Stack

A stack is defined as a restricted list where all insertion and deletions are made only at one end which is the top. Each stack has a data member commonly named as "top", which points to the topmost element in the stack. There are two basic operations that can be performed on a stack they are:

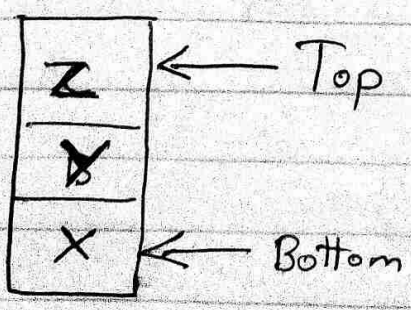
1. push
2. pop

Insertion of an element in the stack is called as push.

Deletion of an element from the stack is called pop.

In stack we cannot access data elements from any intermediate position other than the top position.

Eg:- Consider Stack S = (X, Y, Z) here X is the bottommost element and Z is the topmost element



Stack Operation

Three basic stack operations are push, pop, & getTop. Beside these function, there are some more operations such as stack_initialization, stack_empty, stack_full. that can be implemented on a stack. stack_initialization function is simple constructor which prepares the stack for use and sets it to a vacant state.

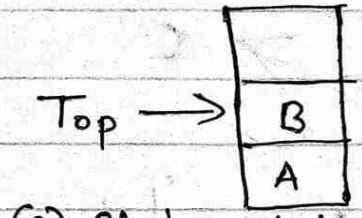
The stack_empty function checks whether the stack is empty. stack_empty function is used as safeguard against an attempt to pop an element from an empty stack. Popping an empty ~~cond~~ stack results in error. The stack_empty condition is also called as stack underflow.

The stack_full function checks whether the stack is full. stack_full function is useful as safeguard against an attempt push new element to a full stack. Pushing an element to a full stack results in error. Such ~~full~~ stack full condition is called stack overflow.

Get top is another stack operation which returns top element of the stack without actually popping it.

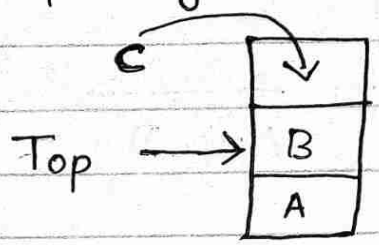
Push

Push operation inserts an element on the top of the stack. Recently added element is always at the top of the stack. Before every push we should check whether there is a room for new element

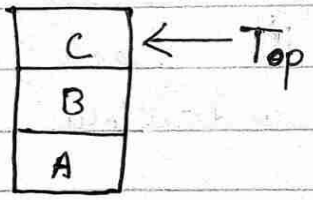


(a) Stack with two element

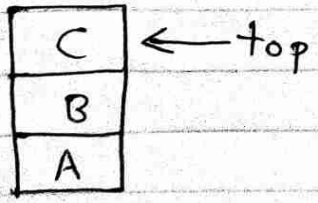
(b) pushing element to stack



(C) After push operation



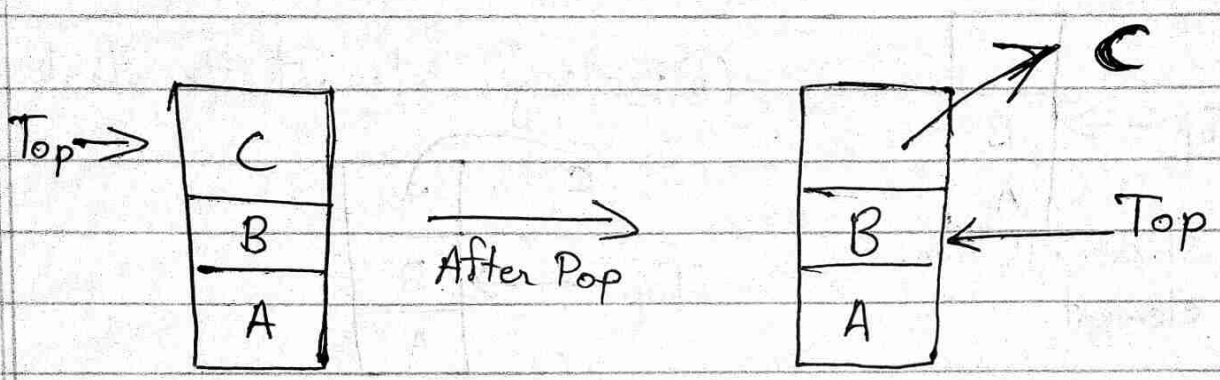
When there is no space to accommodate the new element on the stack, then the stack is said to be full. If the push is performed when the stack is full it is said to be in overflow state, ie; no element can be added when the stack is full. Push operation modifies the top since the newly inserted element becomes the topmost element.



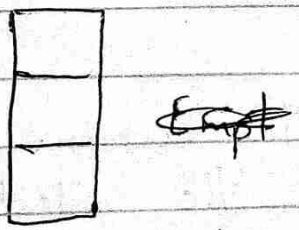
In this case stack is full since stack capacity is 3

Pop

The pop operation deletes an element from the top of the stack and returns the same to the user. It modifies the stack so that the next element becomes the top element



When there is no element available on the stack, the stack is ~~called~~ said to be empty. If pop performed when the stack is empty, then the stack is said to be in a condition called stack underflow.

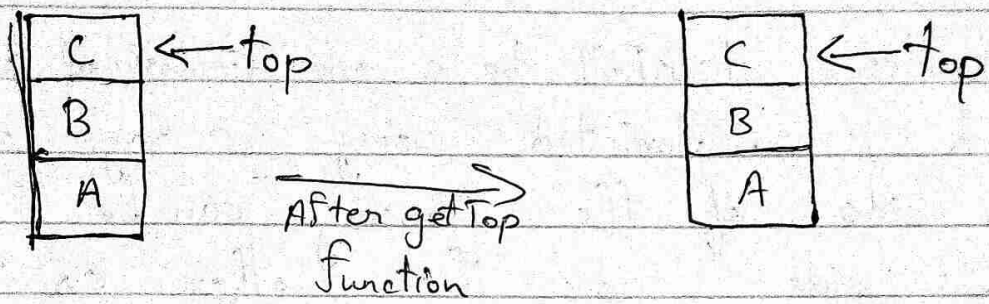


Empty stack

The pop should not be performed when the stack is empty, a hence before every pop it is necessary to ensure that the stack is not empty.

GetTop

The getTop operation gives information about topmost element and returns the element on the top of the stack. In this operation only a copy of the element which is at the top of the stack is returned.



Representation of Stack using Array

Stack can be implemented using both static data structure (eg: array) and dynamic data structure (linked list). The simplest way to represent a stack is by using a one-dimensional array. The stack implemented using an array is called a contiguous stack.

A stack is an ordered collection of elements. Hence it would be easy to manage a stack represented using an array. The only difficulty with an array is it's static memory allocations. Once declare, the size cannot be modified during run-time.

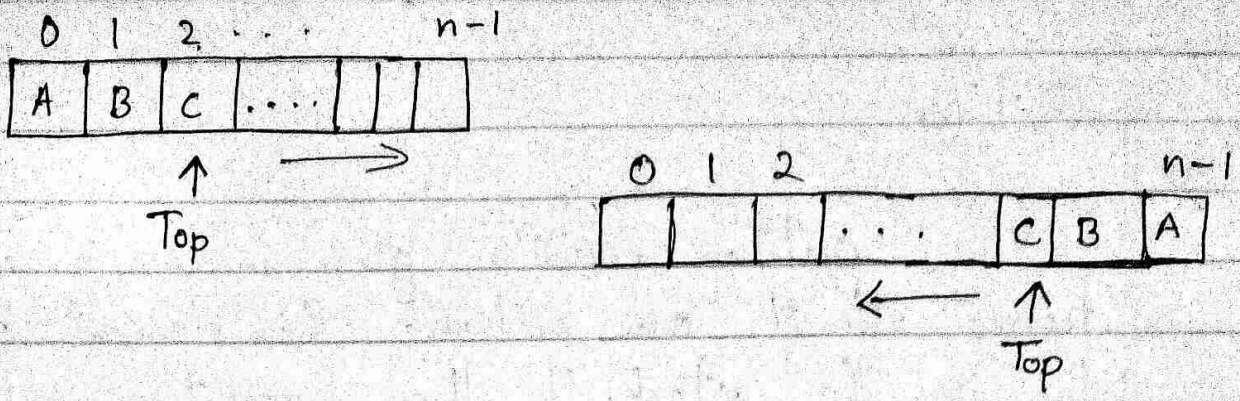


Figure: Stack representation using array

Consider stack[n] be a one-dimensional array. When the stack is implemented using arrays, one of the two sides of the array can be considered as the top side and the other as the bottom (lower) side as above figure. Top side is most commonly used. The elements are stored in the stack from the Ist location onwards. Ist element is stored at the 0th location of the array stack, which is at stack[0], IInd element is stored at stack[1], the i^{th} element at stack[i-1] & n^{th} element at stack[n-1].

Top is an integer variable associated with the array which points to the top element in the stack. Initial value of top is -1 when the stack is empty. It can hold the elements from index 0, & can grow to n-1 as maximum as this is static stack using arrays.

Code segment 1 gives the definition of class stack and lists the function prototypes of basic operations.

Code segment 1

```
class stacks
 {
    private:
        int stack[60];
        int max, top;
    public:
        stacks ()
        { max=50; top=-1;}
        int gettop();
        int pop();
        void push(int num);
        int empty();
        int isfull();
 };
```

The constructor must initialize the stack top. We cannot initialize it to one of the values in the range of 0 to ~~not~~ ~~be~~ max-1 because these are the indices of the stack array; Hence top is assigned -1. Each push operation increment top by one. When the element is added to empty stack top should be set to 0 as the new element will be ~~stor~~ stored at stack[0].

Empty

Empty is an operation that takes the stack as an argument, checks whether it is empty or not & returns the boolean value true or false.

Stack empty state can be examined by comparing the value of top with the value -1, because top = -1 represents an empty stack.

Code segment for empty()

```
if (top == -1)
    return 1; // Boolean value of true = 1
else
    return 0; // Boolean value of false = 0
```

Gettop

gettop operation checks for the stack empty state. If the ~~empty~~ stack is empty, it reports the "Stack underflow" error message. else returns a copy of the element that is at the top of the stack. Here top doesnt change as the element is not deleted from the stack. rather the element is still at top location.

code segment for gettop()

```
if (top == -1)
    cout << "Stack Empty (underflow)" << endl;
else
    return (stack[top]);
```

Push

Push operation inserts an element onto the stack of the maximum size (max). Element insertion is possible only if the stack is not full. The stack full state can be verified by comparing the top with max-1. If the stack is not full, the top is incremented by 1 & the element is added on the top of the stack.

code segment for push

```
if (top == max-1)
    cout<< "Stack full (overflow) \n";
else
{   top++; //incrementing top by one
    stack[top] = num; // add new element in new top position
}
```

Pop

Pop operation deletes the element at the top of the stack and returns the same. This is done only if the stack is not empty. If the stack is empty no deletion is possible. This is checked by the empty() function. If the stack is not empty, then the element at the top of the stack is returned & the top is decreased by one.

code segment for pop

```
if (top == -1)
    cout<< "Stack empty \n";
else
    return (stack[top--]);
```

Program code for all basic operations in stack

```
class stacks
 {
      int stack[50], max, top;
     public:
        stack() { max=50; top=-1;}
        int gettop();
        int pop();
        void push(int num);
        int empty();
        int isfull();
};
 int stacks::empty()
  {
     if (top==-1)
        return 1;
     else
        return 0;
  }

 int stacks::isfull()
 {
     if (top==max-1)
       return 1;
     else
        return 0;
 }
int stacks::pop()
 {
     if (empty()==1)
        cout<<"Stack is empty (underflow)\n";
     else
       return(stack[top--]); //equal to top--; stack[top];
 }
```

```
int stacks :: gettop()
{
    if(empty() != -1)
    if(empty() == 0)
        return (stack[top]);
    else
        cout<<" Stack is empty \n";
}

void Stacks :: push (int num)
{
    if (isfull() == 0)
     stack[top++] = num; //equal to top++; stack[top]
    else
     cout<<" Stack is full \n";
}

void main() //requires main() is linux
{
    Stacks s;
     s.pop();
     s.push(1);
     s.push(12);
     cout<<" Element at top of stack: "<<s.gettop()<<endl;
     cout<<" Popping element: "<<s.pop()<<endl;
     cout<<" Popping element: "<<s.pop()<<endl;
}
```

Output of the program

```
Stack is empty (underflow)
Element at top of stack: 12
Popping element: 12
Popping element: 1
```