UNIT - II

STACKS AND QUEUES

STACK ADT

Introduction

- ***A stack is a linear data structure in which the elements are added and removed only from one end, which is called the top.***
- ***A stack is called a LIFO (Last-In, First-Out) data structure***
- Hence the element that is inserted last is the first one to be taken out.
- Stacks can be implemented either using an array or a linked list.

Linear Data Structure

- ***In the linear data structure, the data is organized in a linear order in which elements are linked one after the other.***
- first element has only one next element
- last element has only one previous element
- all other elements have a next and a previous element

Example : 11, 22,33, 44, 55

Add and Remove from one end

- In any data structure **Addition** i.e., Insertion and **Remove** i.e., Deletion are two very essential operations to be perform

- **Insertion :** Insert the element either First, Last or Middle of the list

- **Deletion :** Delete the element either First, Last or Middle of the list

LIFO

- ***A stack is called a LIFO (Last-In, First-Out) data structure***
- ***The element that is inserted last is the first one to be taken out***

Example :

- Place set of books in a box
- Wear set of Bangles one by one and remove

IMPLEMENTATION

A stack can be implemented by two methods

- ***Array Implementation of Stack***
- ***Linked Implementation of Stack***

  (also called Linked Stack)

Array Implementation of Stack

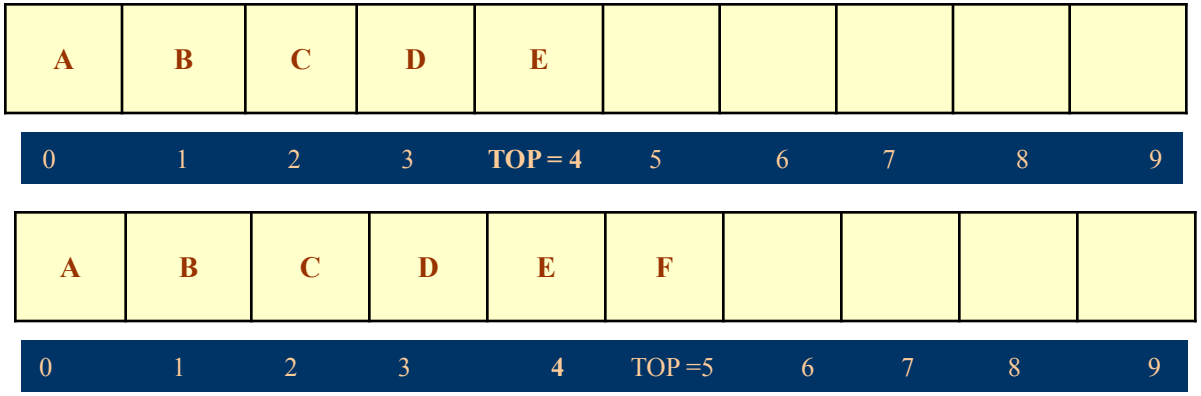
- In computer's memory stacks can be represented as a linear array.
- Every stack has a variable TOP associated with it.
- **TOP** is used to store the address of the topmost element of the stack. It is this position from where the element will be added or deleted.
- There is another variable **MAX** which will be used to store the maximum number of elements that the stack can hold.
- **TOP = -1 □ the stack is empty**
- **TOP = MAX -1 □ the stack is full.**

Stack Operations

- PUSH
- POP
- PEEK

Push Operation

- The push operation is used to insert an element in to the stack.
- The new element is added at the topmost position of the stack.
- However, before inserting the value, we must first check if TOP=MAX-1, because if this is the case then it means the stack is full and no more insertions can further be done.
- If an attempt is made to insert a value in a stack that is already full, an OVERFLOW message is printed.



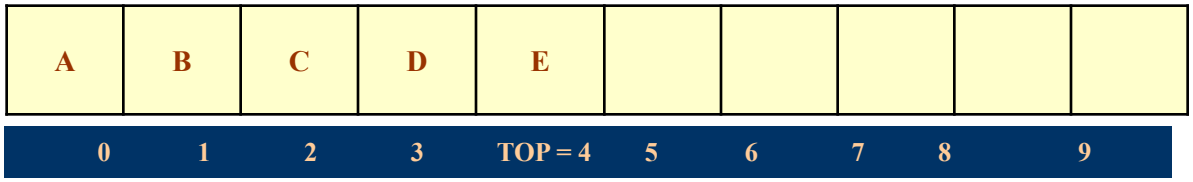
Algorithm for Push Operation

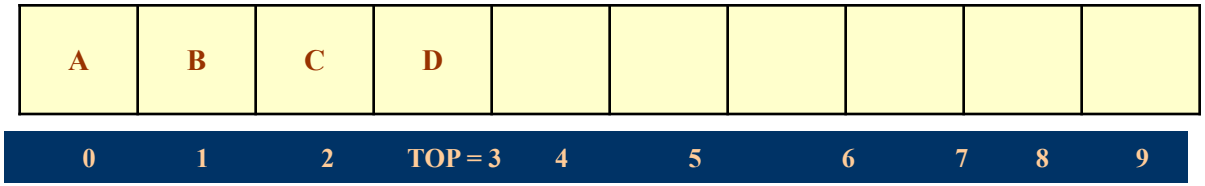
```
Algorithm to PUSH an element from a
stack

Step 1: IF TOP = MAX - 1, then
        PRINT "OVERFLOW"
        Go TO Step 4
          [END OF IF]
Step 2: SET TOP = TOP + 1
Step 3: SET STACK[TOP] = VAL
Step 4: END
```

Pop Operation

- The pop operation is used to delete the topmost element from the stack.
- However, before deleting the value, we must first check if TOP=-1, because if this is the case then it means the stack is empty so no more deletions can further be done.
- If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed.





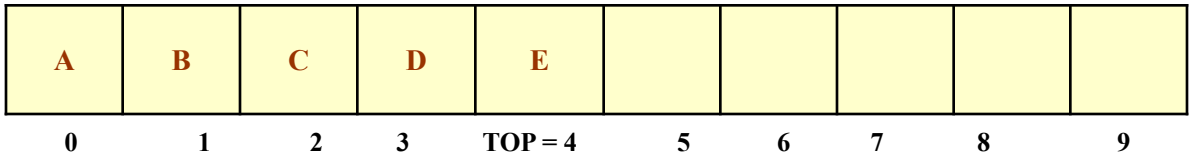
Algorithm for Pop Operation

```
Algorithm to POP an element from a stack

Step 1: IF TOP = -1, then
        PRINT "STACK IS EMPTY"
        Go TO Step 4
          [END OF IF]
Step 2: SET VAL = STACK[TOP]
Step 3: SET TOP = TOP - 1
Step 4: END
```

Peek Operation

- Peek is an operation that returns the value of the topmost element of the stack without deleting it from the stack.
- However, the peek operation first checks if the stack is empty or contains some elements.
- If TOP = -1, then an appropriate message is printed else the value is returned.



Here Peek operation will return E, as it is

the value of the topmost element of the stack.

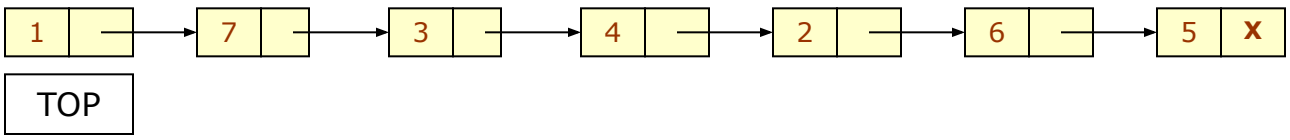
Algorithm for Peek Operation

```
Algorithm to PEEK an element from a stack

Step 1: IF TOP = -1, then
        PRINT "STACK IS EMPTY"
        Go TO Step 3
          [END OF IF]
Step 2: RETURN STACK[TOP]
Step 3: END
```

Linked Implementation of Stack (Linked Stack)

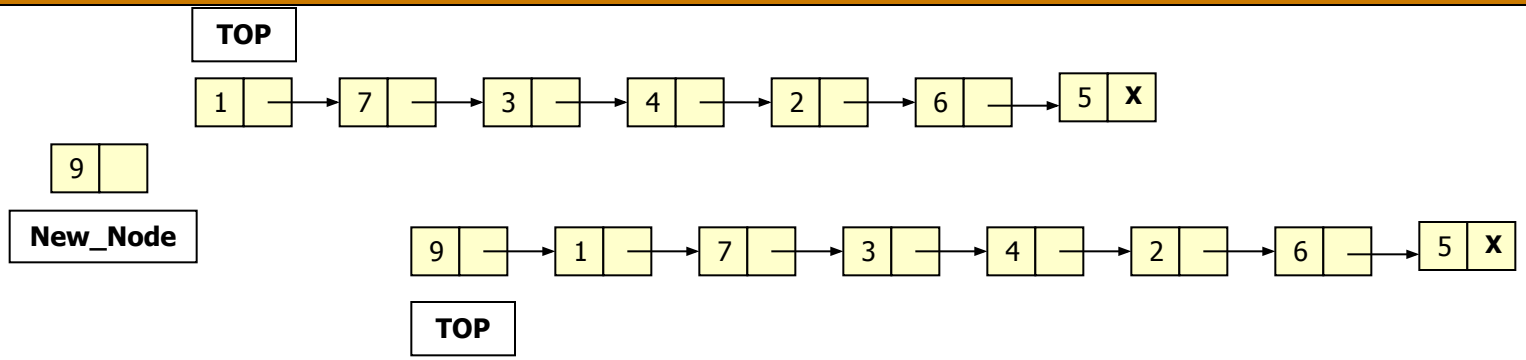
- In a linked stack, every node has two parts – one that stores data and another that stores the address of the next node.
- The START pointer of the linked list is used as **TOP**.
- **TOP = NULL** – The stack is empty.
- **PUSH** – Insert at beginning in Singly Linked List
- **POP** – Delete at beginning in Singly Linked List



Push Operation on a Linked Stack

Algorithm to PUSH an element in a linked stack

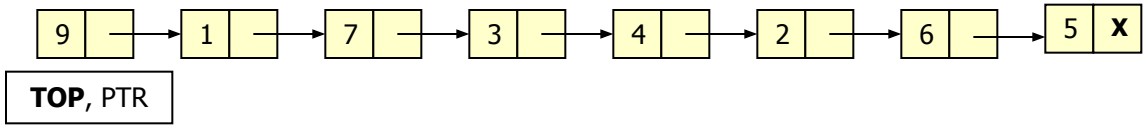
```
Step 1: IF AVAIL = NULL, then
          Write OVERFLOW
          Go to Step 5
        [END OF IF]
Step 2: SET New_Node = AVAIL
Step 3: SET New_Node -> DATA = VAL
Step 4: IF TOP = NULL, then
          SET New_Node -> NEXT = NULL
          SET TOP = New_Node
        ELSE
          SET New_node -> NEXT = TOP
          SET TOP = New_Node
        [END OF IF]
Step 5: END
```

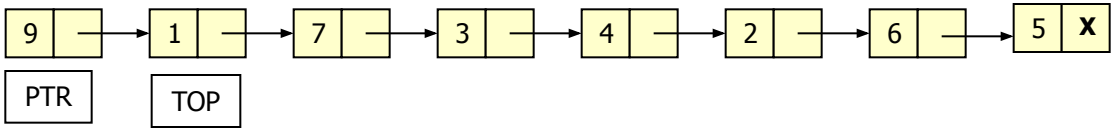


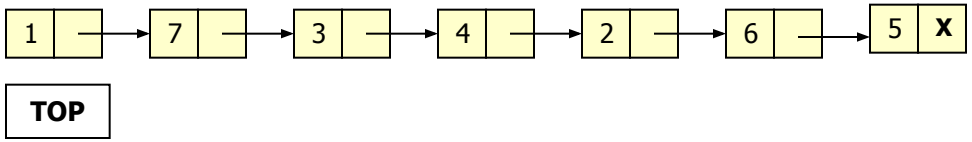
Pop Operation on a Linked Stack

Algorithm to POP an element from a stack

```
Step 1: IF TOP = NULL, then
          PRINT "UNDERFLOW"
                Goto Step 5
        [END OF IF]
Step 2: SET PTR = TOP
Step 3: SET TOP = TOP ->NEXT
Step 4: FREE PTR
Step 5: END
```



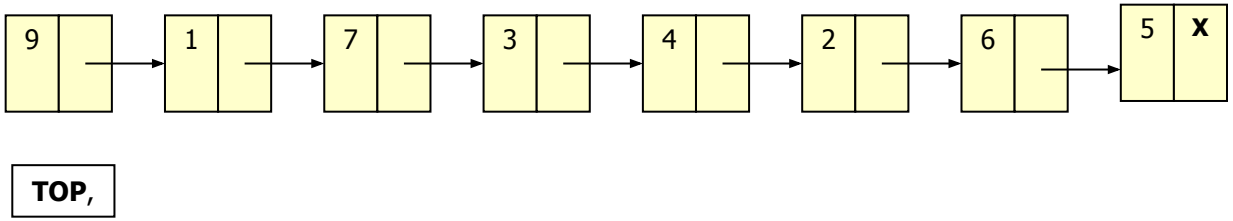




Peek Operation on a Linked Stack

Algorithm for PEEK operation in a stack

```
Step 1: IF TOP = NULL, then
          PRINT "UNDERFLOW"
                Goto Step 3
        [END OF IF]
Step 2: RETURN TOP -> DATA
Step 3: END
```



LINKED STACK – C PROGRAM

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *next;
};

struct node *top = NULL;

void push(int);
void display();
void pop();
void peek();
```

LINKED STACK – C PROGRAM

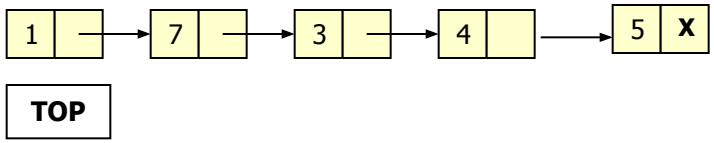
```
int main()
{
    int val, option;
    do
    {
        printf("\n *****MAIN MENU*****");
        printf("\n 1. PUSH");
        printf("\n 2. POP");
        printf("\n 3. DISPLAY");
        printf("\n 4. EXIT");
        printf("\n Enter your option : ");
        scanf("%d", &option);

        switch(option)
        {
            case 1:
                printf("\n Enter the number to be pushed on stack :");
                scanf("%d", &val);
                push(val);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);

        }
    }while(option<4);
    return 0;
}
```

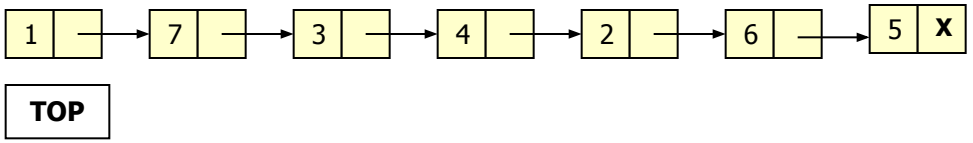
LINKED STACK – C PROGRAM

```
void push(int val)
{
    struct node *new_node;
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = val;
    if(top == NULL)
    {
        new_node -> next = NULL;
        top = new_node;
    }
    else
    {
        new_node -> next = top;
        top = new_node;
    }
}
```

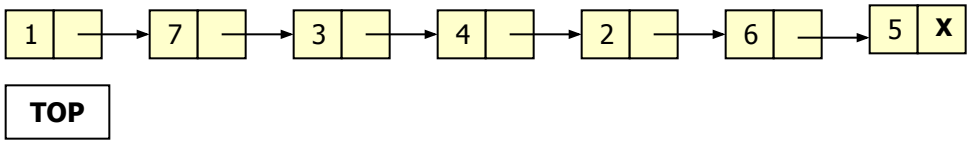


LINKED STACK – C PROGRAM

```
void pop()
{
    struct node *ptr;
    ptr = top;
    if(top == NULL)
        printf("\n STACK UNDERFLOW");
    else
    {
        top = top -> next;
        printf("\n The value being deleted is : %d", ptr -> data);
        free(ptr);
    }
}
```



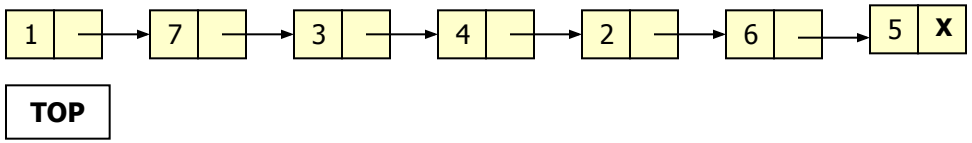
LINKED STACK – C PROGRAM




```
void peek()
{
    if(top == NULL)
        printf("\n STACK UNDERFLOW");
    else
    {
        printf("\n The topmost element of the stack is %d", top -> data);
    }
}
```

LINKED STACK – C PROGRAM

```
void display()
{
    struct node *ptr;
    ptr = top;
    if(top == NULL)
        printf("\n STACK IS EMPTY");
    else
    {
        while(ptr != NULL)
        {
            printf("\n%d", ptr -> data);
            ptr = ptr -> next;
        }
    }
}
```



Output

1
7
3
4
2
6
5