Linear Data Structure: Array and Linked List:

Array is a contiguous storage of homogenous data. We use array for fetching random elements and to avoid memory overflow / shortage of memory.

Problems in Array:

1. Array have fixed size.
2. Complicated Insertion and Deletion.

Applications of Arrays:

1. Online ticket systems use arrays to represent tickets or seats.
2. Used to create the leader-board of a game to track the score and rank of each player.
3. Two-dimensional arrays are used in image processing, speech processing etc.

Implementation:

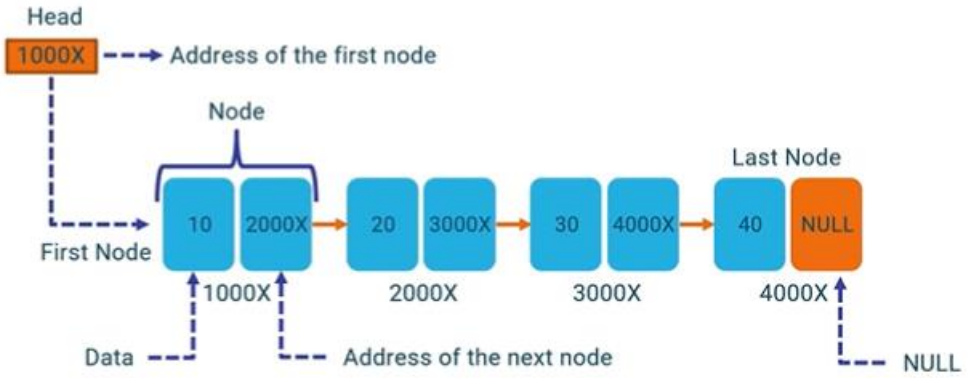
One-Dimensional Array

```
#include<iostream>
using namespace std;

int main()  {
  int arr[50],n,idx ;
  cout<<"Enter the size of array:"<<endl;
  cin>>n;
  cout<<"Enter the elements of array:"<<endl;
  for(idx=0;idx<n;idx++) {
     cin>>arr[idx];
  }
  cout<<"Array elements are:"<<endl;
  for(idx=0;idx<n;idx++) {
     cout<<arr[idx]<<" ";
  }
   return 0;
}
```

Two-Dimensional Array

```
#include<iostream>
using namespace std;
int main() {
  int arr[3][2];
  cout<<"Enter the elements of array:"<<endl;
  for(int row=0;row<3;row++) {
    for (int col = 0; col < 2;col++) {
      cin>>arr[row][col];
    } }

cout << "Elements in the array are:\n";
  for (int row = 0; row < 3; row++) {
    for (int col = 0; col < 2; col++) {
      cout <<arr[row][col]<<"  ";
    } }
return 0;
}
```

Linked Lists Representation:



Advantages of a Linked List:

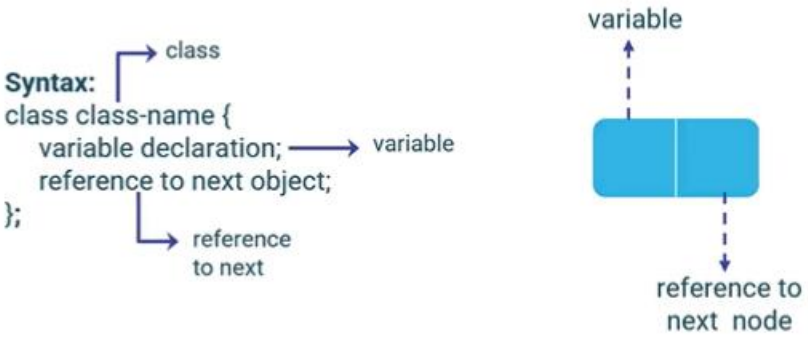
1. It can grow or shrink dynamically to any size.

2. More efficient Insert and Delete operations.
3. No memory wastage.

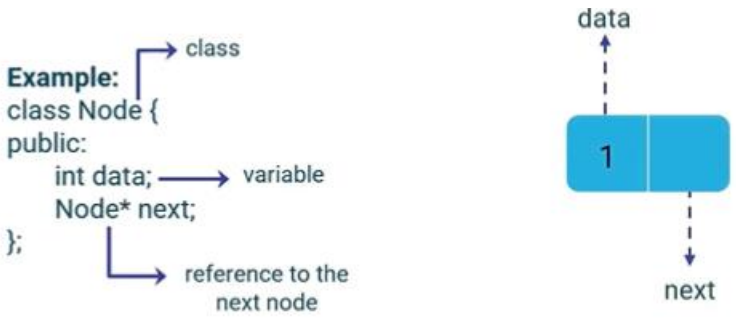
Disadvantages of a Linked List:

1. Required more memory.
2. Traversal is difficult.
3. Reverse traversal is difficult.

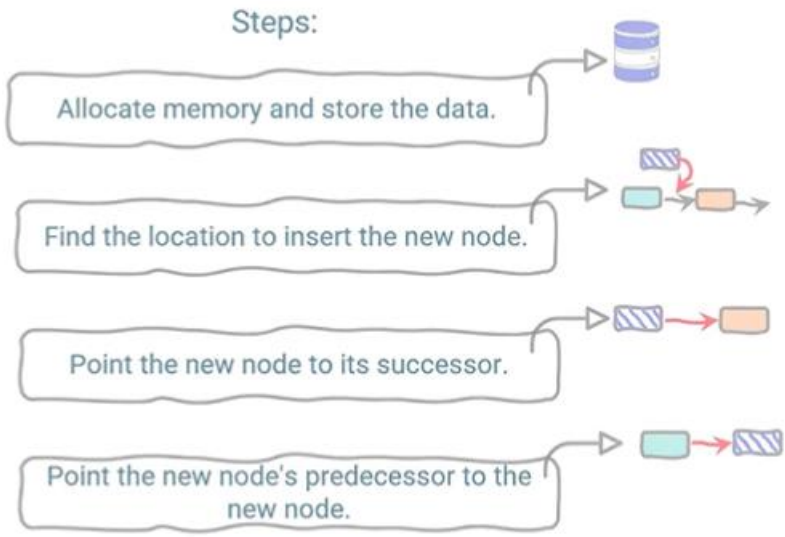
Implementation of a Linked List:



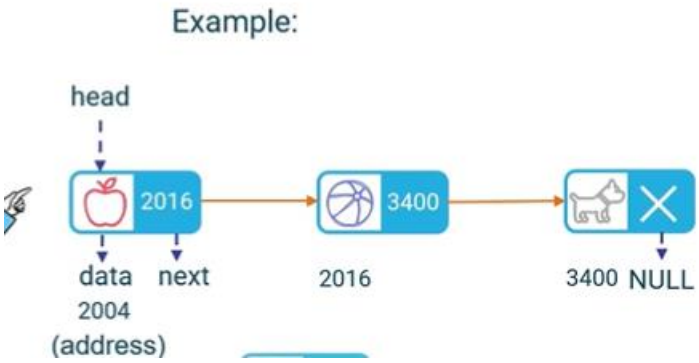
Example:



Insert a New Node To The List:



Example:



Insert a Node At Front - Singly Linked List:

```
#include <iostream>
using namespace std;

class Node {
    public:
        int data;
        Node* next;
};

class LinkedList {
  public:
     Node *head,*tail;
       LinkedList()  {
            head = NULL;
             tail = NULL;
        }
```

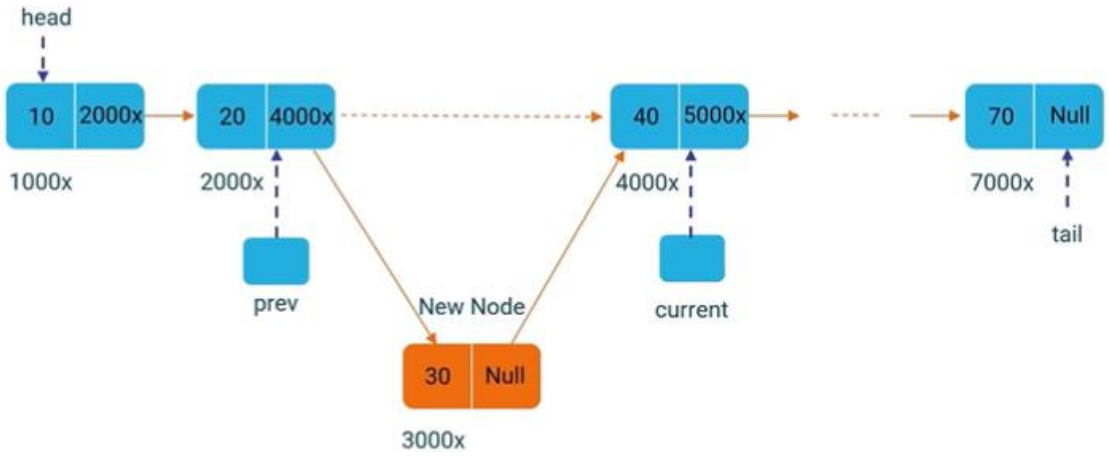
```
void insertNodeAtFront(int value)  {
    Node *newNode= new Node;
    newNode->data = value;
    newNode->next = NULL;
    if(head == NULL) {
        head = newNode;
        tail = newNode;
    }

    else {
        newNode->next=head;
        head=newNode;
    }
}
};  //class LinkedList ends here

int main() {
   LinkedList lst;
   lst.insertNodeAtFront(40);
   lst.insertNodeAtFront(30);
   lst.insertNodeAtFront(20);
   lst.insertNodeAtFront(10);
   return 0;
}
```

Allocating memory to the new node and storing data into it

Insert a Node at the End:

```
void insertNodeAtEnd(int value)  {

   Node  *newNode = new  Node;
   newNode->data = value;
   newNode->next = NULL;

   if (head== NULL)     {
     head = newNode;
     tail=head;
      return;
  }

  else{
        tail->next = newNode;
        tail=tail->next;
        return;
     }
  }
```

Insert a Node at a Specific Position:




```
void insertAtPosition(int pos, int value) {
   Node *prev=new Node;
   Node *current=new Node;
   current=head;
   Node *newNode=new Node;
   newNode->data=value;
   newNode->next=NULL;

     if(pos < 1) {
       cout<< "Position cannot be less than one.";
     } else if(pos == 1){
        newNode->next = head;
        head = newNode;


  } else {
       for(int i=1; i<pos ; i++) {
            prev=current;
            current=current->next;
            if(current == NULL) {
               cout<< "Invalid position";
               return;
       } }
      prev->next=newNode;
      newNode->next=current;   } }
```

Search and Display Elements in the List:

```
void search(int value)    {
   Node *current=head;
   while(current!=NULL)   {
        if(current->data==value)  {
            cout<<"Element"<<value<<" is found ";
             return;
       }
       current=current->next;
   }
   cout<<"Element "<<value<<" not found in the List";
   }
}
```

```
void displayList()
{
      Node *current=head;
      while (current != NULL) {
            cout<< current->data <<" ";
            current = current->next;
       }
   }
```

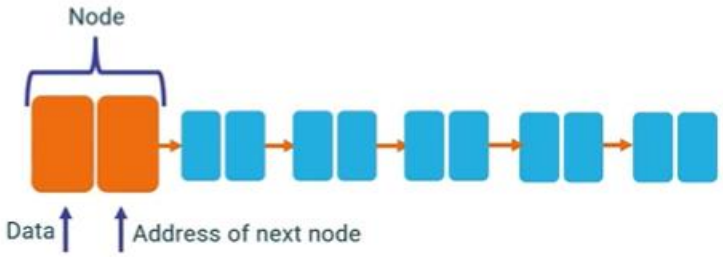
Delete a Node from the List:

```
void deleteNode(int value)  {
   bool flag=false;
   Node *current= new Node;
   Node *previous=new Node;
   previous=head;
   current=head;
   while (current!=NULL)  {
     if(current->data==value and current==head){
       head=current->next;
       free(current);
       flag=true;
       break;   }
else if(current->data==value) {
        previous->next=current->next;
        if(current==tail){
           tail=previous;
        }
```

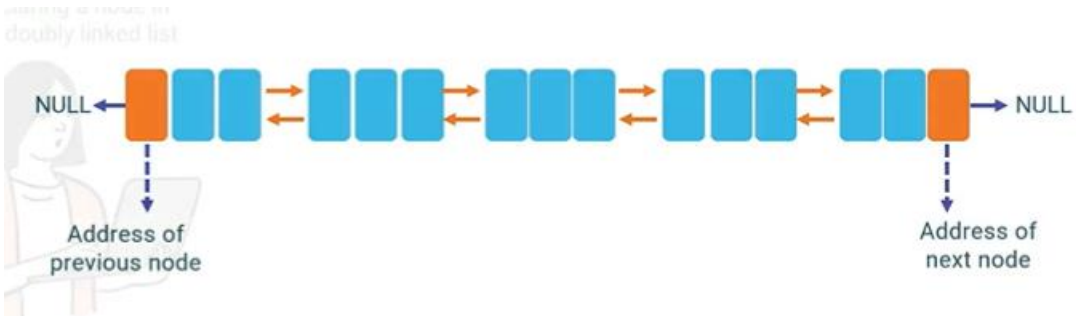
```
free(current);
 flag=true;
 break;
 } else {
    previous=current;
    current=current->next;
  }
}
if(flag==true)
      cout<<"Element deleted";
  else
       cout<<"Element not found";
}
```

```
int main() {
   LinkedList lst;
   lst.insertNodeAtFront(10);
   lst.insertNodeAtEnd(40);
   lst.insertNodeAtEnd(50);
   lst.insertAtPosition(2,20);
    lst.insertAtPosition(3,30);
    lst.search(30);
    lst.deleteNode(30);
    lst.displayList();
    return 0;
}
```

Singly Linked List:



Doubly Linked List:



Advantages of DLL:

1. Ease of Traversal.
2. More efficient Insert, Delete and Search Operations.

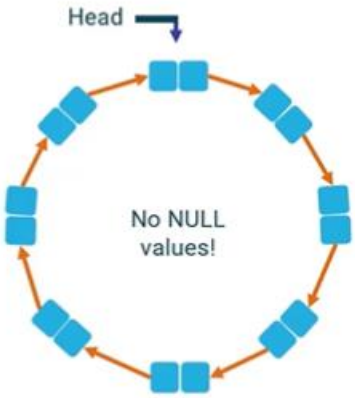
Disadvantages of DLL:

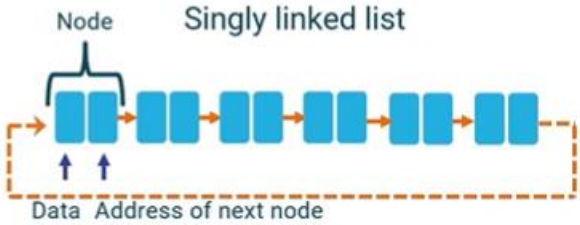
1. More space.
2. Deletion, Insertion - The number of modifications increases.

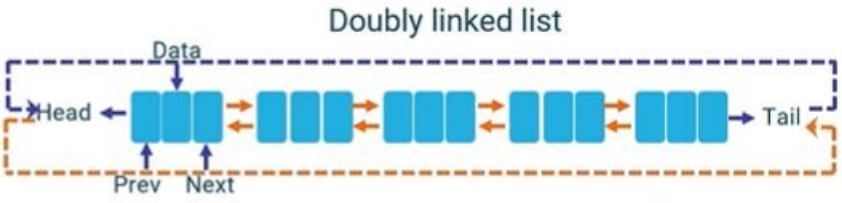
Implementation of DLL:

```
class DLLNode {
  public:
    int data ;
    DLLNode *prev;
    DLLNode *next;
};
```

Circular Linked List:







Advantages of Circular Linked List:

1. Ease of Traversal.
2. Best to use when we want to traverse in loop.

Disadvantages of Circular Linked List:

1. Infinite loop
2. Operations are complex

Applications of Linked List:

1. Not sure about the number of elements.
2. Used in browser cache with BACK-FORWARD visited pages, images, traverse browser history etc.
3. The undo and redo functionalities in Word, Paint, and other software.
4. To track various information in a circular fashion. E.g. To keep track of players turns in multi player games.

Summary:

- A linear list is an ordered collection of values. Arrays and linked lists are linear list.
- An array is a collection of homogenous elements stored contiguously and its size is fixed.
- Insertion and deletion operations are a bit difficult in arrays because of shifting elements.
- A linked list is a collection of connected elements called nodes, arranged in a linear sequence.
- The size of a Linked List is not fixe. It can grow or shrink dynamically.
- A singly linked list stores data and the address of the next node.
- A doubly linked list contains memory area to store the data, the address of the previous node and the address of the next node.
- A circular list is a list in which the link field ("next") of the last node refers to the first node of the list.