

Stacks

What is Stack?

Stack is a linear data structure in which operations are performed in a particular manner.

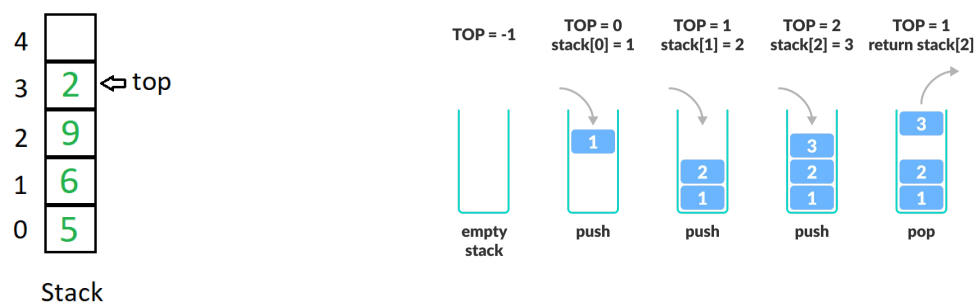
operations are performed on (LIFO) Last in first out or (FILO) First in Last out manner

A new data element is stored by pushing it on the top of the stack. And a data element is retrieved by popping the top element off the stack and returning it.

A stack can be implemented using array as well as linked list but using a linked list is more efficient due to its dynamic nature.

Stacks provide 4 basic operations for interaction

1. `push()` - Adds data to the top of stack
2. `pop()` - Returns and removes data from the top of stack
3. `top()` - Returns data from top of stack (Without removing it)
4. `isEmpty()` - Checks if a stack is empty or not (Returns a Boolean Value)



In the above image, although item 2 was kept last, it was removed first. This is exactly how the LIFO (Last In First Out) Principle works.



Why & When Stack?

A Stack data structure is useful when order of action is important. A stack ensures that a system does not move to a new action before completing those before it.

Some real-life usage of stack are listed below

1. Reversing: By Default a stack will reverse its input as it follows first in last out principle
2. Undo/Redo: This approach can be used in editors to implement the undo and redo functionality. The state of program can be push into a stack each time a change is made. In order to undo use `pop()` to return and remove the last change.
3. Recursion: Recursive functions use something called "Call Stack". When a program calls a recursive function that function goes on top of the stack.
4. Call Stack: Programming languages use a data stack to execute code. When a function is called, it is added to the call-stack and removed once completed.
5. In Browser: The back-button in browser saves all the URL you visit in a stack. Each time you visit a new page its URL gets added to top of stack. And when u press back button it is removed and returned from the stack using the `pop()` function.

Pros and Cons of Stack

 Pros	 Cons
<u>Does not take much effort to add a new element as compared to array.</u>	You have no access to underneath element, only element present on top is accessible.
<u>In stack we can allocate memory dynamically.</u>	Because of dynamic memory allocation if we not use all memory space then there will be wastage of memory and we'll suffer memory leak.
<u>Untitled</u>	

Working of a Stack

1. A variable called `top` is used to keep track of index at top.
2. On initializing a stack we set value of `top` to -1, and we can also check if stack is empty or not by comparing `nextIndex == -1`.
3. On pushing a new element we place the new element in the position pointed by `top` and increase value of `top`.
4. On popping top element we return the value of element pointed by `top` and then decrease value of `top`.
5. Before pushing we check if stack is already full.
6. before popping we check if stack is already empty.

Time-Complexity

Operation	Big O	Explanation
<u>Insertion (push()) at top</u>	O(1)	We always have to insert at top of stack which is a one step process
<u>Deletion (pop()) at top</u>	O(1)	We always have to remove from the top of stack which is a one step process
<u>Searching</u>	O(N)	We have to iterate from the top most element to the Nth element
<u>Accessing a particular element</u>	O(N)	We have to iterate from the top most element to the Nth element

Code Implementation

- Stack Using Array

```
#include <iostream>

using namespace std;

#define n 100

class stack{
    int* arr;    //arr and top are private data members.
    int top;

public:
    stack(){
        arr = new int[n];    //dynamically allocating n size for the array/stack
        top = -1;
    }
    //push
    void push(int x){
        if(top==n-1){
            cout<<"Stack Overflow"<<endl;
            return;
        }
        top++;
        arr[top] = x;
    }
    //pop
    void pop(){
        if(top==-1){
            cout<<"Stack underflow"<<endl;
            return;
        }
        top--;
    }
    //top
```

```

int Top(){
    if(top==-1){
        cout<<"Empty Stack"<<endl;
        return -1;
    }
    return arr[top];
}
//empty
bool empty(){
    return top==-1;
}
};

int main(){

    stack st;
    st.push(1);
    st.push(2);
    st.push(3);

    cout<<st.Top()<<endl;
    st.pop();
    cout<<st.Top()<<endl;
    st.pop();
    st.pop();
    st.pop();

    cout<<st.empty()<<endl;

    return 0;
}

```

- Stack Using Linked-List

```

#include <iostream>

using namespace std;

class node{
public:
    int data;
    node* next;
    node(int x){
        data = x;
        next = NULL;
    }
};

class stack{
    node* top;

public:
    stack(){
        top = NULL;
    }

    void push(int x){
        node* n = new node(x);
    }
}

```

```

        n->next = top;
        top = n;
    }

    void pop(){
        if(top==NULL){
            cout<<"Stack Underflow\n";
            return;
        }

        node* todelete = top;
        top = top->next;
        delete todelete;
    }

    int Top(){
        if(top==NULL){
            cout<<"Empty Stack\n";
            return -1;
        }
        return top->data;
    }

    bool empty(){
        return top==NULL;
    }
};

int main(){
    stack st;

    st.push(1);
    st.push(2);
    st.push(3);

    cout<<st.Top()<<endl;
    st.pop();

    cout<<st.Top()<<endl;
    st.pop();

    cout<<st.Top()<<endl;
    st.pop();

    cout<<st.Top()<<endl;
    st.pop();

    cout<<st.empty()<<endl;

    return 0;
}

```

Using The Inbuilt Stack Data-Structure in STL

Syntax: `stack <data_type> stack_name;`

Some functions of stack

- `push()` Insert an element at the top of stack $O(1)$
- `pop()` removes a element from the top of stack $O(1)$
- `top()` Access the top element of the stack $O(1)$
- `empty()` Returns whether the stack is empty or not $O(1)$
- `size()` Returns size of stack $O(1)$

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    stack <int> stack_name; // Declare a stack

    for (int i = 0; i < 5; i++) { //This loop takes input and adds it to stack
        int temp;
        cin >> temp;
        stack_name.push(temp);
    }

    //This loop prints the element of stack and clears the stack at the ending of loop
    while (!stack_name.empty()) {
        cout << stack_name.top() << " ";
        stack_name.pop();
    }

    return 0;
}
```

Some Standard Questions:

Valid Parentheses

Valid Parentheses - LeetCode

Given a string `s` containing just the characters `'('`, `)'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid. An input string is valid if: Open brackets must be closed by the same type of brackets. Open

🔗 <https://leetcode.com/problems/valid-parentheses/>



Given a string `s` containing just the characters `'('`, `)'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Approach:

1. Iterate over the input string
2. if the `iTH` element of string is opening bracket then `push()` it
3. else it is a closing bracket, then check if the top of stack contains corresponding opening bracket.
4. if yes then `pop()` or else return false.
5. In the end check if stack is empty is yes then return true

```
bool isValid(string s)
{
    stack <char> sta;

    // Iterate over the input string
    for(int i=0;i<s.length();i++)
    {
        // If ith element is opening bracket then push
        if(s[i]=='(' or s[i]=='[' or s[i]=='{')
        {
            sta.push(s[i]);
        }

        // Else it is a closing bracket
        else
        {
            // Check if stack is not empty this avoids runtime error
            if(sta.empty()){return false;}

            // Now check if the top of stack contains corresponding opening bracket
            //If yes the pop it else return false

            if(s[i]==')')
            {
                if(sta.top()=='('){sta.pop();}
                else{return false;}
            }
            if(s[i]==']')
            {
                if(sta.top()=='['){sta.pop();}
                else{return false;}
            }
            if(s[i]=='}')
            {
                if(sta.top()=='{'){sta.pop();}
                else{return false;}
            }
        }
    }

    // If stack is empty that means every thing is fine
    if(sta.empty()){return true;}
    else{return false;}
}
```

Remove Adjacent Duplicates

Remove All Adjacent Duplicates In String - LeetCode

You are given a string `s` consisting of lowercase English letters. A duplicate removal consists of choosing two adjacent and equal letters and removing them. We repeatedly make duplicate removals

🔗 <https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string/>



You are given a string `s` consisting of lowercase English letters. A **duplicate removal** consists of choosing two **adjacent** and **equal** letters and removing them.

We repeatedly make **duplicate removals** on `s` until we no longer can.

Input: `s = "abbaca"`

Output: `"ca"`

Explanation:

For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move. The result of this move is that the string is "aaca", of which only "aa" is possible, so the final string is "ca".

Input: `s = "azxxzy"`

Output: `"ay"`

Approach:

1. Iterate over the input array
2. If the `i`th element and the top element of stack is same then pop it.
3. or else push it
4. Then iterate over the stack and add each element in an empty string
5. Reverse the string and return it.

```
string removeDuplicates(string s)
{
    stack<char> sta;

    // Empty string to store answer in the end
    string ans = "";

    // Loop over input string
    for(int i=0;i<s.size();i++)
    {
        // if stack is empty then its the first element (i=0)
        // If s[i]!=sta.top() then the next and previous element is same
        if(sta.empty() || s[i]!=sta.top()){sta.push(s[i]);}
        else
```



```

    {
        sta.pop();
    }
}

//iterate over the stack
while(!sta.empty())
{
    // add elements of stack in empty string
    ans.push_back(sta.top());
    sta.pop();
}

// Reverse the final ans string bcz the first element will be last element in stack while popping
reverse(ans.begin(), ans.end());
return ans;
}

```

Patrice problems:

Basic Problems -

Reverse the Words of a String using Stack - GeeksforGeeks

Reverse the Words of a String using Stack Given string str consisting of multiple words, the task is to reverse the entire string word by word.

 <https://www.geeksforgeeks.org/reverse-the-sentence-using-stack/>



Standard Problems -

Balanced Parentheses! - InterviewBit

Balanced Parentheses!: Problem Description Given a string A consisting only of '(' and ')'. You need to find whether parentheses in A is balanced or not ,if it is balanced then return 1 else return 0.

 <https://www.interviewbit.com/problems/balanced-parentheses/>



InterviewBit

Reverse a stack using recursion - GeeksforGeeks

Reverse a stack using recursion Write a program to reverse a stack using recursion. You are not allowed to use loop constructs like while, for..etc, and you can only use the following ADT functions on Stack

 <https://www.geeksforgeeks.org/reverse-a-stack-using-recursion/>



Next Greater Element - GeeksforGeeks

Given an array, print the Next Greater Element (NGE) for every element. The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which

 <https://www.geeksforgeeks.org/next-greater-element/>



Next Smaller Element - GeeksforGeeks

Given an array, print the Next Smaller Element (NSE) for every element. The Smaller smaller Element for an element x is the first smaller element on the right side of x in array. Elements for which no

 <https://www.geeksforgeeks.org/next-smaller-element/>



Nearest Smaller Element - InterviewBit

Nearest Smaller Element: Given an array, find the nearest smaller element $G[i]$ for every element $A[i]$ in the array such that the element has an index smaller than i . More formally, $G[i]$ for an element $A[i]$ =

 <https://www.interviewbit.com/problems/nearest-smaller-element/>




InterviewBit

CP Problems -

Little Shino and Pairs | Practice Problems

Prepare for your technical interviews by solving questions that are asked in interviews of various companies. HackerEarth is a global hub of 5M+ developers. We help companies accurately assess,

 <https://www.hackerearth.com/practice/data-structures/stacks/basics-of-stacks/practice-problems/algorithm/little-shino-and-pairs/>



Be a Better Programmer

Fastest growing developer community in the world, where they can **learn**, **compete** and **get hired**.

Contest Page | CodeChef

CodeChef was created as a platform to help programmers make it big in the world of algorithms, computer programming, and programming contests. At CodeChef we work hard to revive the geek in you by hosting a programming contest at the start of the month and two smaller programming challenges at the middle and end

 <https://www.codechef.com/problems/COMPILER>

Problem - C - Codeforces


This is yet another problem dealing with regular bracket sequences. We should remind you that a bracket sequence is called regular, if by inserting "+" and "1" into it we can get a correct mathematical

 <https://codeforces.com/contest/5/problem/C>



Problem - C - Codeforces

Petya recieved a gift of a string s with length up to characters for his birthday. He took two more empty strings t and u and decided to play a game. This game has two possible moves: Petya wants to get

 <https://codeforces.com/contest/797/problem/C>



SPOJ.com - Problem JNEXT


DevG likes too much fun to do with numbers. Once his friend Arya came and gave him a challenge, he gave DevG an array of digits which is forming a number currently (will be called as given number).

 <https://www.spoj.com/problems/JNEXT/>



Largest Rectangle in Histogram - LeetCode


Level up your coding skills and quickly land a job. This is the best place to expand your knowledge and get prepared for your next interview.

 <https://leetcode.com/problems/largest-rectangle-in-histogram/>



Contest Page | CodeChef

CodeChef was created as a platform to help programmers make it big in the world of algorithms, computer programming, and programming contests. At CodeChef we work hard to revive the geek in you by hosting a programming contest at the start of the month and two smaller programming challenges at the middle and end

 <https://www.codechef.com/LTIME94B/problems/LUNCHTIM>