Lesson:

Stack One Shot

PW SKILLS

Pre-Requisites

- Arrays

List of Concepts Involved

- Introduction to stacks
- Types of operations that can be performed on stacks
- Overflow
- Underflow
- Array implementation of a stack
- Linked list implementation of a stack
- Advantage of linked list implementation of stack over array implementation
- Disadvantage of linked list implementation of stack over array implementation
- Declaration of Stack in stl
- List of member functions

Topic 1: Introduction to stacks

A stack is a linear data structure. The thing that separates it from other linear data structures is how it performs insertion and deletion operations. A pile of plates at a cafeteria would be a good example to understand how stacks work. If someone wants to take a plate from the pile, he/she takes the one at the top. Similarly, if a new plate has to be added to the pile, it is placed at the top of the pile. We call this type of operation LIFO (last in first out) or FILO (first in last out).

A stack is a linear data structure that follows, LIFO (or FILO) approach for insertion or deletion of objects.

Topic 2: Types of operations that can be performed on stacks

Main operations-

1. push(): inserts an element onto the top of the stack.
2. pop(): removes and returns the top element from the stack.

Auxiliary operations-

1. top(): returns the top element of the stack.
2. size(): returns the current size of the stack.
3. isEmpty(): returns true if the stack is empty, else returns false.
4. isFull(): returns true if the stack is full, else returns false.

To understand them in a better way, let's assume we have a stack of maximum size 5 that currently contains 3 elements-

Let's try to call the auxiliary functions for the stack-

1. top(): it will return 4, the top element of the stack.
2. size(): it will return 3, the number of elements in the stack.
3. isFull(): it will return false, the number of elements in the stack(3) is not equal to the size of the stack(5) i.e. more elements can be added to the stack.
4. isEmpty(): it will return false, there are non-zero number of elements in the stack.

Now if we say push(3). The value 3 will be added at the top of the stack.

Let's try to call the auxiliary operations again-

1. top(): this time it will return 3, the new top element of the stack.
2. size(): it will return 4, the size of the stack.
3. isFull(): it will return false, the stack is not full yet.
4. isEmpty(): it will return false, the stack isn't empty.

Similarly if we say push(5). The value 5 will be added at the top of the stack.

Let's try to call the auxiliary operations again-

1. top(): this time it will return 5, the new top element of the stack.
2. size(): it will return 5, the size of the stack.
3. isFull(): it will return true this time, the number of elements in the stack is equal to the capacity of the stack.
4. isEmpty(): it will return false, the stack isn't empty.

Let's try the operation pop() this time. When we call the pop operation on the stack, the top element 5 is removed from the stack.

If we call it again, 3 will also be removed.

If we call it 3 more times, the stack will become empty

Let's try to call the auxiliary operations-

1. top(): since the stack is empty, this operation isn't valid. In such cases, we can throw an Exception stating that the stack is empty.
2. size(): it will return 0, the size of the stack.
3. isFull(): it will return false, there are no elements in the stack.
4. isEmpty(): it will return true, there are no elements in the stack.

Topic 3: Overflow

It is the name of the condition when we try to insert elements into our stack more than it can hold.For example,

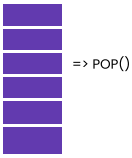
1. We create an integer stack of max size 5.

2. We insert 5 values into the stack 1->5.

3. Now even though we have already filled the entire stack, we try to fill in more elements into it.

Topic 4: Underflow

It is the name of the condition when we try to pop elements from an empty stack.| |



In such a scenario, we can throw an Exception stating there is an underflow.

Topic 5: Array implementation of a stack

In the array implementation of a stack, we use an array to represent the stack. The elements are added into the stack in the increasing order of the indices of the array. The last element inserted in the array is considered the top of the stack and is present at the highest index.

We start by declaring 3 things-

1. An integer variable to store the maximum size/capacity of the stack. Let's call it 'capacity'.
2. An array to represent the stack. Let's call it 'st'.
3. An integer variable denoting the position of the top element of the stack. Let's call it 'top'. We initialise it with -1, to denote that the stack is empty.

Now let's implement the stack functions-

top()

Since we already have a variable to represent the index of the top element of the stack. We can directly return the value at that index.

If the **top_idx** is equal to -1 i.e. there is no element in the stack, we can give an error saying that the stack is empty.

For instance,

We have an array of max capacity 5, with 3 elements in it.

Code

Time Complexity: O(1)

size()

Since there is 0 based indexing in the array and the elements are inserted into the stack in the increasing order of the array indices, the size of the array will be the position of top variable + 1.For instance, We have an array of max capacity 5, with 3 elements in it.

The size of the stack = 2(**top_idx**) + 1 = 3.

Code

Time Complexity: O(1)

isEmpty()

If the value of **top_idx** is -1, we can return true. Else it will be false.

Code

Time Complexity: O(1)

isFull()

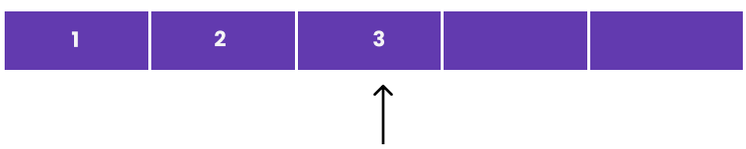
If the size of the stack is equal to the capacity of the stack, we can return true. Else it will be false.

Code

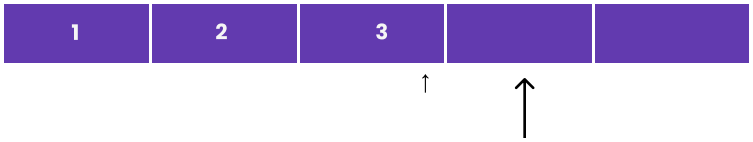
Time Complexity: O(1)

push()

1. The first thing we need to do before inserting an element into the stack is to check if the stack is full or not. We can do that with the isFull() function.
    a. If the stack is already full, we cannot add a new element to the stack. So, we can give the overflow error and return.
    b. Else we can increment the value of **top_idx** and place our new value at that position. For example, We have an array of max capacity 5, with 3 elements in it.



top_idx = 2 (it represents the index of the top element, 0-based indexing)
Let's insert an element into the stack.
First we increment the value of top_idx => top_idx = 3.



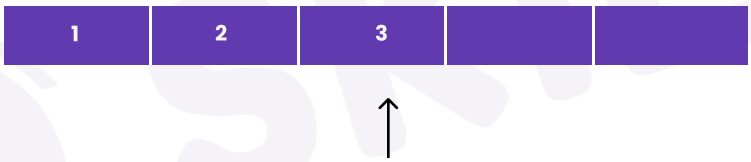
We place the new value at the index denoted by top.

Code

Time Complexity: O(1)

pop()

1. To remove an element from the stack, we start by checking if the stack is empty or not. We can do that using the isEmpty() function.
    a. If the stack is already empty, we can't remove any element from the stack. So we can give an underflow error and return.
    b. Else we can just decrement the value of top_idx. For example,
    We have an array of max size 5 that contains 3 elements in it.



top = 3
Let's pop an element from the stack. To do this we'll just decrement the value of top.

Now even if we have the value 3 present in our array as an element, by moving the top one place backward, we don't consider it a part of our stack. Our stack only consists of elements from index 0 to index top (both inclusive).
Code

Time Complexity: O(1)

Stack class Code

Topic 6: Linked list implementation of a stack

In the linked list implementation of a stack, we use a linked list to represent the stack. Each node of the linked list represents the element of the stack. The head of the linked list acts as the top of the stack and thus the insertion and deletion operations of the stack are performed at the head of the linked list.

=>3->4->2->1

Initially we create 2 things-

1. A node to represent the head of the linked list.
2. A variable to store the current size of the linked list.

Now let's implement the stack functions-

top()

Since the head of the linked list represents the top element of the stack, we can directly return the value stored in the head.

If the head is pointing to NULL i.e. there is no element in the stack, we can give an Exception stating that the stack is empty.For instance,

Code

Time Complexity: O(1)

size()

Since we have already created a variable to store the size of the linked list, we can return its value directly.

Code

Time Complexity: O(1)

isEmpty()

If the head of the linked list points to NULL, we can return true. Else it will be false.

Code

Time Complexity: O(1)

isFull()

In the case of the linked list representation, the amount of nodes that we can add to the linked list depends upon the memory limit of the system. So you'll automatically get an allocation failure message. In other words we don't need to create a separate function for it.

push()

The insertion of an element in the stack will be the same as the insertion of a node at the beginning of a linked list.

Steps-

1. Make a new node to insert in the list and assign it to a variable.
2. Make this new node point to the head of the linked list.
3. Make this new node the head of the list.
4. Increment the stack size.

Code

Time Complexity: O(1)

pop()

To pop an element from the stack, we just need to remove the head of the linked list.

Steps-

1. If the head of the list is pointing to NULL, give the underflow error and return.
2. Make 2 temporary variables to store the head and its value.
3. Make the head variable point to the second element of the list.

4. Return the value of the removed node.

Code

Time Complexity: O(1)

Stack class Code

Topic 7: Topic 7: Advantage of linked list implementation of stack over array implementation

The main advantage of using a linked list over arrays is that it is possible to implement a stack that can shrink or grow as much as needed. Using an array will put a restriction on the maximum capacity of the array which can lead to stack overflow. Here each new node will be dynamically allocated. so overflow is not possible.

Topic 8: Disadvantage of linked list implementation of stack over array implementation

Additional memory is required for storing the pointer.

Topic 9: Introduction of stack in STL

We have already studied what stacks are and what type of operations we can perform on stacks.Now we will learn about the stack class defined in the C++ STL.
It uses an encapsulated object of either vector or deque (by default) or list (sequential container class) as its underlying container, providing a specific set of member functions to access its elements.

Topic 10: Declaration

To use lists in a program, we need to start by including the stack header.

```
#include <stack>
```

Syntax for declaring a list

```
stack<data_type> name_of_stack;
```

Example:

```
1. stack<int> st1;
```

This creates a list that can store integer values, and the name of the stack is st1.

```
2. stack<char> st2;
```

This creates a list that can store character values, and the name of the stack is st2.

Note

1. It is compulsory to mention the data type of the type of variables the stack will be storing.
2. The data type can be a struct/class as well.

Topic 11: List of member functions

Function	Use
push(object)	Inserts an element at the top of the stack
pop()	Removes and returns the top element of the stack. An 'EmptyStackException' exception is thrown if we call pop() when the invoking stack is empty.
peek()	Returns the top element of the stack without removing it from the stack
size()	Returns the number of elements present in the stack
empty()	Returns if the stack is empty or not

Example

Output

Top element of the stack: 1
Is the stack empty? No
Size of the stack: 1

Top element of the stack: 2
Is the stack empty? No
Size of the stack: 2

Top element of the stack: 3
Is the stack empty? No
Size of the stack: 3

Stack before popping the top element: 1 2 3
Stack after popping the top element: 1 2

Upcoming lecture

- Problems based on stack.