WEEK 7

**PROGRAM 1: IMPLEMENT STACK DATA STRUCTURE.**

**1. Push**

**2. Pop**

size=10

s=[0]\*(size+1)

top=0

def is\_empty():

if top==0:

return True

return False

def push(x):

global top

top=top+1

if top>size:

print("Stack over flow")

else:

s[top]=x

def pop():

global top

if is\_empty():

print("Stack under flow")

else:

top=top-1

return s[top+1]

if \_\_name\_\_=='\_\_main\_\_':

pop()

push(10)

push(20)

push(30)

push(40)

push(50)

push(60)

push(70)

push(80)

push(90)

push(100)

push(110)

print(s[1:size+1])

**OUTPUT:**

****

**ALGORITHM:**

**STEP 1:** Declare and initialize the size of a stack

**STEP 2:** Define a function isempty() for checking whether the stack is empty or not

**STEP 3:** Define a function Push() for inserting elements into the stack

**STEP 4:** Define a function Pop() for deleting elements from the stack

**STEP 5:** Call the functions push() , pop() with values

**PROGRAM 2:**

**IMPLEMENT BRACKET MATCHING USING STACK**

def areBracketsBalanced(expr):

stack= []

for char in expr:

if char in ["(","{","["]:

stack.append(char)

else:

if not stack:

return False

current\_char = stack.pop()

if current\_char == '(':

if char !=")":

return False

if current\_char == '{':

if char !="}":

return False

if current\_char == '[':

if char !="]":

return False

if stack:

return False

return True

if \_\_name\_\_ == "\_\_main\_\_":

expr="{()} []"

if areBracketsBalanced(expr):

print("Balanced")

else:

print("Not Balanced")

**OUTPUT**

****

**ALGORITHM:**

**STEP 1:** Define a function areBracketBalanced()

**STEP 2**: Declare stack S

**STEP 3**: Now traverse the string expression using a pointer.if the current pointer is at the opening bracket ('(' or '{' or '[') then push it to stack S.else the current pointer is at closing bracket (')' or '}' or ']') then pop from the stack.

if the popped bracket is the matching opening bracket then brackets are valid else brackets are not valid.