self.list[self.top2]=element

                self.top2-=1

    def pop(self,stack\_no):

        '''This function is used for deleting the element'''

        if self.isempty():

            raise Exception("Stack is Empty")

        if stack\_no == 0:

            if self.top1>0:

                del\_item=self.list[self.top1-1]

                self.list[self.top1-1]=' '

                self.top1-=1

                return del\_item

            else:

                raise Exception("Index out of Range")

        elif stack\_no == 1:

            if self.top2<self.capacity-1:

                del\_item=self.list[self.top2+1]

                self.list[self.top2+1]=' '

                self.top2+=1

                return del\_item

            else:

                raise Exception("Index out of Empty")

    def \_\_getitem\_\_(self,index):

        '''This function is used for getting the element'''

        return self.list[index]

    def \_\_setitem\_\_(self,index,element):

        '''This function is used for assigning the element'''

        self.list[index]=element

    def isfull(self):

        '''This function checks wheather stack is full'''

        return self.top1>self.top2

    def isempty(self):

        '''This function checks wheather stack is empty'''

        return (self.top2 - self.top1 + 1) == self.capacity

    def \_\_str\_\_(self):

        '''This function dispalys the stack elements'''

        lst=[]

        for element in self:

            if element == None:

                lst.append(" ")

            else:

                lst.append(element)

        return "A "+str(lst)+" B"

def fun():

    '''This function is used for appending and

       deleting the elements from the stack'''

    ts=TwoStack(8)                          #Creating the instance for TwoStack

    #appending the elements

    ts.push(0,1)

    ts.push(0,2)

    ts.push(0,3)

    ts.push(0,4)

    ts.push(1,5)

    ts.push(1,6)

    ts.push(1,7)

    ts.push(1,8)

    print("\nThe Double Stack is Full")

    print(ts)

    #deleting the elements

    print("\nAn Element is popped from Stack A")

    ts.pop(0)

    print(ts)

    #appending the element

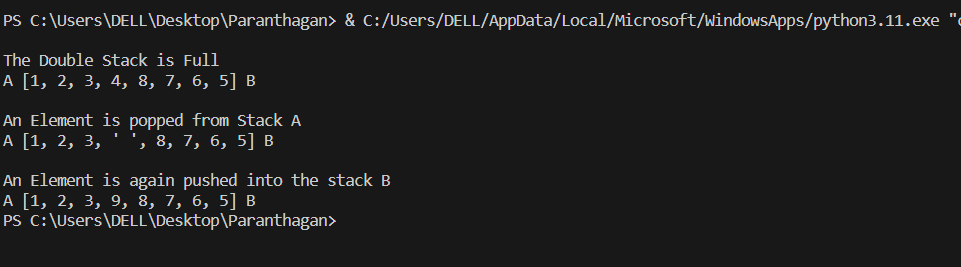
    print("\nAn Element is again pushed into the stack B")

    ts.push(1,9)

    print(ts)

fun()                   #Calling the Function

**Output:**

****