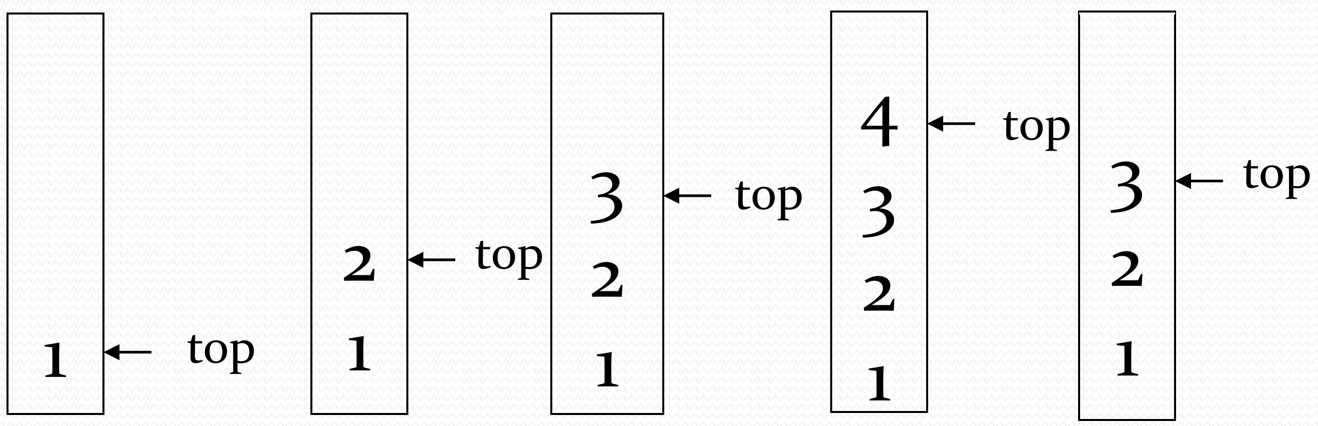
Stack

- Shital Dongre
- Assistant Professor
- VIT, Pune.

What is a stack?

- linear data structure
- It is an ordered group of homogeneous items of elements.
- Elements are added to and removed from the top of the stack
- Stack principle: LAST IN FIRST OUT(LIFO)
- It means the last element inserted is the first one to be removed
- Ex- stack of plates

Last In First Out



Applications of stack

- Balancing of symbols
- Infix to Postfix /Prefix conversion
- Redo-undo features at many places like in editors.
- Forward and backward feature in web browsers
- Used in many algorithms like Tower of Hanoi, tree traversals, topological graph sorting etc.
- Other applications can be Backtracking, N queen problem etc.

Operations on stack

- isEmpty
- Push
- Pop
- isFull

- Below is the complete algorithm
  Let arr[0..n-1] be the input array and element to be searched be x.
- Find the smallest Fibonacci Number greater than or equal to n. Let this number be fibM [m'th Fibonacci Number]. Let the two Fibonacci numbers preceding it be fibMm1 [(m-1)'th Fibonacci Number] and fibMm2 [(m-2)'th Fibonacci Number].
- While the array has elements to be inspected:
  - Compare x with the last element of the range covered by fibMm2
  - **If** x matches, return index
  - **Else If** x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.
  - **Else** x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate elimination of approximately front one-third of the remaining array.
- Since there might be a single element remaining for comparison, check if fibMm1 is 1. If Yes, compare x with that remaining element. If match, return index.

- i=min(offset+m2,n)
- Offset-It marks the range that has been eliminated,

me.

fibMm2	*fibMm1*	*fibM*	*offset*	i=min(offset+fibL n)	*arr[i]*	*Consequence*
5	8	13	0	5	45	Move one down, reset offset
3	5	8	5	8	82	Move one down, reset offset
2	3	5	8	10	90	Move two down
1	1	2	8	9	85	Return i

isEmpty - Returns true(1) if stack is empty, else false(0).

```
int isEmpty()
{
        if (top==-1)
          return 1;
        else
          return 0;
}
```

```
#define MAX_STACK_SIZE 100

int top= -1

int stack[MAX_STACK_SIZE]
```

isFull - Returns true(1) if stack is Full, else false(0).

```
int isFull()
{
        if (top==(MAX_STACK_SIZE -1))
          return 1;
        else
          return 0;
}
```

- **Push- Add item in stack**

```
void push( int num)
{
  if(isFull())
        printf("\n Stack is Full");

 top = top + 1;
 stack[top] = num;
}
```

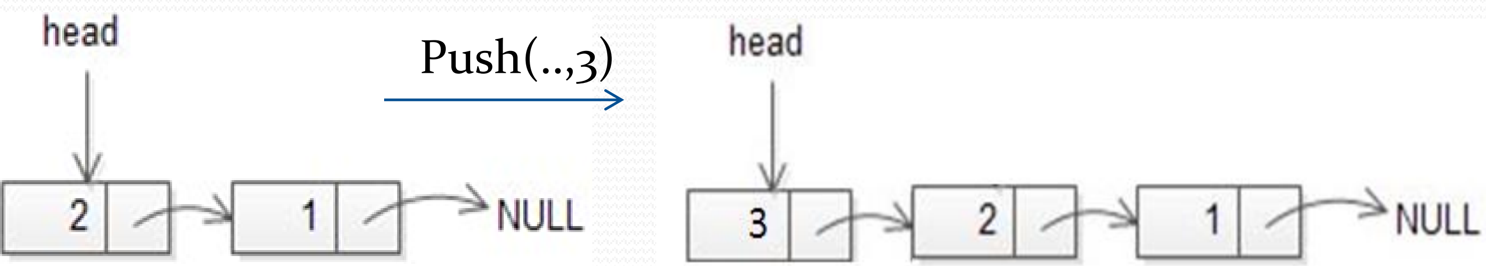
- **Pop- Remove item from stack**

```
int pop()
{
int num;

        if(isEmpty())
        printf("\n Stack is empty");

num=stack[top];
top--;
return num;
}
```

Stack using Linked list

- Extend stack size dynamically
- isFull() - condition not applicable
- isEmpty()- head node not available

```
void push(struct Node** head, int data)
{
    struct Node* node = (struct
   Node*)malloc(sizeof (struct Node));
   node->data =data;
    node->next = *head;
    *head = node;   //top
    }
```



```
void  pop(struct Node** head)
{
   if (isEmpty(*head))
      printf(" Stack is Empty");



   struct Node* temp = *head;
   *head = (*head)->next;
   int num = temp->data;
   free(temp);
    printf(" Popped element: %d", num);
   }
```