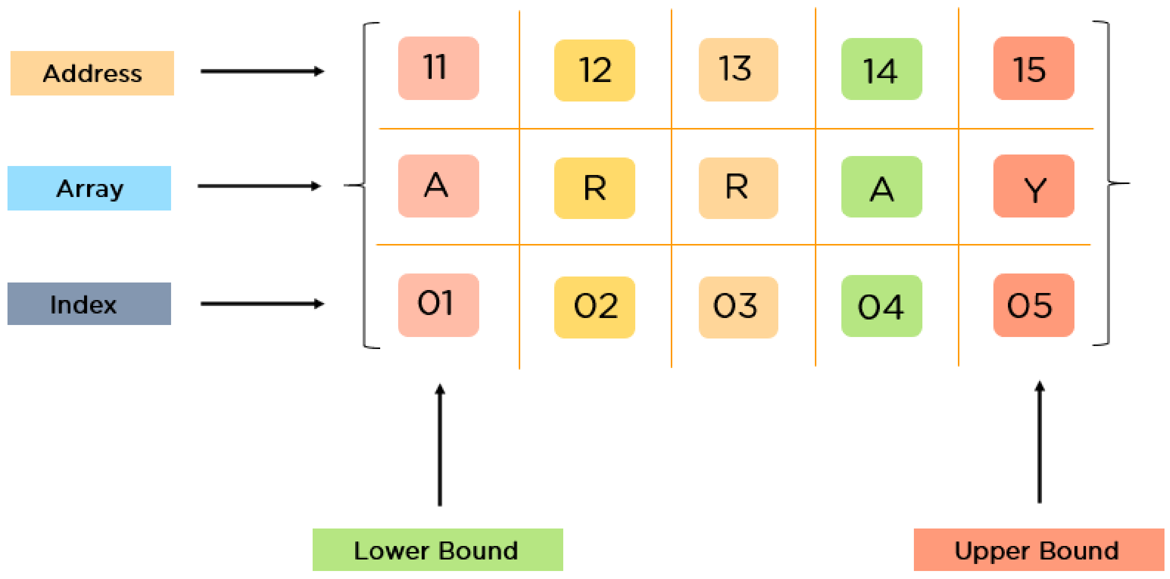
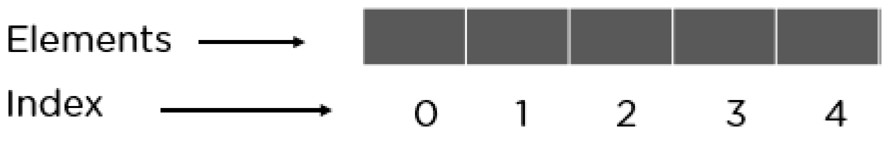
Data Structure Manual (1)

Array

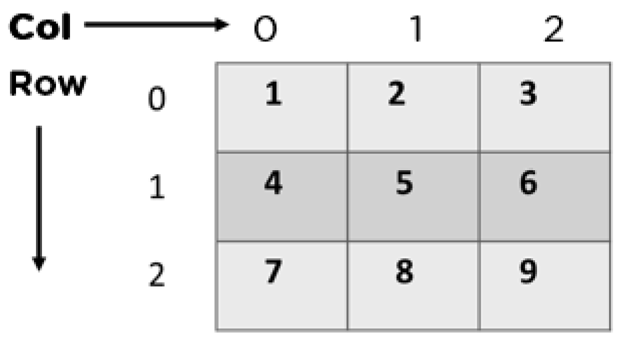


An array is a linear data structure that collects elements of the same data type and stores them in contiguous and adjacent memory locations. Arrays work on an index system starting from 0 to (n-1), where n is the size of the array.

One-Dimensional Arrays



Multi-Dimensional Arrays



Initialize Array in C

```
// Method 1
int arr[6] = {2, 3, 5, 7, 11, 13};

// Method 2
int arr[]= {2, 3, 5, 7, 11};

// Method 3
int arr[5];
arr[0]=1;
arr[1]=2;
arr[2]=3;
arr[3]=4;
arr[4]=5;
```

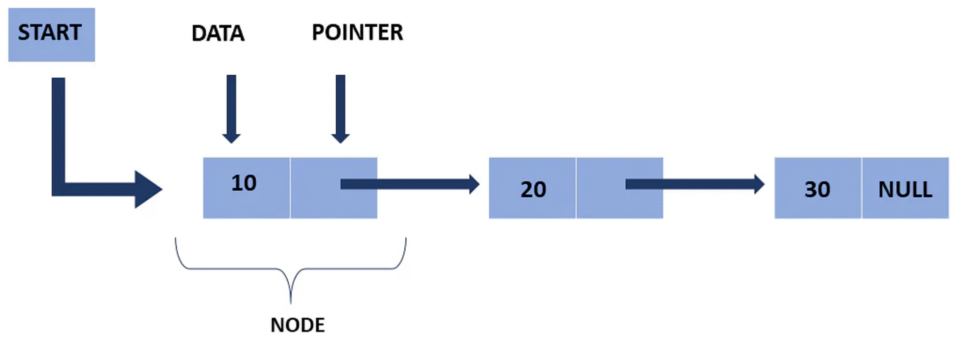
Initialize Two-Dimensional Array in C

```
// Method 1
int arr[4][3]={10,20,30,40,50,60,20,80,90,100,110,120};

// Method 2
int arr[4][3]={{10,20,30},{40.50,60},{70,80,90},{100,110,120}};
```

Printing Two-Dimensional Array

```
int arr[4][3]={{10,20,30},{40.50,60},{70,80,90},{100,110,120}};

for(int i = 0; i < 4; i++)
{
  for(int j = 0; j < 3; j++)
  {
    printf(" %d", arr[i][j]);
  }
}
```

Linked List



A linked list is a linear data structure that stores a collection of data elements dynamically.

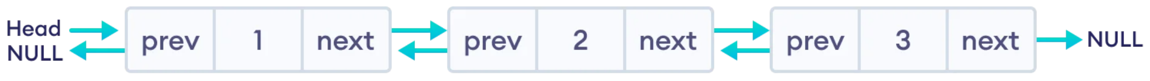
This representation of a linked list depicts that each node consists of two fields. The first field consists of data, and the second field consists of pointers that point to another node

Simple Structure of Linked List

```
struct node
{
  int data;
  // singly Linked List
  struct node *next;
};


// implement 3 nodes ( 1 -> 2 -> 3 )
int main()
{
  struct node* n1;
  struct node* n2;
  struct node* n3;

  n1 = (struct node*)malloc(sizeof(struct node));
  n2 = (struct node*)malloc(sizeof(struct node));
  n3 = (struct node*)malloc(sizeof(struct node));

  n1->data = 1;
  n1->next = n2;

  n2->data = 2;
  n2->next = n3;

  n3->data = 3;
  n3->next = NULL;
}
```

Singly Linked List Time Complexity

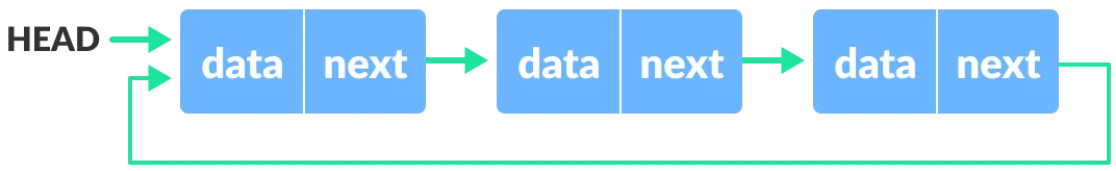
Doubly Linked List



A doubly linked list is a type of linked list in which each node consists of 3 components:

- `prev` - address of the previous node
- `data` - data item
- `next` - address of next node

Circular Linked List

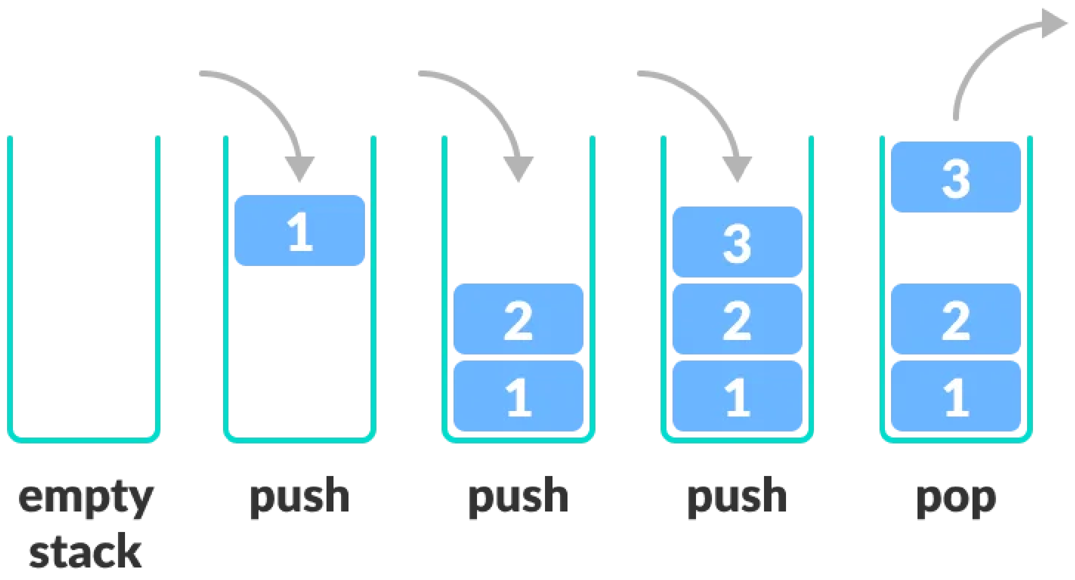


A circular linked list can be either singly linked or doubly linked.

- for singly linked list, next pointer of last item points to the first item
- In the doubly linked list, pointer of the first item points to the last item as well.

  prev

Stack



A stack is a linear data structure that follows the principle of **Last In First Out (LIFO).**

This means the last element inserted inside the stack is removed first.

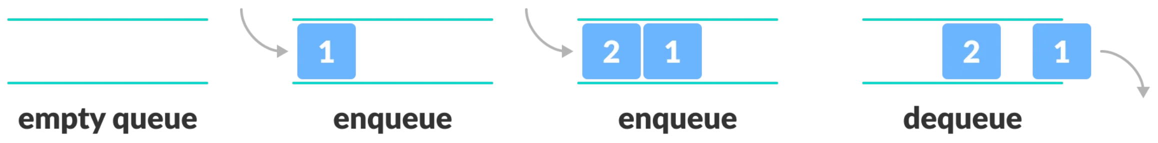
Basic Stack Operations

- **Push**: Add an element to the top of a stack
- **Pop**: Remove an element from the top of a stack
- **IsEmpty**: Check if the stack is empty
- **IsFull**: Check if the stack is full
- **Peek**: Get the value of the top element without removing it

Stack Time Complexity

peek()	O(1)

Queue



A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

Queue follows the **First In First Out (FIFO).**

The item that goes in first is the item that comes out first.

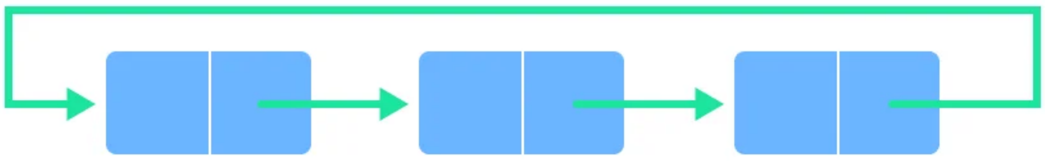
Basic Queue Operations

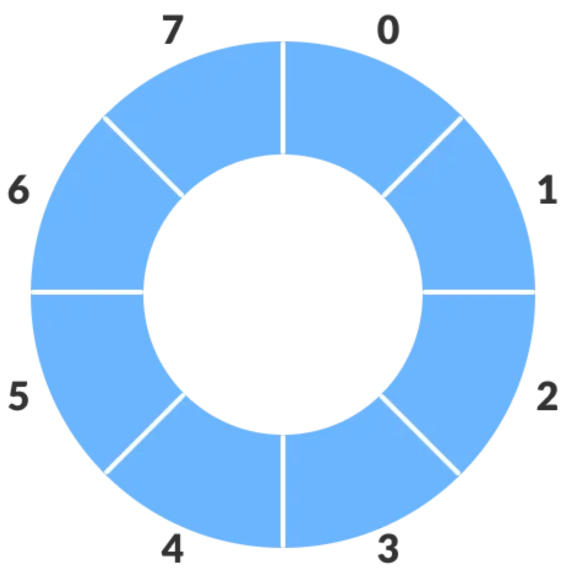
- **Enqueue**: Add an element to the end of the queue
- **Dequeue**: Remove an element from the front of the queue
- **IsEmpty**: Check if the queue is empty
- **IsFull**: Check if the queue is full
- **Peek**: Get the value of the front of the queue without removing it

Four Types of Queue

- Simple Queue
- Circular Queue*
- Priority Queue
- Double Ended Queue

Circular Queue





A circular queue is the extended version of a regular queue where the last element is connected to the first element. Thus forming a circle-like structure.

Circular Queue Works

Circular Queue works by the process of circular increment.

When we try to increment the pointer and we reach the end of the queue, we start from the beginning of the queue.

Circular increment is performed by modulo division with the queue size.

`REAR = (REAR + 1) % QUEUE_SIZE = 0 (start of queue)`