Lab Work 8: Linked List

Learning Objectives

- **Basic concepts of linked list**
- **How to create a 3-node link list**
- **How to print out your link list from any node**
- **How to add a node**
- **How to delete a node**

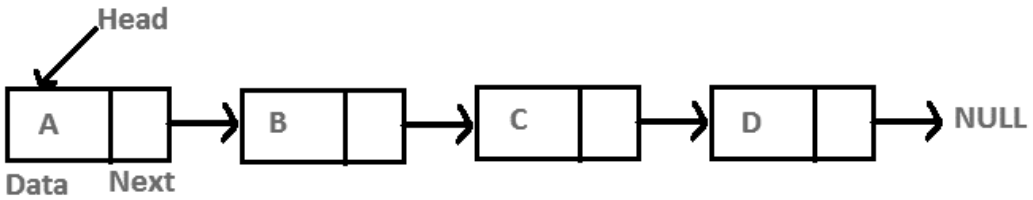
Linked Lists are linear or sequential data structures in which elements are stored at non-contiguous memory locations and are linked to each other using pointers.

Like arrays, linked lists are also linear data structures but in linked lists elements are not stored at contiguous memory locations. They can be stored anywhere in the memory but for sequential access, the nodes are linked to each other using pointers.

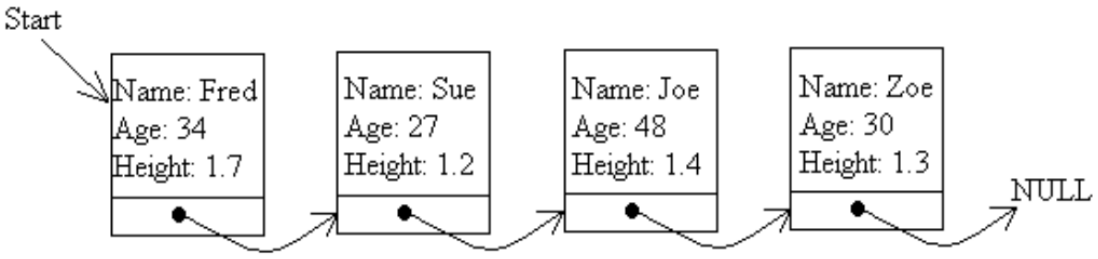
Each element in a linked list consists of two parts:

Data: This part stores the data value of the node. That is the information to be stored at the current node.

Next Pointer: This is the pointer variable or any other variable which stores the address of the next node in the memory



Another example



Advantages of Linked Lists over Arrays: Arrays can be used to store linear data of similar types, but arrays have the following limitations:

The size of the arrays is fixed, so we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage. On the other hand, linked lists are

dynamic and the size of the linked list can be incremented or decremented during runtime.

Inserting a new element in an array of elements is expensive, because a room has to be created for the new elements, and to create room, existing elements have to shift.

For example, in a system, if we maintain a sorted list of IDs in an array id[].

id[] = [1000, 1010, 1050, 2000, 2040].

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000). Deletion is also expensive with arrays unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

On the other hand, nodes in linked lists can be inserted or deleted without any shift operation and is efficient than that of arrays.

Disadvantages of Linked Lists:

Random access is not allowed in Linked Lists. We have to access elements sequentially starting from the first node. So, we cannot do a binary search with linked lists efficiently with its default implementation. Therefore, lookup or search operation is costly in linked lists in comparison to arrays.

Extra memory space for a pointer is required with each element of the list.

Not cache-friendly. Since array elements are present at contiguous locations, there is a locality of reference which is not there in the case of linked lists.

Representation of Linked Lists

A linked list is represented by a pointer to the first node of the linked list. The first node is called the head node of the list. If the linked list is empty, then the value of the head node is NULL.

Each node in a list consists of at least two parts:

data

Pointer (Or Reference) to the next node

In C++, we can represent a node using structure. Below is an example of a linked list node with integer data.

```
struct Node
{
   int data;
   struct Node* next;
};
```

Here's Another example

```
struct Node
{
     char name[20];          // Name of up to 20 letters
     int age;                // D.O.B. would be better
     float height;           // In meters
     Node *next;             // Pointer to next node
};
```

Exercise 1: Create a three node linked list

```
#include <iostream>
using namespace std;
struct Node
{
   int data; // Data
   Node *next;  // Pointer
};

//change the compiler settings to -std=c++11
//compiler options
//general settings select the first checkbox
//add the following command to when calling to compiler
//the add -std=c++11
//will update your compiler to c++11
//the compile the file
// Program to create a simple linked
// list with 3 nodes
int main()
{
   Node* head= nullptr;
```

```
    Node* second = nullptr;
    Node* third = nullptr;

    // allocate 3 nodes in the heap
    head = new Node;
    second = new Node;
    third = new Node;
    head->data = 1; //assign data in first node

    // Link first node with the second node
    head->next = second;
    second->data = 2;
    second->next = third;
    third->data = 3; //assign data to third node
    third->next=nullptr;

      // Print the linked list


    return 0;
}
```

Question

Try to draw out the diagrams on a piece of paper to understand what is happening in exercise 1

Exercise 2 : print out the content of the linked list starting from a given node

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

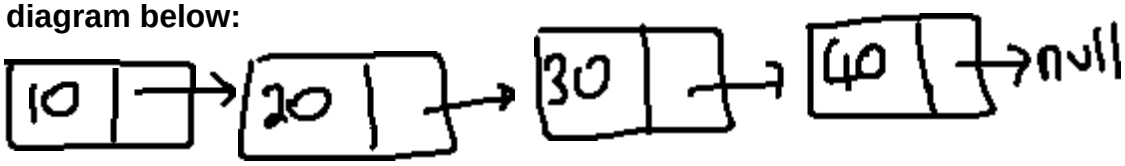
```
{
    int data; // Data
    Node *next;  // Pointer
};
// This function prints contents of linked list
// starting from  the given node
void printList(Node *node)
{
    while (node != nullptr)
    {
        cout<<node->data<<" ";
        node = node->next;
    }
}

//change the compiler settings to -std=c++11
//compiler options
//general settings select the first checkbox
//add the following command to when calling to compiler
//the add -std=c++11
//will update your compiler to c++11
//the compile the file
// Program to create a simple linked
// list with 3 nodes
int main()
{
    Node* head= nullptr;
    Node* second = nullptr;
    Node* third = nullptr;

    // allocate 3 nodes in the heap
    head = new Node;
```

```
    second = new Node;
    third = new Node;
    head->data = 1; //assign data in first node

    // Link first node with the second node
    head->next = second;
    second->data = 2;
    second->next = third;
    third->data = 3; //assign data to third node
    third->next=nullptr;

      // Print the linked list
    printList(head);

    return 0;
}
```

Question

Go through the codes in exercise 2, then answer the question below:

1. **Explain how does the code work when there is four nodes like the diagram below:**



2. **Explain how does the code work when there is only one node in the linked list**

3. **Explain how does the code work when there is no node at all. Please look at the diagram below:**

Insertion of singly linked list

Given the head node of a linked list, the task is to insert a new node in this already created linked list.

There can be many different situations that may arise while inserting a node in a linked list. Three most frequent situations are:

1. **Inserting a node at the start of the list.**
2. **Inserting a node after any given node in the list.**
3. **Inserting a node at the end of the list.**

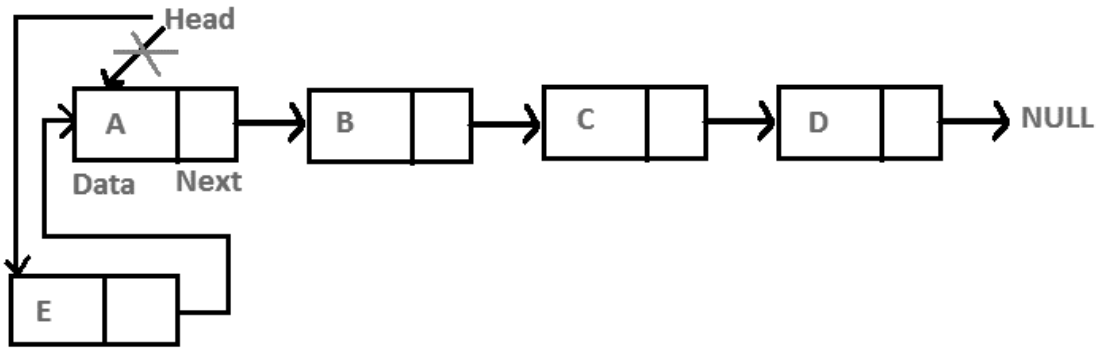
We have seen that a linked list node can be represented using structures as:

```
// A linked list node
struct Node
{
  int data;
  struct Node *next;
};
```

Let us now take a look at each of the above three listed methods of inserting a node in the linked list:

Inserting a Node at Beginning: Inserting a node at the start of the list is a four-step process. In this process, the new node is always added before the head of the given Linked List and the newly added node becomes the new head of the Linked List.

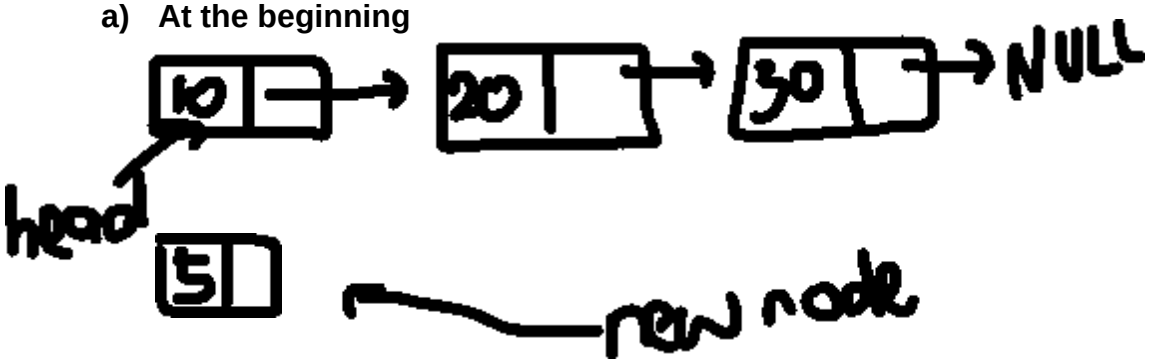
For example, if the given Linked List is 10->15->20->25 and we add an item 5 at the front, then the Linked List becomes 5->10->15->20->25.



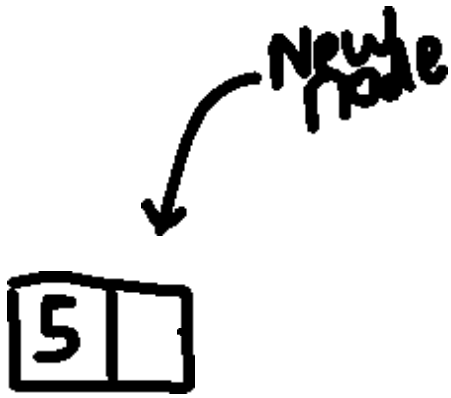
Question

Explain how does the process of inserting a node at head will happen for each of the scenario listed below

a) At the beginning



In the end:



b) At the beginning

In the end

c) At the beginning

In the end

Exercise 3

```
#include<iostream>
using namespace std;
struct Node{
   int data;
   Node* next;
   Node(int x){
      data=x;
      next=NULL;
```

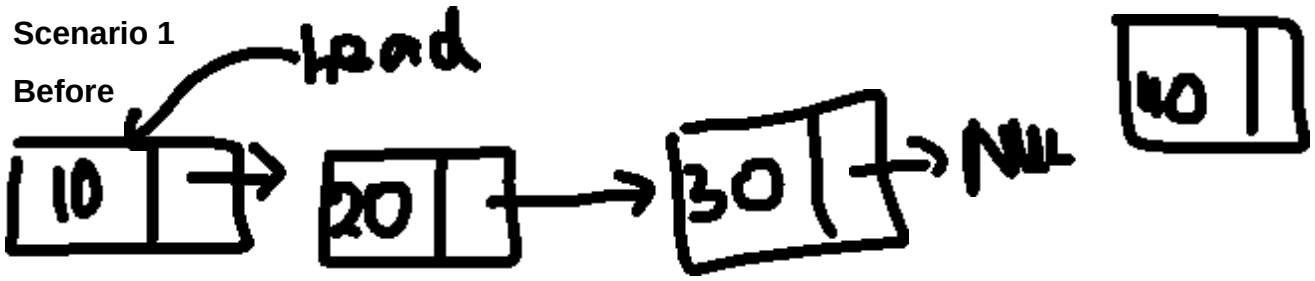
```
    }
};

//function insertBegin will return a pointer, that is why
//there is a * symbol at the function header, as you can
//see below
//the Node *head pointer is not the same with the
// head pointer you declared in int main()
//so you need to return back the node
Node *insertBegin(Node *head,int x){
    Node *temp=new Node(x);
    temp->next=head;
    return temp;

}
void printlist(Node *node)
{
    while (node != nullptr)
    {
        cout<<node->data<<" ";
        node = node->next;
    }
}


int main()
{       Node *head=NULL;
        head=insertBegin(head,30);
        head=insertBegin(head,20);
        head=insertBegin(head,10);
        printlist(head);
        return 0;
```

}

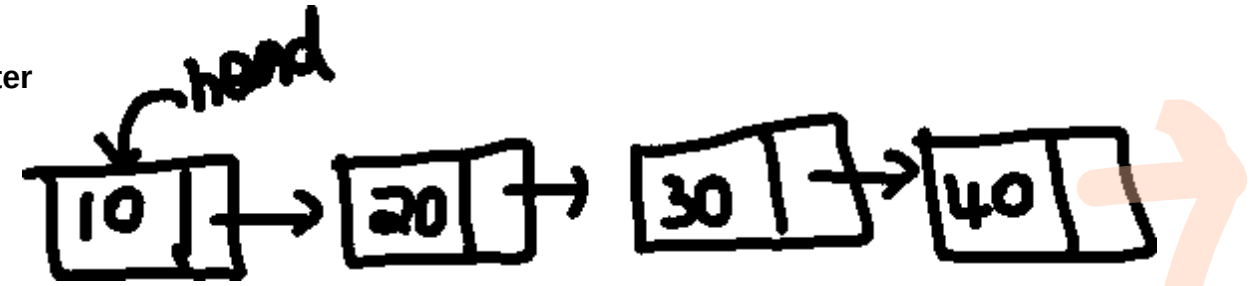
Adding a node at the end of the linked list

Scenario 1

Before



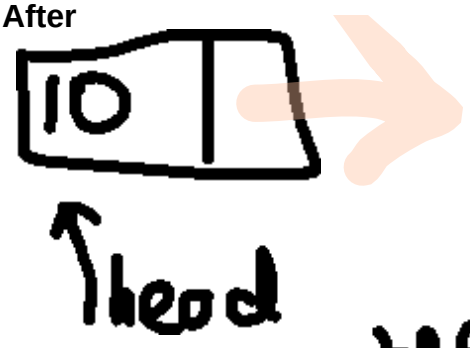
After



Scenario 2

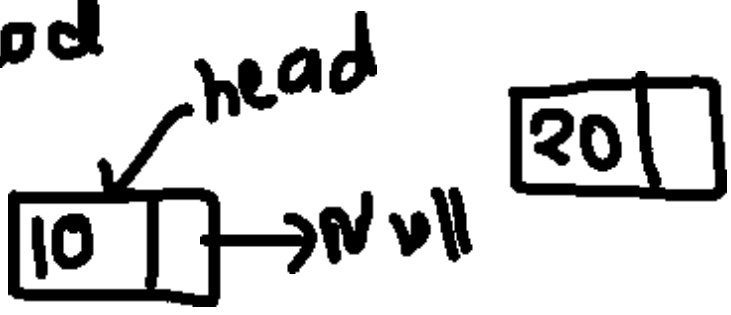
Before

After

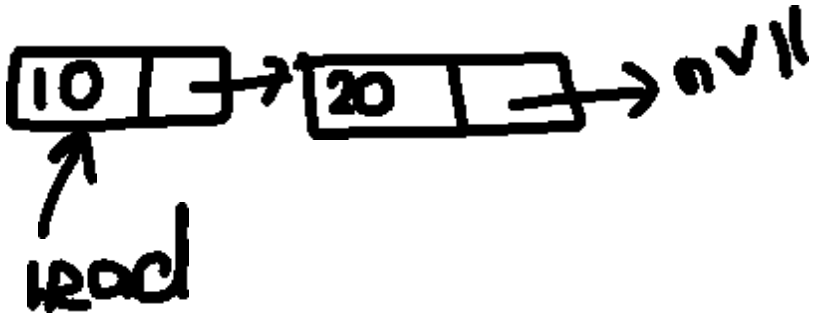


Scenario 3

Before

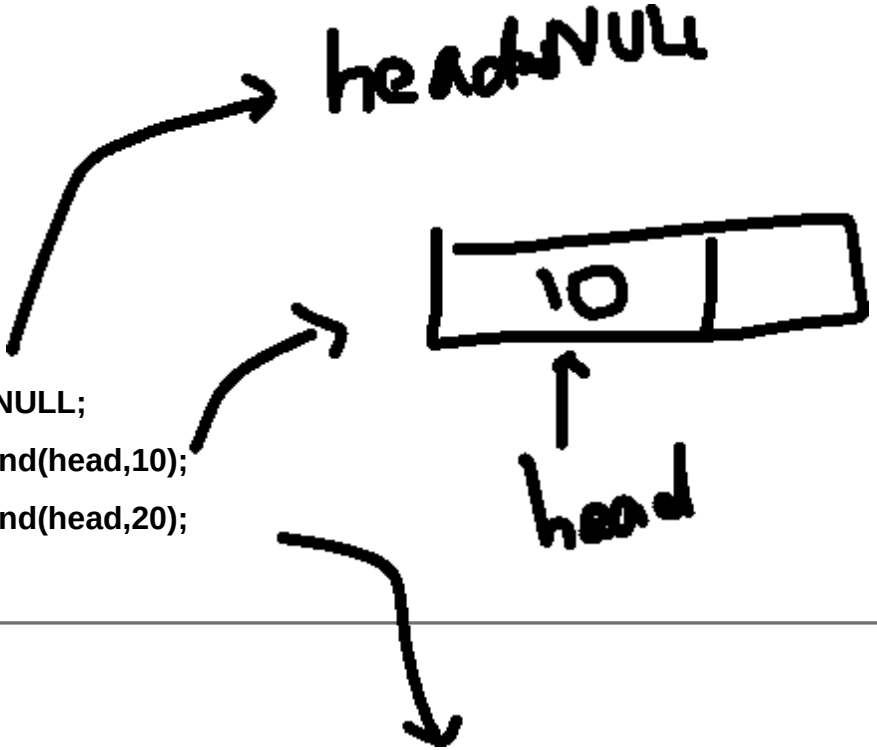


After

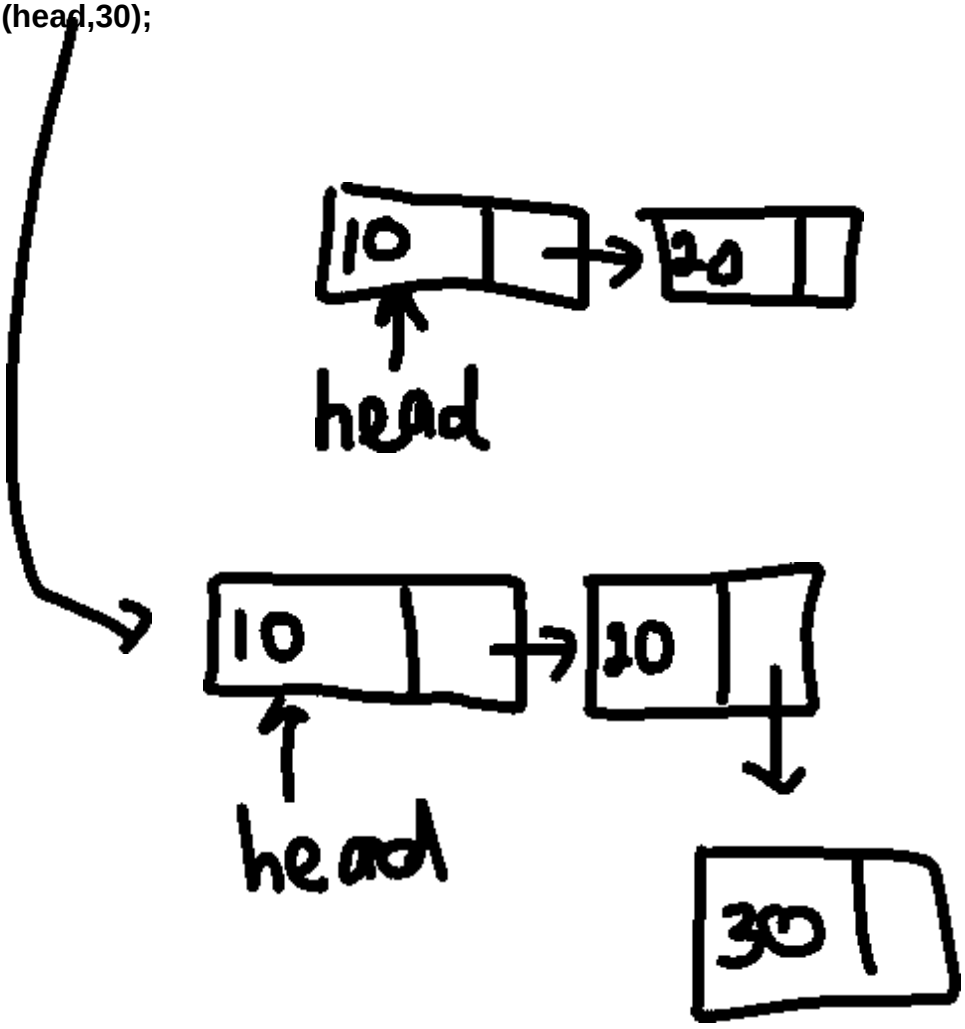


Exercise 4: Adding a node at the end of the linked list

```
struct Node{
    int data;
    Node* next;
    Node(int x){
        data=x;
        next=NULL;
    }
};

Node *insertEnd(Node *head,int x){
    Node *temp=new Node(x);
    if(head==NULL)return temp;
    Node *curr=head;
    while(curr->next!=NULL){
        curr=curr->next;
    }
    curr->next=temp;
    return head;

}

int main()
{
        Node *head=NULL;
        head=insertEnd(head,10);
        head=insertEnd(head,20);
```



```
        head=insertEnd(head,30);

        return 0;
}
```



Exercise 5: Deleting a node at the beginning of the linked list

```
struct Node{
   int data;
   Node* next;
   Node(int x){
      data=x;
      next=NULL;
   }
};
```

```
Node *delHead(Node *head){
   if(head==NULL)return NULL;
   else{
      Node *temp=head->next;
      delete(head);
      return temp;
   }
}
int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);
        head=delHead(head);
        return 0;
}
```

Exercise 6: Delete the node at the end of the linked list

```
struct Node{
   int data;
   Node* next;
   Node(int x){
      data=x;
      next=NULL;
   }
};


Node *delTail(Node *head){
   if(head==NULL)return NULL;
   if(head->next==NULL){
      delete head;
      return NULL;
```

```
    }
    Node *curr=head;
   while(curr->next->next!=NULL)
      curr=curr->next;
    delete (curr->next);
    curr->next=NULL;
    return head;
}
int main()
{
        Node *head=new Node(10);
        head->next=new Node(20);
        head->next->next=new Node(30);

        head=delTail(head);


        return 0;
}
```