C Code For Implementing Stack Using Array in Data Structures

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-24

In the last tutorial, we covered how to implement stacks by using arrays. We also dealt with the basic structure behind defining a stack with all the customizations. We also learned about some of the operations one could do while handling stacks. Today, we'll try implementing stacks using arrays in C.

I've attached the snippet below. Keeping that in mind will help you understand the implementation.

Understanding the code snippet 1:

1. So, the first thing would be to create the struct *Stack* we discussed in the previous tutorial. Include three members, an integer variable to store the size of the stack, an integer variable to store the index of the topmost element, and an integer pointer to hold the address of the array.

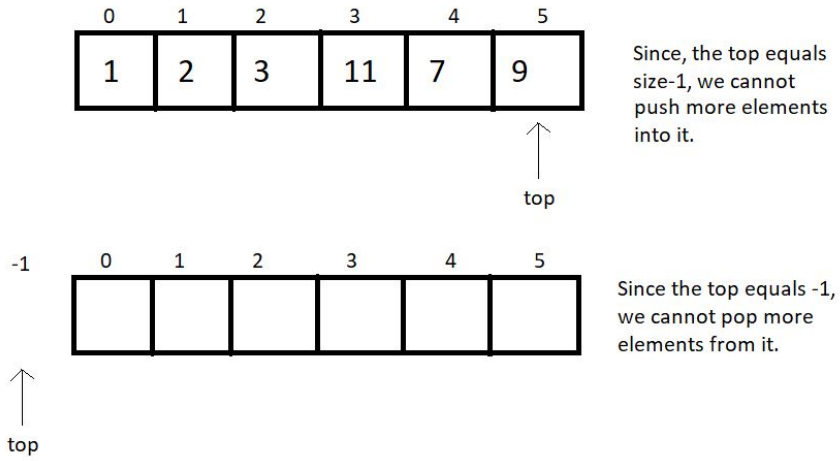
```
struct stack
{
    int size;
    int top;
    int *arr;
};
```

Code Snippet 1: Creating stack

2. In *main,* create a struct stack *s,* and assign a value 80(you can assign any value of your choice) to its size, -1 to its top, and reserve memory in heap using malloc for its pointer *arr*. Don't forget to include <stdlib> .

3. We have one more method to declare these stacks. We can define a struct stack pointer *s,* and use the arrow operators to deal with their members. The advantage of this method is that we can pass these pointers as references into functions very conveniently.

4. Before we advance to pushing elements in this stack, there are a few conditions to deal with. We can only push an element in this stack if there is some space left or the top is not equal to the last index. Similarly, we can only pop an element from this stack if some element is stored or the top is not equal to -1.



5. So, let us first write functions to check whether these stacks are empty or full.

6. Create an integer function *isEmpty,* and pass the pointer to the stack as a parameter. In the function, check if the top is equal to -1. If yes, then it's empty and returns 1; otherwise, return 0.

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

Code Snippet 2: Implementing isEmpty

7. Create an integer function *isFull,* and pass the pointer to the stack as a parameter. In the function, check if the top is equal to (size-1). If yes, then it's full and returns 1; otherwise, return 0.

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

Code Snippet 3: Implementing isFull

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;
    }
}

int main()
{
    // struct stack s;
    // s.size = 80;
    // s.top = -1;
    // s.arr = (int *) malloc(s.size * sizeof(int));

    struct stack *s;
    s->size = 80;
    s->top = -1;
    s->arr = (int *)malloc(s->size * sizeof(int));

    return 0;
}
```

Code Snippet 4: Implementing isEmpty and isFull

Since there is no element inside the stack, we can now check if it's empty.

```
    // Check if stack is empty
    if(isEmpty(s)){
        printf("The stack is empty");
    }
    else{
        printf("The stack is not empty");
    }
```

Code Snippet 5: Calling the function isEmpty

Output:

```
The stack is empty
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 1: Output of the above program

So, yes, that worked fine. Now, we can easily push some elements inside this stack manually to test this *isEmpty* function. This should not be a tough job. Just insert an element at top+1 and increment top by 1.

```
// Pushing an element manually
    s->arr[0] = 7;
    s->top++;

    // Check if stack is empty
    if(isEmpty(s)){
        printf("The stack is empty");
    }
    else{
        printf("The stack is not empty");
    }
```

Code Snippet 6: Inserting an element in the stack

Output after inserting an element:

```
The stack is not empty
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 2: Output of the above program

So, they are all working well. We can now proceed to implement our push, pop, and other operations. Let's reserve the other day for them and keep things here for today. Try implementing them on your own and tell me if you could. We'll see them in the next tutorial.

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 implement the operation, including the push and the pop in stacks. Till then, keep learning and coding.