08/11/2023
Wednesday

STACK

Stack is a linear data structure. It follows LIFO (Last in First Out) principle.

Stack underflow - Stack is empty and we are in the process to remove an item from stack is stack underflow.

Stack overflow - Stack is full and we are in the process to insert more items in stack. This condition is stack overflow.

Stack STL -

- **Header file** = #include<stack>
- **Creation** = stack <data_type> stack_name;

for instance, stack <int> st;
stack <char> st;
stack <string> st;

- **Insert** = stack_name.push(data);

for instance, st.push(10);

- Remove = stack_name.pop( );

  for instance, st.pop( );

- Size = stack_name.size( );

  for instance, st.size( );

- Check if empty = stack_name.empty( );

  for instance, st.empty( );

- fetch top item = stack_name.top( );

  for instance, st.top( );

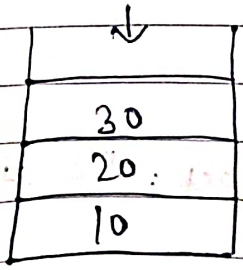
Code -

```
#include <iostream>
#include <stack>
using namespace std;

int main() {
  //creation
   stack<int>st;

  //insertion
   st.push(10);
   st.push(20);
   st.push(30);

  //size
   cout<<"size of stack: "<<st.size()<<endl;
```

```
//remove
st.pop();

//emptiness
if(st.empty()){
  cout<<"stack is empty"<<endl;
}
else{
  cout<<"stack is not empty"<<endl;
}

//top element
  cout<<"top element: "<<st.top()<<endl;
}
```

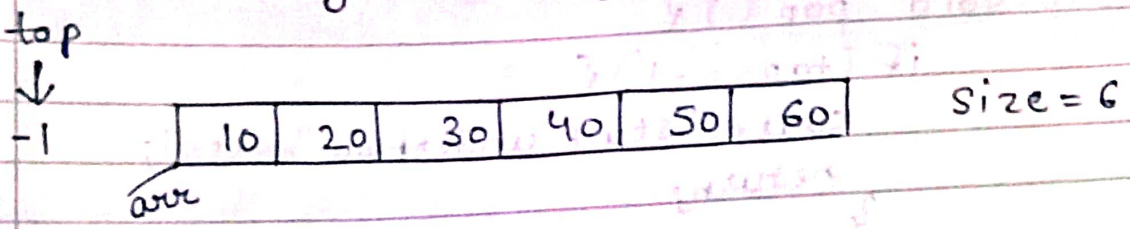
O/P- size of stack: 3
stack is not empty
top element: 20

Note - Stack ke elements ko left, right and bottom se access nahi kr skte, only one way is there i.e. top. Agr top ke bottom element ko access krna hai to top element ko pop krna pdega. And ek time par sirf ek hi element ko access kar skte hai.



Note- When top is at -1, it means there is no element present in the stack.

* **Implementing stack by own using array -**

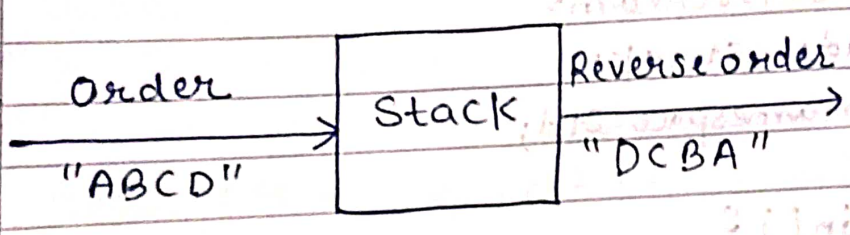


Code-

```
class stack {
public:
int * arr;
int size;
int top;

stack (int size) {
    arr = new int [size];
    this->size = size;
    this->top = top;
  }

// push
void push (int data) {
    if (top == size - 1) {
        cout << "stack overflow" << endl;
        return;
      }
    else {
        top ++;
        arr [top] = data;
      }
}
```

```
//pop
void pop () {
    if (top == -1) {
        cout << "stack underflow" << endl;
        return;
    }
    else {
        top--;
    }
}

//size
int getSize () {
    return top+1;
}

// fetch top
int getTop () {
    if (top == -1) {
        cout << "stack is empty" << endl;
        return -1;
    }
    else {
        return arr[top];
    }
}
```

```
// check if empty
bool isEmpty(){
    if (top == -1){
        return true;
    }
    else{
        return false;
    }
}
};
```

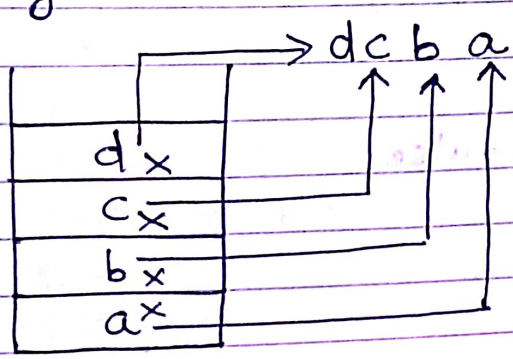


Left se koi ordering bheji and right se us ordering ka reverse order mil rha hai.

Note - Stack is Recursion, Recursion is stack. Ye line sirf hmare liye hai kisi or ko explain krte time nhi bolni.
Jo kaam humne recursion se kiya hai, usi kaam ko hum stack ki help se bhi kr skte hai. Kyuki recursion mai bhi call stack hi bn rha hota hai.

* Reverse string using stack -

String = "abcd"



① cout << st.top();
② st.pop();

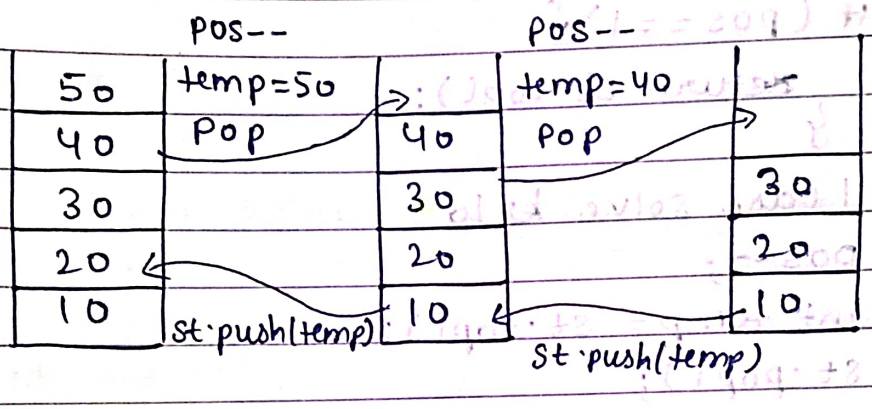
Code -

```
#include<iostream>
#include<stack>
using namespace std;

int main(){
  string str="abcd";
  stack<char> st;

  for(auto ch: str){
    char c= ch;
    st.push(c);
  }

  while(!st.empty()){
    cout<< st.top();
    st.pop();
  }
}
```

* **Middle element of the stack -**

Mid in case of odd = size/2 + 1 = pos

Mid in case of even = size/2 = pos



Base case - if (pos == 1) → mid found

Step 1: Pos--
2: temp = st.top()
3: st.pop()
4: Recursive call
5: Backtracking → st.push(temp)

✍ temp isliye create kiya hai so that recursive call se return krte time stack ke elements ko restore kiva ske using BT.

Code-

```
#include <iostream>
# include <stack>
using namespace std;

int solve (stack<int>&st , int &pos){
    // base case
    if (pos ==1){
        return st.top();
    }
    // I case solve krlo
    pos--;
    int temp = st.top();
    st.pop();

    // Recursive call
    return solve (st, pos);

    // backtracking
    st.push (temp);
}

int middle (stack<int >&st){
    int size= st.size();
    int pos= 0;
    // stack is empty
    if (st.empty()){
        cout << "stack is empty , no mid element"<<endl;
        return -1;
    }
```

```
//stack is not empty
else{
    if(size &1){
        pos = size/2 +1;
    }
    else{
        pos = size/2;
    }
}

return solve(st, pos);
}

int main(){
    stack<int>st;
    st.push(10);
    st.push(20);
    st.push(30);
    st.push(40);
    cout<<"middle element: "<<middle(st)<<endl;
}
```

I/P - 10 20 30 40

O/P - 30

* Insert an element at the bottom of a stack -

Given a stack of N integers and an element X. You have to insert X at the bottom of stack.

Example 1 -

I/P - N = 5
X = 2
St = {4, 3, 2, 1, 8}

O/P - {2, 4, 3, 2, 1, 8}

= 2 is inserted at the bottom of the stack.

Code -

```
class solution {
public:
    stack<int> insertAtBottom(stack<int> St, int X)
    {
        //base case
        if(st.empty()){
            st.push(X);
            return St;
        }
        // 1 case solve krlo
        int temp= st.top();
        st.pop();
        //recursive call
        St = insertAtBottom(st, X);
        //BT
        st.push(temp);
        return St;
} };
```

* **Reverse a stack -**

Given a stack. Reverse it.

Example 1-

I/P- St = {3, 2, 1, 7, 6}

O/P- {6, 7, 1, 2, 3}

Recursive Approach- Base case - Jb stack empty ho jaye, then return.

Step 1- int temp= St.top();
Step 2- St.pop();
Step 3- Make recursive calls until base case achieved.
Step 4- Insert temp at bottom of stack during backtracking.

Code-

```
class solution{
public:
  void insertAtBottom(stack<int> &St, int X){
    //base case
    if(St.empty()){
      St.push(X);
      return;
    }
    // Solve 1 case
    int temp= St.top();
    St.pop();
```

```
    //recursive call
    insertAtBottom(St, x);
    //BT
    St.push(temp);
    return;
}

void Reverse(stack<int>&St){
    if(St.empty()){
        return;
    }
    //solve 1 case
    int temp = St.top();
    St.pop();
    //recursive call
    Reverse(St);
    //BT
    insertAtBottom(St, temp);
}
};
```

Note - Jb bhi stack mai top element find out kr rhe ho ya pop kre stack mai se, always check if stack is empty st.empty()

* **Sort Stack -**

Example -

I/P - stack - 11 2 32 3 41

O/P - 41 32 11 3 2

Recursive approach - Base case - when stack gets empty, return;

Step 1- int temp = st.top();

Step 2- st.pop();

Step 3- Make a recursive call until base case is achieved.

Step 4- Insert temp in stack using insertInSortedStack function while backtracking.

Code -

```
void insertInSortedStack (stack<int> &st, int data) {
    //base case

    if (st.empty() || data > st.top()) {
        st.push(data);
        return;
    }
    // 1 case solve kro
    int temp = st.top();
    st.pop();
    //recursive call
    insertInSortedStack (st, data);
    //BT
    st.push(temp);
    return;
}
```

```
void sortStack (stack<int>&st){
    //base case
    if (st.empty()){
        return;
    }
    //1 case solve bro
    int temp= st.top();
    st.pop();
    //recursive call
    sortStack (st);
    //BT
    insertInSortedStack (st, temp);
    return;
}
```