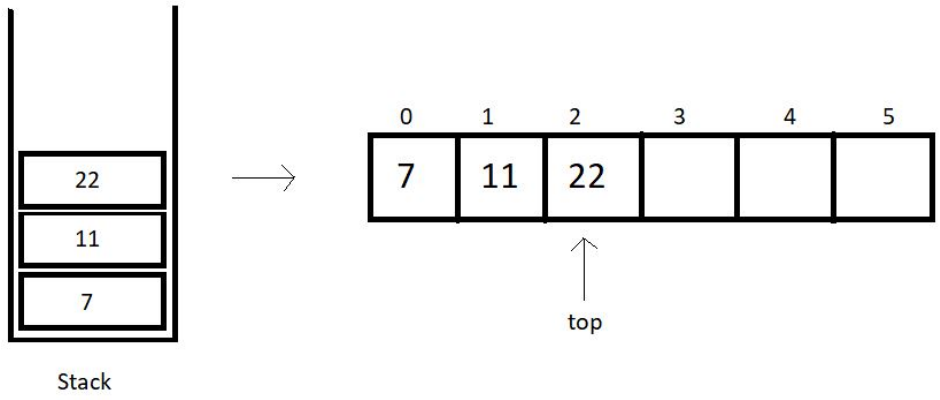
stackTop, stackBottom & Time Complexity of Operations in Stack Using Arrays

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-28

In the last tutorial, we talked about the peek operation and implemented it in C using arrays. Today, we will explore some new stack operations.

Take a deep breath of relief since the things we are discussing today are the easiest of all. We will first start by learning about two operations we have in stacks, stackTop and stackBottom.

Let's consider a stack array for the understanding purpose.



stackTop:

This operation is responsible for returning the topmost element in a stack. Retrieving the topmost element was never a big deal. We can just use the stack member *top* to fetch the topmost index and its corresponding element. Here, in the illustration above, we have the *top* member at index 2. So, the stackTop operation shall return the value 22.

stackBottom:

This operation is responsible for returning the bottommost element in a stack, which intuitively, is the element at index 0. We can just fetch the bottommost index, which is 0, and return the corresponding element. Here, in the illustration above, we have the bottommost element at index 0. So, the stackBottom operation shall return the value 7.

One thing one must observe here is that both these operations happen to work in a constant runtime, that is O(1). Because we are just accessing an element at an index, and that works in a constant time in an array.

Time complexities of other operations:

- **isEmpty():** This operation just checks if the top member equals -1. This works in a constant time, hence, O(1).
- **isFull():** This operation just checks if the top member equals size -1. Even this works in a constant time, hence, O(1).
- **push():** Pushing an element in a stack needs you to just increase the value of top by 1 and insert the element at the index. This is again a case of O(1).
- **pop():** Popping an element in a stack needs you to just decrease the value of top by 1 and return the element we ignored. This is again a case of O(1).
- **peek():** Peeking at a position just returns the element at the index, (top - position + 1), which happens to work in a constant time. So, even this is an example of O(1).

So, basically all the operations we discussed follow a constant time complexity.

Implementation:

I would suggest you all implement them on your own before moving ahead. I have attached the snippet below, for your referral.

Understanding the snippet below:

1. First of all, copy everything we have covered up to this point in your IDEs. I don't want to repeat them all and make this lengthy.

2. I suppose you have all the functions and declarations done.

3. Create an integer function *stackTop,* and pass the reference to the stack you created as a parameter. Just return the element at the index top of the array. And that's it.

```
int stackTop(struct stack* sp){
    return sp->arr[sp->top];
}
```

Code Snippet 1: Implementing *stackTop*

4. Create an integer function *stackBottom,* and pass the reference to the stack you created as a parameter. And then return the element at the index 0 of the array.

```
int stackBottom(struct stack* sp){
    return sp->arr[0];
}
```

Code Snippet 2: Implementing *stackBottom*

Here is the whole source code:

```
#include<stdio.h>
#include<stdlib.h>

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

int isEmpty(struct stack* ptr){
    if(ptr->top == -1){
            return 1;
        }
        else{
            return 0;
        }
}

int isFull(struct stack* ptr){
    if(ptr->top == ptr->size - 1){
        return 1;
    }
    else{
        return 0;
    }
}

void push(struct stack* ptr, int val){
    if(isFull(ptr)){
        printf("Stack Overflow! Cannot push %d to the stack\n", val);
    }
    else{
        ptr->top++;
        ptr->arr[ptr->top] = val;
    }
}

int pop(struct stack* ptr){
    if(isEmpty(ptr)){
        printf("Stack Underflow! Cannot pop from the stack\n");
        return -1;
    }
    else{
        int val = ptr->arr[ptr->top];
        ptr->top--;
        return val;
    }
}

int peek(struct stack* sp, int i){
    int arrayInd = sp->top -i + 1;
    if(arrayInd < 0){
        printf("Not a valid position for the stack\n");
        return -1;
    }
    else{
        return sp->arr[arrayInd];
    }
}
```

```
int stackTop(struct stack* sp){
    return sp->arr[sp->top];
}

int stackBottom(struct stack* sp){
    return sp->arr[0];
}

int main(){
    struct stack *sp = (struct stack *) malloc(sizeof(struct stack));
    sp->size = 50;
    sp->top = -1;
    sp->arr = (int *) malloc(sp->size * sizeof(int));
    printf("Stack has been created successfully\n");

    printf("Before pushing, Full: %d\n", isFull(sp));
    printf("Before pushing, Empty: %d\n", isEmpty(sp));
    push(sp, 1);
    push(sp, 23);
    push(sp, 99);
    push(sp, 75);
    push(sp, 3);
    push(sp, 64);
    push(sp, 57);
    push(sp, 46);
    push(sp, 89);
    push(sp, 6);
    push(sp, 5);
    push(sp, 75);

    // // Printing values from the stack
    // for (int j = 1; j <= sp->top + 1; j++)
    // {
    //     printf("The value at position %d is %d\n", j, peek(sp, j));
    // }

    return 0;
}
```

Code Snippet 3: Implementing *stackTop & stackBottom*

Now, since we have done pushing elements into the stack, we can call our functions, *stackTop* and *stackBottom*.

```
    printf("The top most value of this stack is %d\n", stackTop(sp));
    printf("The bottom most value of this stack is %d\n", stackBottom(sp));
```

Code Snippet 4: Calling functions *stackTop & stackBottom*

And the output we received was:

```
The top most value of this stack is 75
The bottom most value of this stack is 1
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 1: Output of the above program

Finally, I can say that it is all that we had when we implemented stacks using arrays. If you haven't gained enough confidence, hold on and go through these lectures again. We can now move to use linked lists to implement these stacks.

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. 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 be seeing stacks using linked lists. Till then keep coding.