Data Structures

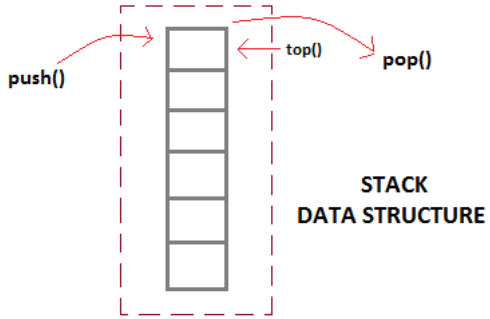
" Data Structures are widely used to organize data into unique structures to enhance programs performance. "

Test Yourself !

If you have studied all the lessons of Data Structure, then evaluate yourself by taking these tests.

Stacks

Stack is an abstract data type with abounded(predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the top of the stack, the only element that can be removed is the element that was at the top of the stack, just like a pile of objects.



Basic features of Stack

1. Stack is an ordered list of similar data type.
2. Stack is a **LIFO** structure. (Last in First out).
3. **push()** function is used to insert new elements into the Stack and **pop()** is used to delete an element from the stack. Both insertion and deletion are allowed at only one end of Stack called **Top**.
4. Stack is said to be in **Overflow** state when it is completely full and is said to be in **Underflow** state if it is completely empty.

Applications of Stack

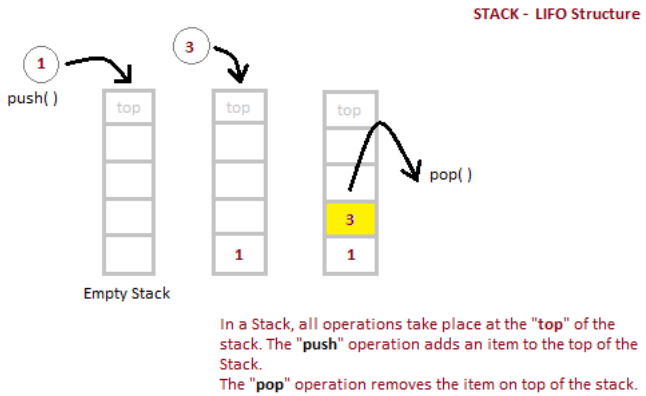
Coming Soon

The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.

There are other uses also like : **Parsing**, **Expression Conversion**(Infix to Postfix, Postfix to Prefix etc) and many more.

Implementation of Stack

Stack can be easily implemented using an Array or a Linked List. Arrays are quick, but are limited in size and Linked List requires overhead to allocate, link, unlink, and deallocate, but is not limited in size. Here we will implement Stack using array.



```
/*  Below program is written in C++ language  */

Class Stack
{
  int top;
  public:
  int a[10];    //Maximum size of Stack
  Stack()
  {
    top = -1;
  }
};

void Stack::push(int x)
{
  if( top >= 10)
  {
    cout << "Stack Overflow";
  }
  else
  {
    a[++top] = x;
    cout << "Element Inserted";
  }
}

int Stack::pop()
{
  if(top < 0)
  {
    cout << "Stack Underflow";
    return 0;
  }
  else
  {
    int d = a[--top];
    return d;
  }
}

void Stack::isEmpty()
{
  if(top < 0)
  {
    cout << "Stack is empty";
  }
  else
  {
    cout << "Stack is not empty";
  }
}
```

Position of Top	Status of Stack
-1	Stack is Empty

0	Only one element in Stack
N-1	Stack is Full
N	Overflow state of Stack

Analysis of Stacks

Below mentioned are the time complexities for various operations that can be performed on the Stack data structure.

- **Push Operation** : O(1)
- **Pop Operation** : O(1)
- **Top Operation** : O(1)
- **Search Operation** : O(n)

← Prev (heap-sort)

Next → (queue-data-structure)