Implementing Stack Using Array in Data Structures

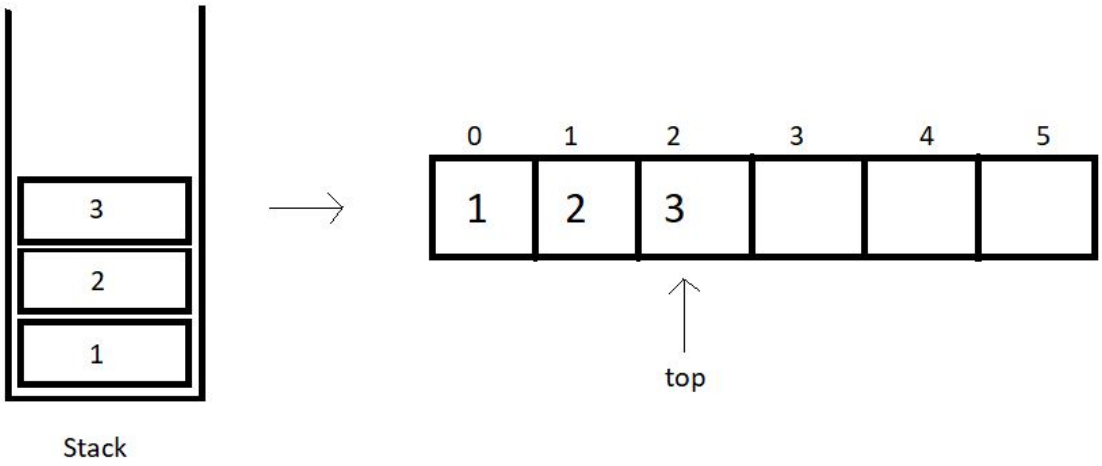
codewithharry.com/videos/data-structures-and-algorithms-in-hindi-23

In the last tutorial, we learned about the stack data structure and its applications in several programming phases. We also discussed some of the operations possible on a stack. Today, we'll try to implement these ideas on a stack using arrays. Although we have another choice of linked lists.

If you remember, a stack is a collection of elements following LIFO(Last In First Out); the element that gets pushed the last is the first one to come out of the stack.

Stack Using an Array

If we recall, arrays are linear data structures whose elements are indexed, and the elements can be accessed in constant time with their index. To implement a stack using an array, we'll maintain a variable that will store the index of the top element.



So, basically, we have few things to keep in check when we implement stacks using arrays.

1. A fixed-size array. This size can even be bigger than the size of the stack we are trying to implement, to stay on the safe side.

2. An integer variable to store the index of the top element, or the last element we entered in the array. This value is -1 when there is no element in the array.

We will try constructing a structure to embed all these functionalities. Let's see how.

```
struct stack{
    int size;
    int top;
    int* arr;
}
```

So, the struct above includes as its members, the size of the array, the index of the top element, and the pointer to the array we will make.

To use this struct,

1. You will just have to declare a struct stack
2. Set its top element to -1.
3. Furthermore, you will have to reserve memory in the heap using malloc.

Follow the example below for defining a stack:

```
struct stack S;
S.size = 80;
S.top = -1;
S.arr = (int*)malloc(S.size*sizeof(int));
```

We have used an integer array above, although it is just for the sake of simplicity. You have the freedom to customize your data types according to your needs.

We can now move on implementing the stack ADT, particularly their operators. We have in the list, push and pull, peek, and isempty/full operation. Let's visit them one by one.

push():

By pushing, we mean inserting an element at the top of the stack. While using the arrays, we have the index to the top element of the array. So, we'll just insert the new element at the index (top+1) and increase the top by 1. This operation takes a constant time, O(1). It's intuitive to note that this operation is valid until (top+1) is a valid index and the array has an empty space.

pop():

Pop means to remove the last element entered in the stack, and that element has the index top. So, this becomes an easy job. We'll just have to decrease the value of the top by 1, and we are done. The popped element can even be used in any way we like.

We will also see the other operations in our tutorial dedicated to operations on Stack. Today, we just used an array to implement stacks. You might have found this difficult to digest, but you should go over it again and again. Read the notes provided.

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. Don't forget to download the notes from the link given below. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial, where we'll code these implementations of stacks in C. Till then, keep learning.