**Assignment 16**

**Algorithm for push():**

1. START

2. IF TOP = NONE THEN # stack is empty

2.1 ALLOCATE MEMORY FOR ARR[0]

2.2 TOP = 0

2.3 ARR[TOP] = ELEM # add the element

2.4 GO TO 6

3. ALLOCATE MEMORY FOR ARR[TOP+1]

4. ARR[TOP + 1] = ELEM # add the element

5. TOP = TOP +1 # increase the value of top

6. END

**Algorithm for pop():**

1. START

2. IF TOP = NONE THEN # when stack is empty

2.1 DISPLAY “EMPTY STACK”

2.2 GO TO 4

3. IF TOP =0 THEN #one element is left

3.1 DISPLAY ARR[0]

          3.2 DEALLOCATE ARR[0]

         3.3 TOP = NONE #set value to none to imply that stack is empty

    ELSE # valid elements are there

3.4 DISPLAY ARR[TOP]

             3.5 DEALLOCATE ARR[TOP]

             3.6 TOP = TOP - 1

4. END

1. Write a program that implements the push and pop function of the stack using lists. Send the list and top as parameters to these functions. Push should add the element into the list and change top value. Pop should return the top value and delete the element from the list. Display the deleted element. Boundary conditions should also be checked.
2. Import the file from Q1 into another program. Send a stack of
3. Empty list of integers for popping
4. Push 2 elements into the stack
5. Pop 1 element from the stack
6. Import the file from Q1 into another program, send a string in the form of a stack. Print the reverse of the string using the stack functions.
7. Import the file from Q1 into another program, convert a decimal number into its binary equivalent using the stack functions.
8. Consider a data structure that can hold data such as Name and Salary of an employee. Write a menu driven program to implement a **Stack** of employees. Write relevant functions Hire () and Fire () which pushes and pops employees in the stack. Take care of the boundary conditions.

* Hire Employee
* Fire Employee
* Exit

1. Write a menu driven program to accept and store a list of names who come to the canteen for meals. Perform the following operations by implementing a **stack**. Take care of the boundary conditions.

1. Issue a token (ask for name, generate a token number, and add to the stack [<token>, <name>])
2. Issue meal (accept ‘n’ remove ‘n’ students from the stack. Display their names)
3. Exit

Sample list: [ [1,”Josh”],[ 2,”Rohan”]]