Date _____ Page _____

Day - 133

Stack

⇒ Stack is data structure that works on the LIFO principle (Last In First Out).

⇒ Every operation takes place only at the end, called the top of stack.

⇒ It is a linear DS.

⇒ When we define a stack as an abstract data type, then we are only interested in know the stack operation from user POV.

⇒ In simple mean, we are not interested in knowing the implementation but we want to know only what type of operation, we can perform.

~~Operations of stack~~

Operations:

[i]. Push: Insert the element at the top of the stack.

Date _____ Page _____

(iii). Pop: Delete the top element of the stack.

(iii). Top: Return the top element of the stack.

(iv). Size: Return the size of the stack.

(v). Emply: If stack is empty then it returns 1 otherwise 0.

(vi). Full: If stack is full then it returns 1 otherwise 0.

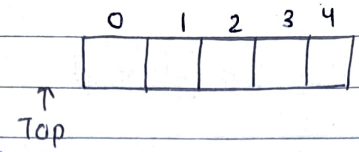
⇒ Stack follows the Last In First Out property (LIFO) but it can't follow First In Last Out property (FILO) because here we can't gaurantee that the first element that comes will comes out in the last.

* Implementation of stack:

⇒ We can implement stack by using array and linked list.

By using array

Date _____ Page _____



⇉ First, we create an array (suppose of 5 size).

⇉ We will init. a variable top that points to the top of the stack.

Push

⇉ First, we will increase the top by 1.

⇉ Then insert the element.

⇉ If the top becomes equal to the size of the stack then we can't insert more element.

⇉ This condition is called stack overflow.

Pop

⇉ First, we will delete the top element.

⇉ Then, decrease the top by 1.

⇉ If there is no element in the stack.

⇉ Then if we do pop operation. It will not perform.

⇉ And this condition is called stack underflow.

Date _____ Page _____

Code

```
class stack{
    int *arr;
    int size;
    int top;
    public:
    stack(int s){
        size = s;
        top = -1;
        arr = new int[s];
    }
    void push(int value){
        top++;
if(top==size-1) arr[top]= value;
cout<<"Stack Overflow";
return;
}
    }

    void pop(){
        if(top==-1){
        cout<<"Stack underflow";
        return;
        }
        top--;
    }
```

Date _____ Page _____

By using Linked List

⇒ First, we create a node then attach to the LL.

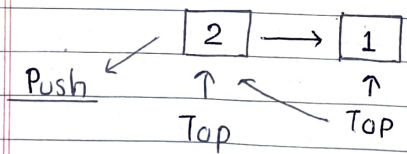
⇒ But in this, we have to manage a head pointer, top pointer.

⇒ Also, if we delete a node or perform pop operation then we have to traverse ~~a~~ the linked list to ~~get the~~ shift the top to the previous element in singly LL.

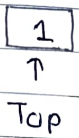
⇒ We can also use DLL.

⇒ But we have a better approach than this.

⇒ We will insert the element at the ~~top of~~ front of the LL.



Pop



⇒ We also take a size variable.

Date _____ Page _____

Code

```
class Node{
    public:
    int data;
    Node *next;

    Node(int value){
        data = value;
        next = NULL;
    }
};

class stack{
    Node *top;
    int size;
    public;
    stack(){
        top = NULL;
        size = 0;
    }
    void push(int value){
        Node *temp = new Node(value);
        temp->next = top;
        top = temp;
        size++;
    }
```

Date _____ Page _____

```
void pop(){
    ~~Node * temp =~~
    if( top == NULL){
        cout << "Stack Underflow";
    }else{
    Node * temp = top;
    top = top -> next;
    delete ~~top;~~ temp;
    }
}

int peek(){
    if(top == NULL){
        cout << "stack is empty\n";
        return -1;
    }else{
        return top -> data;
    }
}
```

* STL library for stack

⇒ We can use predefined stack.

```
stack <int> s;
s.push(5);
s.pop();
s.top();
```