****

***Data Structure and Algorithm (Lab)***

***Assignment - 1***

**Name:**

Ali Maqsood.

**Roll no:**

SU92-BSAIM-F23-050.

**Department:**

Software Engineering Department.

**Program:**

Artificial Intelligence.

**Section:**

BSAI-3A

**Question # 1:**

**Objective**: Write a program to evaluate a postfix (Reverse Polish Notation) mathematical expression using a stack.

**Steps**:

1. Read a postfix expression as input (e.g., "23+5\*", which means (2 + 3) \* 5).
2. Use a stack to evaluate the expression:
   * Push numbers onto the stack.
   * When encountering an operator, pop two numbers, apply the operator, and push the result back onto the stack.
3. At the end, the stack should contain a single number, which is the result.

**Code:**

class task1:

    def \_\_init\_\_(self):

        self.stack=[]

    def process(self, statement):

        for i in statement:

            if i.isdigit():

                self.stack.append(int(i))

            else:

                a1=self.stack.pop()

                a2=self.stack.pop()

                if i=="+":

                    self.stack.append(a1+a2)

                elif i=="-":

                    self.stack.append(a1-a2)

                elif i=="\*":

                    self.stack.append(a1\*a2)

                elif i=="/":

                    self.stack.append(a1/a2)

                else:

                    print("Invalid operator. Ending....")

                    break

        return self.stack.pop()

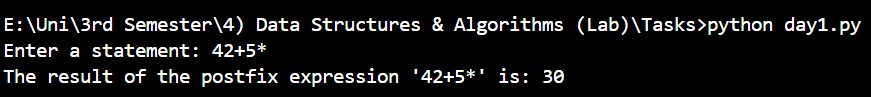
statement = input("Enter a statement: ")

obj1 = task1()

ans = obj1.process(statement)

print(f"The result of the postfix expression \'{statement}\' is: {ans}")

**Output:**



**Question # 2:**

**Objective**: Simulate a simple browser navigation system using two stacks.

**Steps**:

1. Create two stacks: back\_stack and forward\_stack.
2. Implement the following operations:

* visit(url) - Navigate to a new URL and clear the forward\_stack.
* back() - Move to the previous URL (pop from back\_stack and push to forward\_stack).
* forward() - Move to the next URL (pop from forward\_stack and push to back\_stack).

1. Print the current URL after each operation.

**Code:**

class task2:

    def \_\_init\_\_(self):

        self.back\_stack=[]

        self.forward\_stack=[]

        self.current\_url=None

    def current(self):

        self.current\_url=input("Enter the current url: ")

        self.back\_stack.append(self.current\_url)

        self.forward\_stack.clear()

    def previous(self):

        if self.back\_stack:

            self.forward\_stack.append(self.current\_url)

            self.current\_url=self.back\_stack.pop()

        else:

            print("Back Stack is empty....")

    def forward(self):

        if self.forward\_stack:

            self.back\_stack.append(self.current\_url)

            self.current\_url=self.forward\_stack.pop()

        else:

            print("Front Stack is empty....")

    def display(self):

        print(f"Current URL: {self.current\_url}")

        print(f"Back URL: {self.back\_stack.pop()}")

        print(f"Front URL: {self.forward\_stack.pop()}")

obj1=task2()

obj1.current()

obj1.current()

obj1.current()

obj1.previous()

obj1.forward()

obj1.display()

**Output:**



**Question # 3:**

**Objective**: Write a program to check if a string is a palindrome using a stack.

**Steps**:

1. Push all characters of the string onto a stack.
2. Pop characters from the stack to create a reversed string.
3. Compare the reversed string with the original string to determine if it is a palindrome.

**Code:**

class task3:

    def \_\_init\_\_(self):

        self.stack = []

        self.input1=input("Input: ").lower()

    def push(self):

        for i in self.input1:

            self.stack.append(i)

    def check(self):

        reversed\_word=""

        while self.stack:

            reversed\_word+=self.stack.pop()

        if self.input1==reversed\_word:

            print("Output: True.")

        else:

            print("Output: False.")

obj1=task3()

obj1.push()

obj1.check()

**Output:**

