1. Definition:

A stack can be considered as pile of items. It is characterized as ***LIFO*** (***L***ast ***I***n ***F***irst ***O***ut) property. A stack has following essential properties:

A). Stack has two components:
- **I.** A ***container of items*** that contains stack elements
- **II.** A pointer ***top*** that points the top most item.

B). There is a ***push*** operation. It means to add an item onto the stack. Only one item can be pushed at one time.

C). There is a ***pop*** operation. It means to remove an item from the stack. Only one item can be popped at one time.

D). Any ***modification*** can ***only*** be done ***at top***. It means that, either push or pop can only be done at top of the stack.

E). Only top most item is visible and accessible. No other item is accessible.

F). Due the above properties, the item which is pushed last will be popped first. Hence, a stack is known as ***LIFO*** (***L***ast ***I***n ***F***irst ***O***ut) structure.

NOTE: A stack can be implemented in two ways:
- Contiguous implementation, using arrays
- Linked implementation, using linked list and dynamic allocation

Here, we shall discuss only contiguous implementation. Linked implementation is discussed in later chapters.

2. Basic Operation:

1. Push operation:

Before any push operation, we must check for ***fullness*** of the stack. The maximum valid value of ***top*** pointer is ***MAX-1***. If ***top*** gets this value, the stack is full. It is also called ***stack overflow*** condition.

If there is space in the stack, the item will be pushed. The sequence of operations will be:

1. increment ***top***, so that it points to next ***empty*** location
2. insert item at this location.

2. Pop Operation:

Before any pop operation, we must check for ***emptiness*** of the stack. The minimum valid value of ***top*** pointer is ***-1***. If ***top*** gets this value, the stack is empty. It is also called ***stack underflow*** condition.

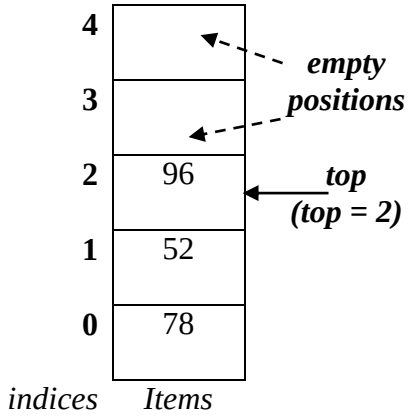
If there is an item in the stack, it can be popped. We need not specify which item is to be popped. **Item on the top will be popped always**. The sequence of operations will be:

1. get item from the location currently being pointed by ***top***
2. decrement ***top***.

The popped item is returned from the function.

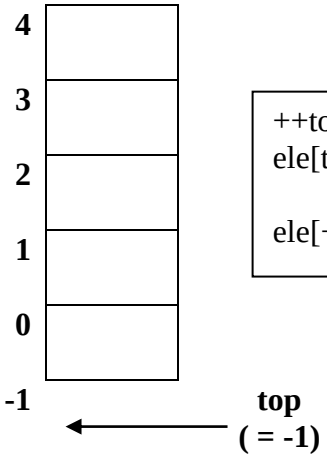
3. Example Operations:

A stack containing 3 items



Important points:
Maximum size = ***MAX***
Valid positions for items= from ***0*** to ***MAX-1***
Filled Positions = from ***0*** to ***top***
Empty Positions = from ***top+1*** to ***MAX-1***
top in empty stack = ***-1***

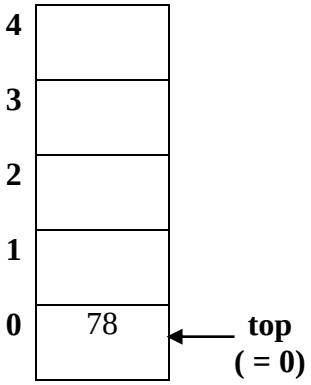
(1) An initial empty stack



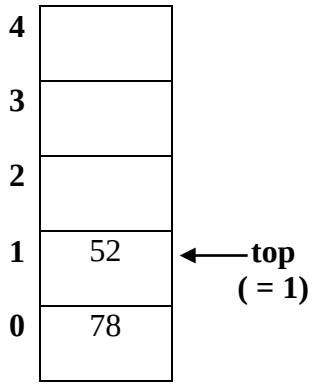

```
++top;
ele[top] = x;

ele[++top]=x
```

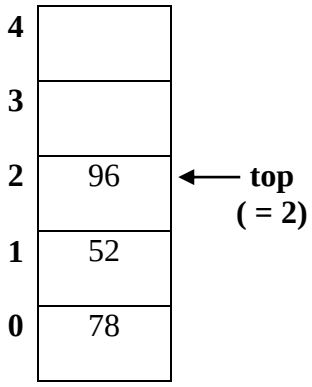
(2) Push item x = 78



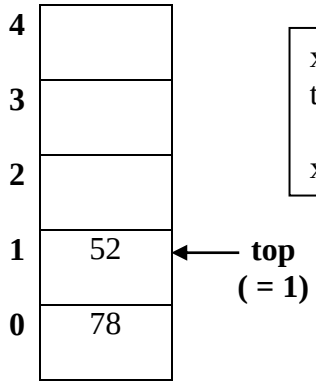
(3) Push item x = 52



(4) Push item x = 96



(5) Pop item & display it.

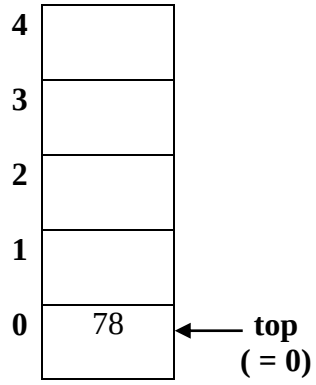



```
x = ele[top];
top- -;

x = ele[top- -];
```

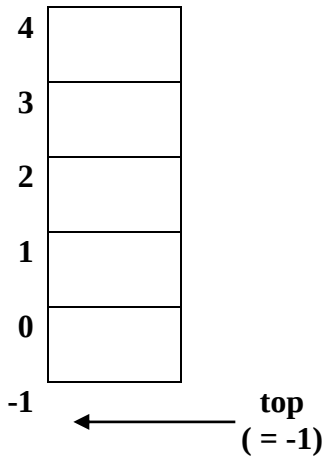
96 is popped, as it was on the top. It was pushed at the last, and popped first. ***(LIFO)***

(6) Pop item & display it.



52 is popped, as it was on the top.

(7)pop item & display it



78 is popped, as it was on the top.

Note: Note that the items 78, 52 & 96 are still there at there respective positions. But these positions are now considered as empty. Any push operation will overwrite these values.

Applications of Stack:

A stack is used where *last-in-first-out* property is required. There are three main applications:

A). during a function call
B). converting between polish notations
C). reversing a string

(A). When a function is called, the function which is called last will be completed first. It suits the property of stack. There is a memory area, specially reserved for this stack. As soon as, a function is called, a new entry is pushed on to the stack. This entry contains following information:

- status of calling function, to go back from the called function
- the space & values of local variables of the called function

As soon as, the called function ends, the status of its calling function is retrieved and the entry is popped from the stack.

(B). There are three type of notations:

- infix notation – operator is **between** the operands: a + b
- prefix notation – operator is **before** the operands: + a b
- postfix notations – operator is **after** the operands: a b +

We normally use infix notation. But, there are certain advantages of prefix & postfix notations. A stack is used to convert the expressions among the notations.

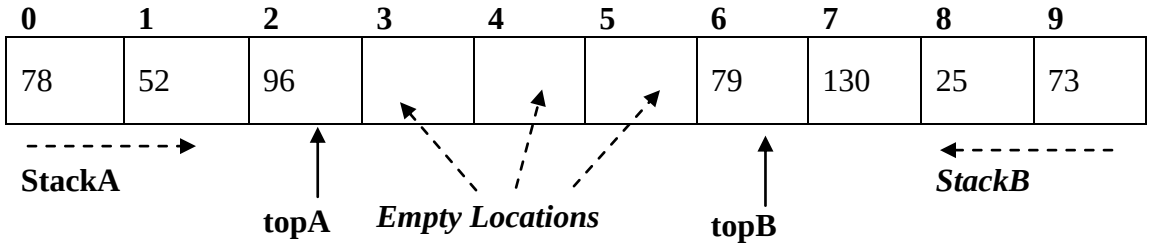
(C). A string can be reversed by using stack. The characters of string pushed on to the stack till the end of the string. The characters are popped and displays. Since, the end character of string is pushed at the last, it will be printed first.

Double Stack:

Sometimes, two stacks are maintained in one array. For efficiency considerations, both stacks grow in opposite directions. One form lower index to higher index; another from higher index to lower index.

Declarations: There is one array, and two top pointers ***topA*** and ***topB.***

Figure:



The empty locations can be occupied by any stack.

A full double stack: