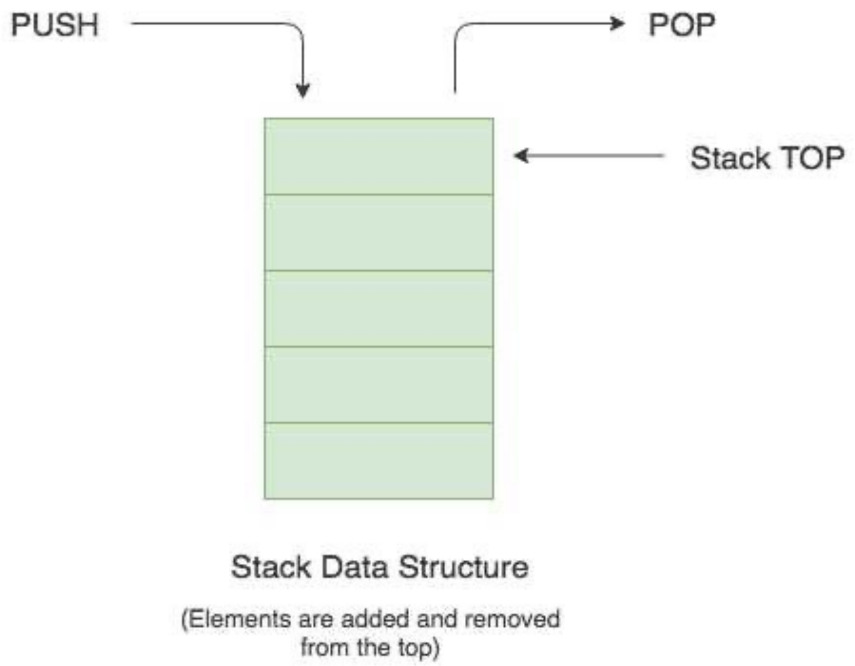
Stack

- **Stack** is a non-primitive linear data structure which stores data in **LIFO** manner where **LIFO** stands for **Last-In-First-Out**.
- In stack, insertion of new value and deletion of existing value is done from same position called **top** of stack.



Stack

incapp

Implementation of Stack:

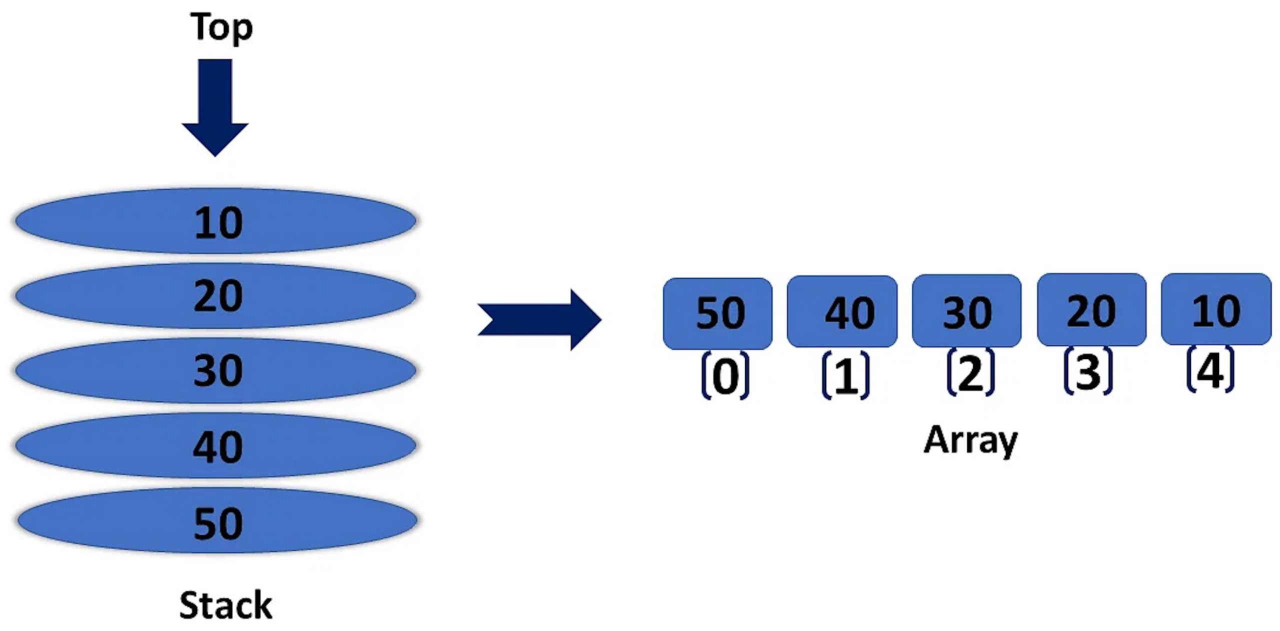
1. **Array Stack:** Stack implemented using array is called array stack.
2. **Linked Stack:** Stack implemented using linked list is called linked stack.

Operations on Stack:

1. **Push:** Adding new value to the top of stack.
2. **Pop:** Removing the existing value from top of stack.
3. **Traverse:** Visiting every value of stack from top to first.

Array Stack

Stack can be implemented using array. In this, we will define an array and new value must be added at the end of array and existing value must be deleted from end. In this implementation of array value can't be added or deleted other than end of array.



Array Stack

Algo to Push in Array Stack

Here **stack** is the array that will act as stack, **top** is initially having value **-1** and will store the index of the last value pushed into the stack, **capacity** will specify the maximum values stack can store, and **value** will store the value to be pushed into stack, and then the algorithm is given below:

```
1. Check, if top == capacity – 1, then:
     a. Print, stack is full.
2. Otherwise,
     a. Read value.
     b. Set top = top + 1.
     c. Set stack[top] = value.
3. Exit.
```

Array Stack

Algo to Traverse the Array Stack

Here **stack** is the array that is acting as stack, **top** is having the index of last value pushed in array stack, **i** will store the index of the element to be traversed one by one in stack, and then the algorithm is given below:

```
1.  Check, if top != -1, then:
      a.  Repeat for i = top to 0
      b.  Print, stack[i].
2.  Otherwise,
      a.  Print, no value to traverse.
3.  Exit.
```

Array Stack

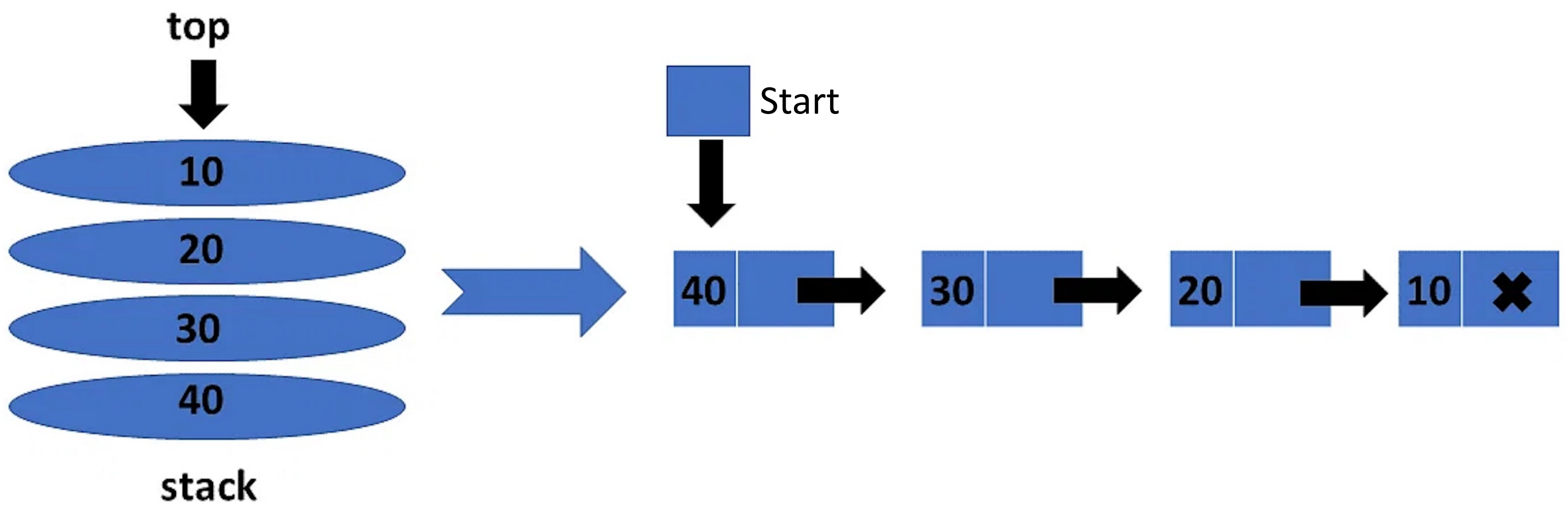
Algo to Pop from Array Stack

Here **stack** is the array that is acting as stack, **top** is having the index of last value pushed in array stack, **value** will store the value to be popped from stack, and then the algorithm is given below:

```
1.  Check, if top != -1, then:
      a.  Set value = stack[top].
      b.  Set top = top – 1.
      c.  Print, value is popped from stack.
2.  Otherwise,
      a.  Print, stack is empty.
3.  Exit.
```

Linked Stack

Stack can be implemented using linked list. In this, new node must be added at begin of linked list and existing node must be deleted from begin of linked list. In this implementation of linked list, node can’t be added or deleted other than begin of linked list.



Linked Stack

Algo to Push in Linked Stack

Here **top** is node pointer initially having value **NULL** and will point to the last node, **newNode** is the newly created node and **value** will store the value to be stored in the data part of the newly created node to be pushed in linked stack, then the algorithm is given below:

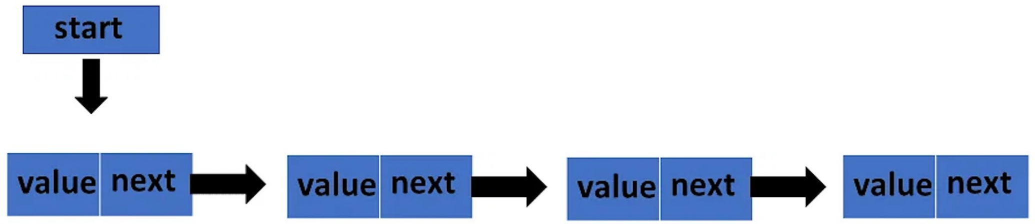
```
1. Create newNode with given value.
2. Set newNode -> next = top.
3. Set top = newNode.
4. Exit.
```

Linked Stack

Algo to Traverse the Linked Stack

Here **top** is node pointer pointing to the last node, **currentNode** is also the node pointer that will point from last node to first node one by one to be visited in linked stack, then the algorithm is given below:

```
1. Check, if top != NULL, then:
    a. Set currentNode = top.
    b. Repeat while currentNode != NULL
        i.  Print, currentNode ->data.
        ii. Set currentNode = currentNode ->next.
2. Otherwise,
    a. Print, no value to traverse in stack.
3. Exit.
```



Linked Stack

Algo to Pop from Linked Stack

Here **top** is node pointer pointing to the last node, **value** will store the value of node to be to be popped from linked stack, then the algorithm is given below:

```
1. Check, if top != NULL, then:
    a. Set value = top ->data.
    b. Set top = top->next.
    c. Print, node of value popped from stack.
2. Otherwise,
    a. Print, memory is not available.
3. Exit.
```