Stack Data Structure (Introduction and Program)

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

Mainly the following three basic operations are performed in the stack:

* **Push:**Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
* **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
* Peek or Top: Returns top element of stack.
* **isEmpty:**Returns true if stack is empty, else false.

[](https://www.geeksforgeeks.org/wp-content/uploads/gq/2013/03/stack.png)

**How to understand a stack practically?**  
There are many real life examples of stack. Consider the simple example of plates stacked over one another in canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO/FILO order.

**Time Complexities of operations on stack:**

push(), pop(), esEmpty() and peek() all take O(1) time. We do not run any loop in any of these operations.

**Applications of stack:**

* [Balancing of symbols](https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-an-expression/)
* [Infix to Postfix](http://quiz.geeksforgeeks.org/stack-set-2-infix-to-postfix/) /Prefix conversion
* Redo-undo features at many places like editors, photoshop.
* Forward and backward feature in web browsers
* Used in many algorithms like [Tower of Hanoi,](https://www.geeksforgeeks.org/recursive-functions/) [tree traversals](https://www.geeksforgeeks.org/618/), [stock span problem](https://www.geeksforgeeks.org/the-stock-span-problem/), [histogram problem](https://www.geeksforgeeks.org/largest-rectangular-area-in-a-histogram-set-1/).
* Other applications can be Backtracking, [Knight tour problem](https://www.geeksforgeeks.org/backtracking-set-1-the-knights-tour-problem/), [rat in a maze](https://www.geeksforgeeks.org/backttracking-set-2-rat-in-a-maze/),[N queen problem](https://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/) and [sudoku solver](https://www.geeksforgeeks.org/backtracking-set-7-suduku/)

**Implementation:**  
There are two ways to implement a stack:

* Using array
* Using linked list

**Implementing Stack using Arrays**

|  |
| --- |
| /\* C++ program to implement basic stack     operations \*/  #include<bits/stdc++.h>  using namespace std;    #define MAX 1000    class Stack  {      int top;  public:      int a[MAX];    //Maximum size of Stack        Stack()  { top = -1; }      bool push(int x);      int pop();      bool isEmpty();  };    bool Stack::push(int x)  {      if (top >= MAX)      {          cout << "Stack Overflow";          return false;      }      else      {          a[++top] = x; //first add 1 to top and then use it for index.          return true;      }  }    int Stack::pop()  {      if (top < 0)      {          cout << "Stack Underflow";          return 0;      }      else      {          int x = a[top--]; **//First use the original top value and then subtract**.          return x;      }  }    bool Stack::isEmpty()  {      return (top < 0);  }    // Driver program to test above functions  int main()  {      struct Stack s;      s.push(10);      s.push(20);      s.push(30);        cout << s.pop() << " Popped from stack\n";        return 0;  } |

**Pros:** Easy to implement. Memory is saved as pointers are not involved.  
**Cons:** It is not dynamic. It doesn’t grow and shrink depending on needs at runtime.

10 pushed to stack

20 pushed to stack

30 pushed to stack

30 popped from stack

Top item is 20

**Implementing Stack using Linked List**

|  |
| --- |
| // C program for linked list implementation of stack  #include <stdio.h>  #include <stdlib.h>  #include <limits.h>    // A structure to represent a stack  struct StackNode  {      int data;      struct StackNode\* next;  };    struct StackNode\* newNode(int data)  {      struct StackNode\* stackNode =                (struct StackNode\*) malloc(sizeof(struct StackNode));      stackNode->data = data;      stackNode->next = NULL;      return stackNode;  }    int isEmpty(struct StackNode \*root)  {      return !root;  }    void push(struct StackNode\*\* root, int data)  {      struct StackNode\* stackNode = newNode(data);      stackNode->next = \*root;      \*root = stackNode;      printf("%d pushed to stack\n", data);  }    int pop(struct StackNode\*\* root)  {      if (isEmpty(\*root))          return INT\_MIN;      struct StackNode\* temp = \*root;      \*root = (\*root)->next;      int popped = temp->data;      free(temp);        return popped;  }    int peek(struct StackNode\* root)  {      if (isEmpty(root))          return INT\_MIN;      return root->data;  }    int main()  {      struct StackNode\* root = NULL;        push(&root, 10);      push(&root, 20);      push(&root, 30);        printf("%d popped from stack\n", pop(&root));        printf("Top element is %d\n", peek(root));        return 0;  } |

Run on IDE

Output:

10 pushed to stack

20 pushed to stack

30 pushed to stack

30 popped from stack

Top element is 20

**Pros:** The linked list implementation of stack can grow and shrink according to the needs at runtime.  
**Cons:** Requires extra memory due to involvement of pointers.