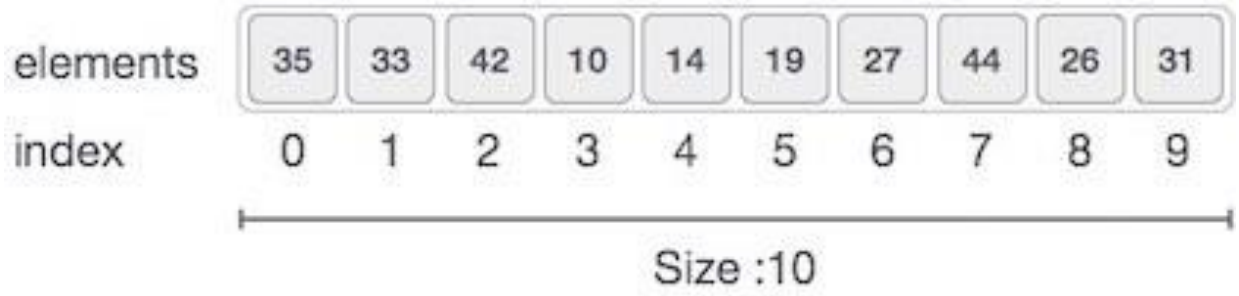
Linear Data structure

Prof. Viswanathan V
VIT Chennai

Array

- Array is a container which can hold a **fix number of items** and these items should be of the **same type**.
- Most of the data structures make use of arrays to implement their algorithms.
- Following are the important terms to understand the concept of Array.
  - **Element** − Each item stored in an array is called an element.
  - **Index** − Each location of an element in an array has a numerical index, which is used to identify the element.



Index starts with 0.
Array length is 10 which means it can store 10 elements.

Array - Basic Operations

Following are the basic operations supported by an array.

Traverse − print all the array elements one by one.

Insertion − Adds an element at the given index.

Deletion − Deletes an element at the given index.

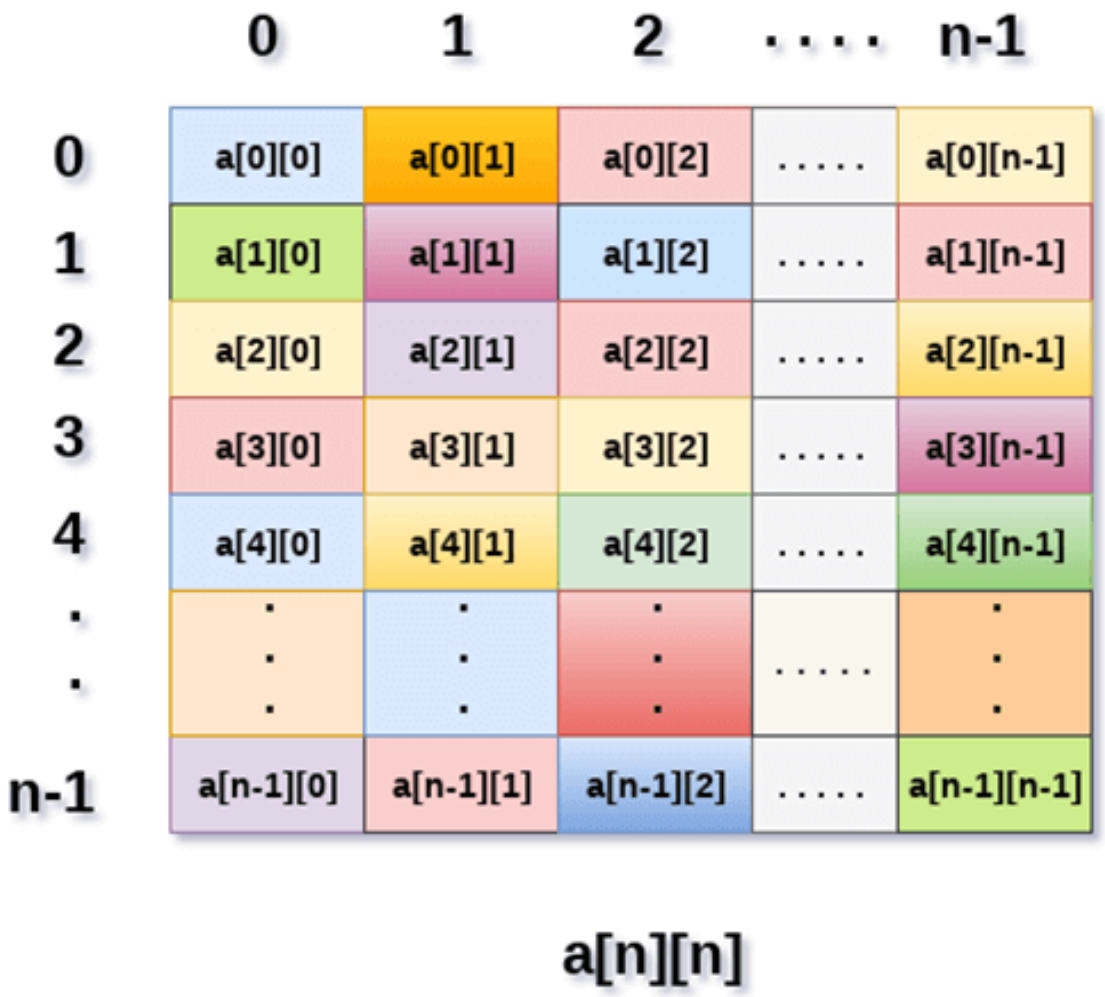
Search − Searches an element using the given index or by the value.

Update − Updates an element at the given index.

Array

- One dimensional array
- Multi dimensional array (array with more than two dimensions, two dimensions are represented **by rows** and **columns**)

2- D array



Declaration :

int x = a[10][15];

Accessing array:

```
for ( int i=0; i<n ;i++)
{
   for (int j=0; j<n; j++)
   {
      a[i][j] = 0;
   }
}
```

Array

Disadvantages

- The *number of elements* to be stored in an array should be *known in advance.*
- An array is a *static structure* (which means the array is of fixed size)
- Insertion and deletion are quite difficult in an array as the elements are stored *in consecutive memory locations and the shifting operation is costly*.

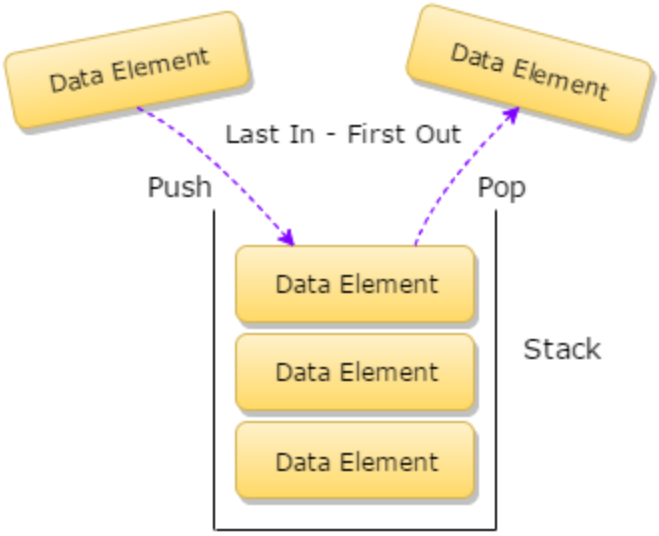
Applications

- Maintains multiple variable names using a *single name*.
- Arrays help to maintain *large data under a single variable name*. This *avoid the confusion* of using multiple variables.
- Arrays can be used for *sorting data elements*.

Stack

- A stack is a linear data structure in which all the insertion and deletion of data at one end only called Top, rather than in the middle.

- Stacks can be implemented by using arrays of type linear.



Stack – Basic Operations

Stack operations may involve initializing the stack, using it and then de-initializing it.

Apart from these basic stuffs, a stack is used for the following two primary operations −

push() − Pushing (storing) an element on the stack.

pop() − Removing an element from the stack.

peek() − get the top data element of the stack, without removing it.

isFull() − check if stack is full.

isEmpty() − check if stack is empty.

PUSH operation

The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps −

Step 1 − Checks if the stack is full.

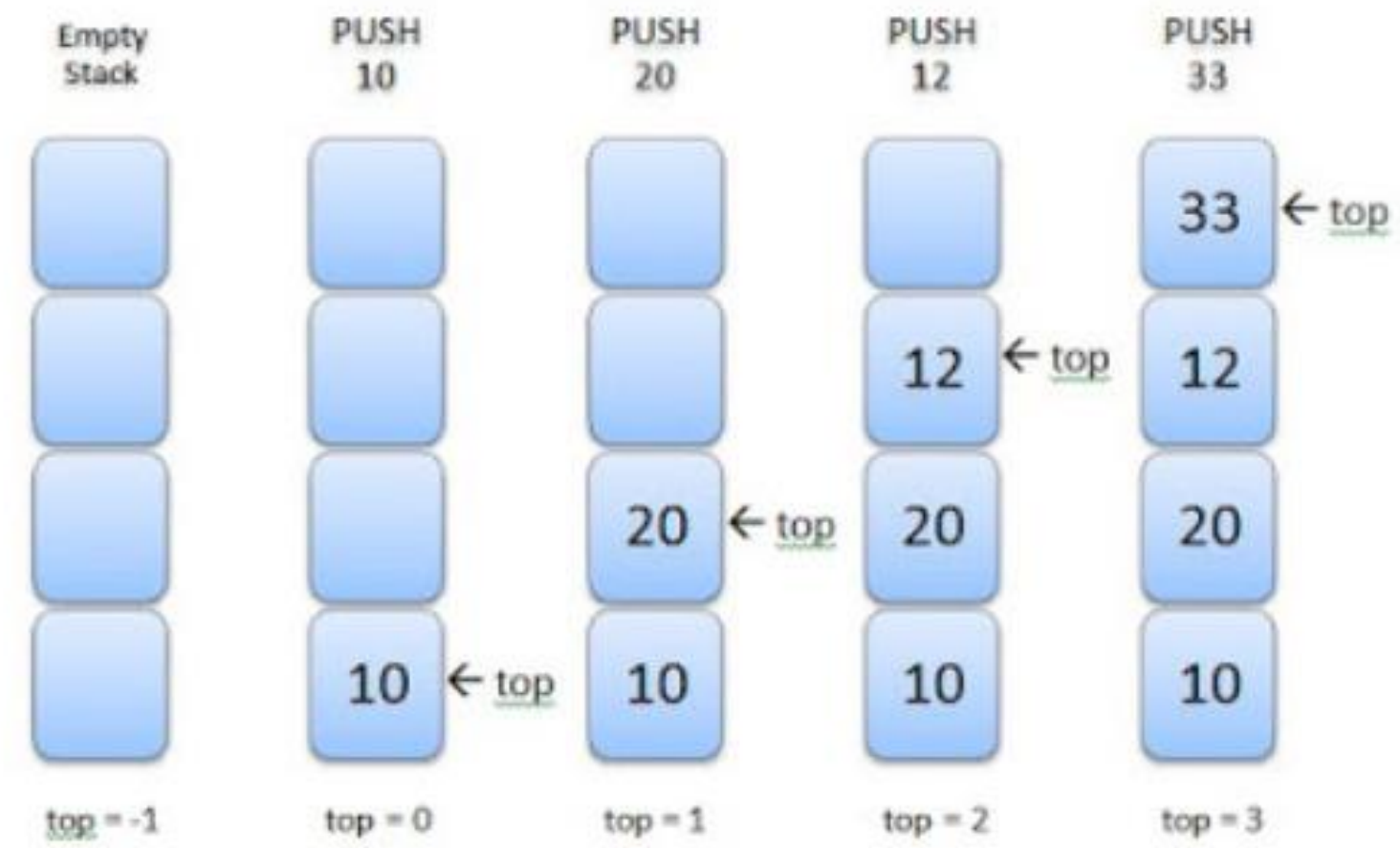
Step 2 − If the stack is full, produces an error and exit.

Step 3 − If the stack is not full, increments **top** to point next empty space.

Step 4 − Adds data element to the stack location, where top is pointing.

Step 5 − Returns True.

PUSH operation



```
Algorithm  isfull()   // procedure to check stack is full
{  //MAXSIZE – size of the stack
   if(top == MAXSIZE)
              return true
   else
               return false
}
```

```
Algorithm isempty() //procedure to check stack empty
{
    if(top == -1)
              return true
     else
          return false
    }
```

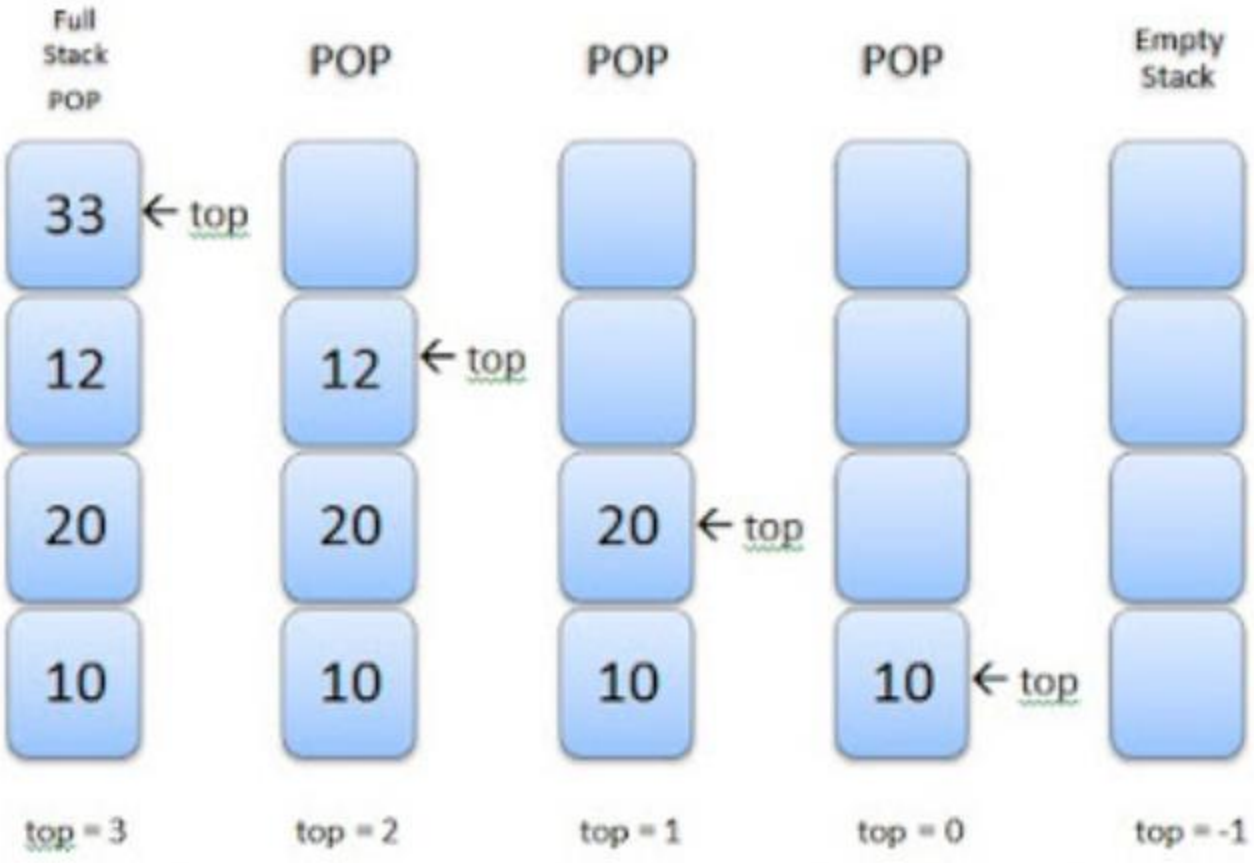
Algorithm to insert an element into the stack (PUSH)

```
Algorithm push(data)
{
    if (!isFull())
     {
          top = top + 1
          stack[top] = data
      }
     else
         print("Could not insert data, Stack is full.")
 }
```

POP operation

- A Pop operation may involve the following steps −
- **Step 1** − Checks if the stack is empty.
- **Step 2** − If the stack is empty, produces an error and exit.
- **Step 3** − If the stack is not empty, accesses the data element at which **top** is pointing.
- **Step 4** − Decreases the value of top by 1.
- **Step 5** − Returns True.

POP operation



Algorithm to remove an item from stack (POP)

```
Algorithm pop()
{
    if(!isempty())
{
     data = stack[top]
     top = top – 1
      return data
 }
else
    print("Could not retrieve data, Stack is empty")
 }
```

Application of Stack
(Evaluation of Postfix Expression)

- **infix** notation, e.g. a - b + c, where operators are used **in**-between operands.
- Operator is written ahead of operands.

  +ab. This is equivalent to its infix notation **a + b**.

  Prefix notation is also known as **Polish Notation**.
- Operator is written after the operands.

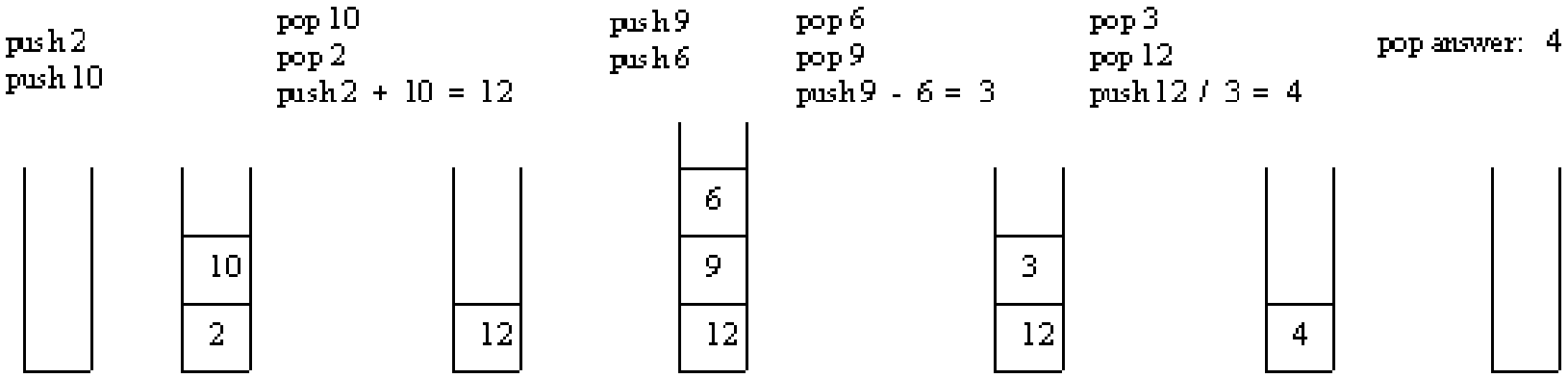
  For example, **ab+**. This is equivalent to its infix notation **a + b**.

S.No.	Operator	Precedence
1	Exponentiation ^	Highest
2	Multiplication (∗) & Division (/)	Second Highest
3	Addition (+) & Subtraction (−)	Lowest

Application of Stack
(Evaluation of Postfix Expression)

- Create an empty stack and start scanning the postfix expression from left to right.
- If the element is an operand, push it into the stack.
- If the element is an operator **O**, pop twice and get A and B respectively. Calculate B**O**A and push it back to the stack.
- When the expression is ended, the value in the stack is the final answer.
- Evaluation of a postfix expression using a stack is explained in below example:

2 10 + 9 6 - /



Infix to Postfix conversion

Conversion from infix to postfix:

Procedure to convert from infix expression to postfix expression is as follows:

1. 1. Scan the infix expression from left to right.
2. If the scanned symbol is left parenthesis, push it onto the stack.
3. If the scanned symbol is an operand, then place directly in the postfix expression (output).
4. If the symbol scanned is a right parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.
5. If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (or greater than or equal) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.

Convert ((A – (B + C)) * D) ↑ (E + F) infix expression to postfix form:

SYMBOL	POSTFIX STRING	STACK	REMARKS
(		(	
(		((	
A	A	((	
-	A	((-	
(	A	((- (	
B	A B	((- (	
+	A B	((- (+	
C	A B C	((- (+	
)	A B C +	((-	
)	A B C + -	(	
*	A B C + -	(*	
D	A B C + - D	(*	
)	A B C + - D *		
↑	A B C + - D *	↑	
(	A B C + - D *	↑ (	
E	A B C + - D * E	↑ (	
+	A B C + - D * E	↑ (+	
F	A B C + - D * E F	↑ (+	
)	A B C + - D * E F +	↑	
End of string	A B C + - D * E F + ↑	The input is now empty. Pop the output symbols from the stack until it is empty.	


Thank
you