STACK

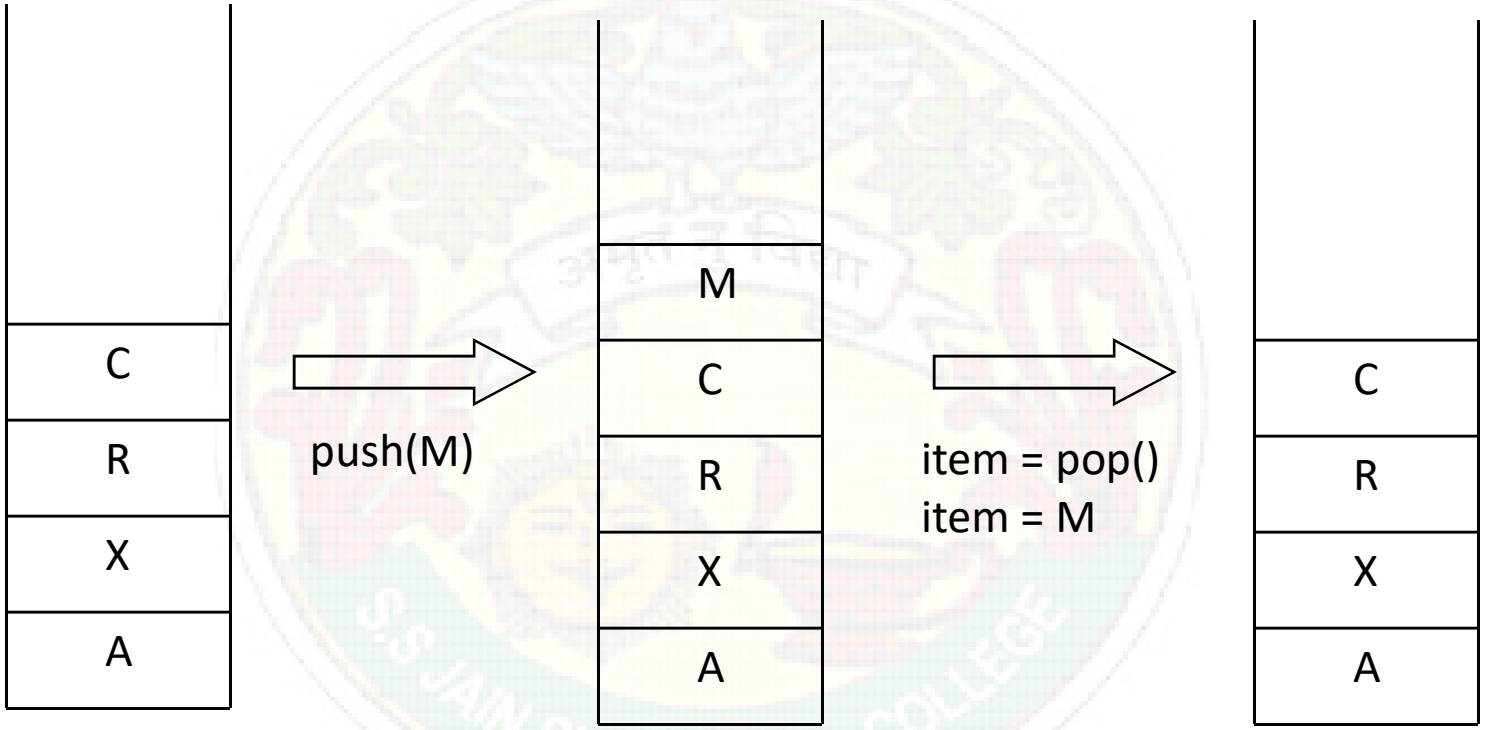
STACK

- Stack is a collection of elements , where element last to be inserted is first to be taken out.
- Stack implement principle of FILO or LIFO.
- Only access to the stack is the top element
  - consider trays in a cafeteria
    - to get the bottom tray out, you must first remove all of the elements above

Operations on Stack

- *Push*
  - the operation to place a new item at the top of the stack
- *Pop*
  - the operation to remove item from the top of the stack
- Peep
  - the operation to display or read item from the top of the stack

Stack



Implementing a Stack

- Stack can be implemented using
  - Array
  - linked list
- Which method to use depends on the application
  - what advantages and disadvantages does each implementation have?

Implementing Stacks: Array

- Advantages
  - best performance
- Disadvantage
  - fixed size
- Basic implementation
  - initially empty array
  - field to record where the next data gets placed into
  - if array is full, push() returns false
    - otherwise adds it into the correct spot
  - if array is empty, pop() returns null
    - otherwise removes the next item in the stack

Implementing Stacks: Array

```
struct stack
{
    int a[10];
    int top;
}s;
s.top = -1;
```

push()

```
void push(int item)
 {
    if(s.top == size-1)
          printf("Overflow");
    else
    {     s.top++;
          s.a[s.top] = item;
    }
}
```

pop()

```
int pop()
 {
    int ele;
    if(s.top == -1)
          printf("Underflow");
    else
    {     ele = s.a[s.top];
          s.top--;
    }
    return ele;
}
```

peep()

```
int peep()
 {
    if(s.top == -1)
          printf("stack empty");
    else
    {
          return (s.a[s.top]);
    }
}
```

Implementing a Stack: Linked List

- Advantages:
  - always constant time to push or pop an element
  - can grow to an infinite size
- Disadvantages
  - the common case is the slowest of all the implementations
  - can grow to an infinite size
- Basic implementation
  - list is initially empty
  - *push()* method adds a new item to the head of the list
  - *pop()* method removes the head of the list

Implementing a Stack: Linked List

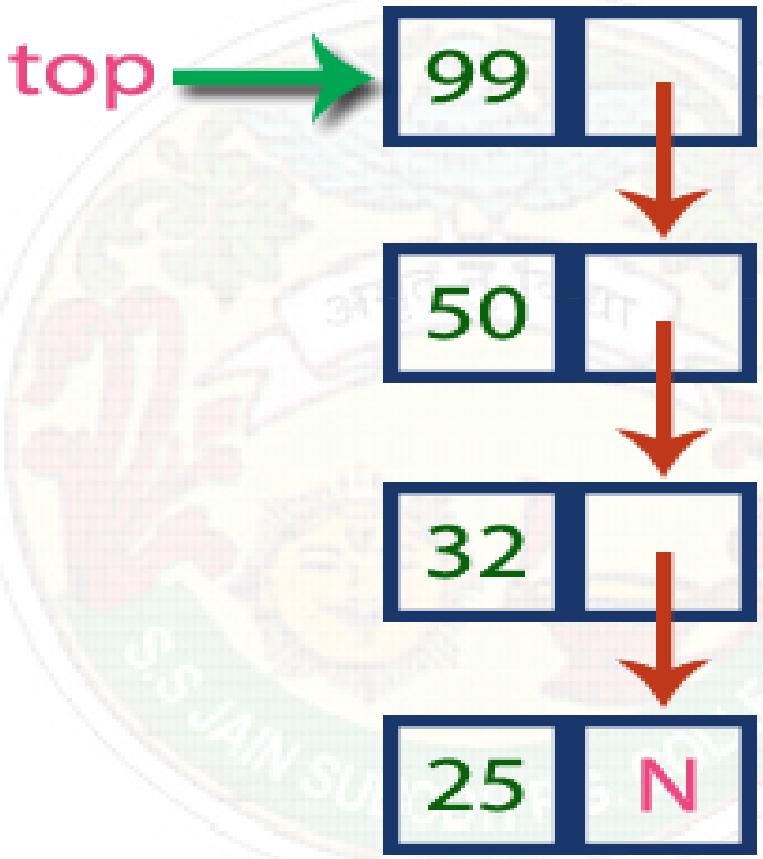
- start become top pointer
- Insertion at start(push)
- Deletion at start(pop)
- Reading element of start(peep)

```
typedef struct link
{
        int info;
        struct link *next;
}node;
node *top = NULL;
```

Implementing a Stack: Linked List

```
void push()
{
        node *n;
        n = (node *) malloc(sizeof(node));
        printf("Enter node data");
        scanf("%d", &n->info);
        n->next = top;
        top = n;
}
```

Example



Implementing a Stack: Linked List

```
void pop()
{
        node* p;
        if(top = = NULL)
                printf("Underflow: No elements");
        else
        {
                p = top;
                printf("Element:%d", p->info);
                top = top->next;
                p->next = NULL;
                free(p);
        }
}
```

Implementing a Stack: Linked List

```
void peep()
{
        node *p;
        if(top = = NULL)
                printf("Underflow:stack empty");
        else
        {
                p = top;
                while(p != NULL)
                {
                        printf("%d", p->info);
                        p = p->next;
                }
        }
}
```


Thanks
S.S. JAIN SUBODH P.G. COLLEGE