Data Structures
Lab Manual 3

Topic: Singly Linked Lists

Session: Spring 2023

Faculty of Information Technology

UCP Lahore Pakistan

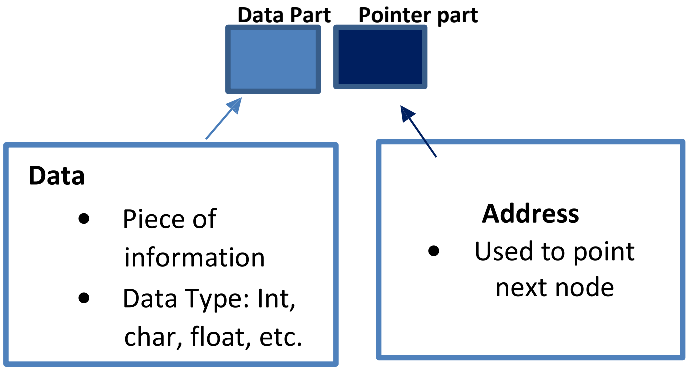
Objectives

- Understanding the basic concept of a singly linked list

- To write a program that implements the basic operations of the linked list in C++.

Linked List Overview

- ✓ List is a collection of components, called nodes. Every node (except the last node) contains the address of the next node. Every node in a linked list has two components:
  - ➢ One to store the relevant information (that is, data)
  - ➢ One to store the address, called the link or next, of the next node in the list.
- ✓ The address of the first node in the list is stored in a separate location, called the head or first.
- ✓ The address of the last node in the list is stored in a separate location, called the tail or last.
- ✓ A Linked List is a set of nodes where each node has two fields' "Data" and "Address".

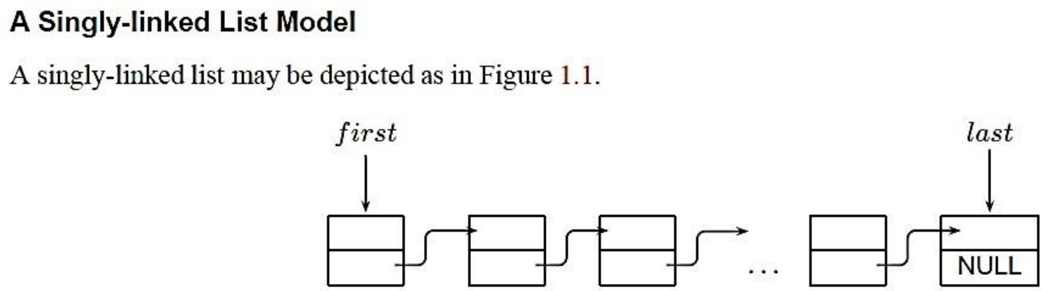

Data Part
Pointer part
Data
• Piece of information
• Data Type: Int, char, float, etc.
Address
• Used to point next node

Types of linked lists:

- ✓ Singly linked list
- ✓ Circular, singly linked list
- ✓ Doubly linked list
- ✓ Circular, double linked list

Basic Operations that can be performed on Linked List:

- ✓ Traversal:
  To traverse all the nodes one after another
- ✓ Append a new node:
  To add a new node at the end
- ✓ Prepend a new node:
  To add a new node at the beginning
- ✓ Insertion:
  To add a node at the given position
- ✓ Deletion:
  To delete a node from the list
- ✓ Updating:
  To update a node in the list
- ✓ Searching:
  To search an element by value
- ✓ Sorting:
  To arrange nodes in a linked list in a specific order
- ✓ Merging:
  To merge two linked lists into one



A Singly-linked List Model

A singly-linked list may be depicted as in Figure 1.1.

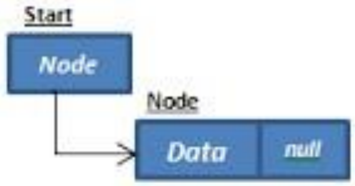
first
last
...
NULL

Inserting a Node in a Singly-linked List

The node to be inserted may be created. We will assume here that the node to insert is pointed to by p.

- If the list was empty then there would be only one node, i.e.

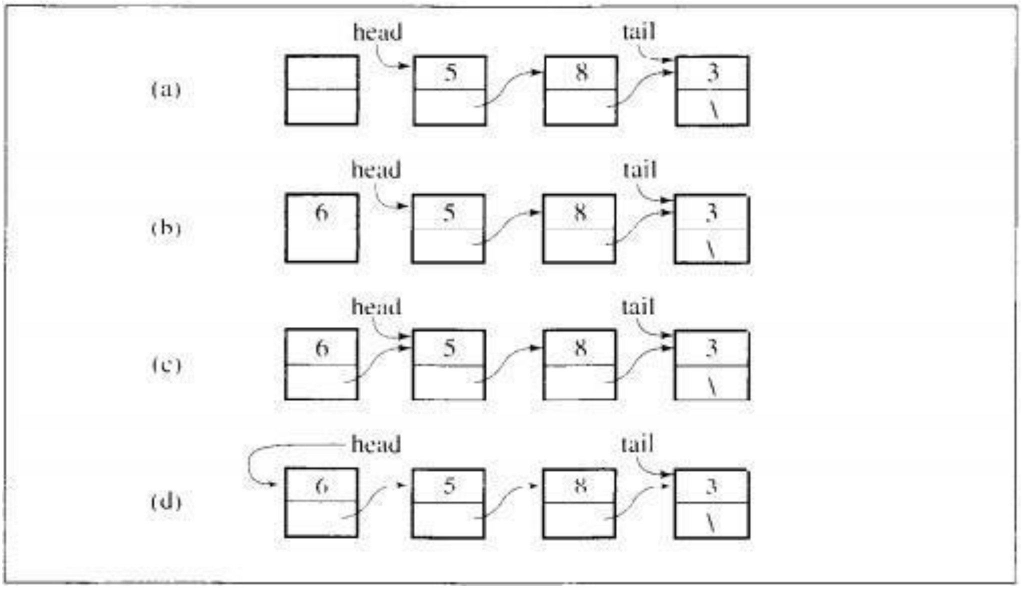
```
 if ( first == NULL )
 {
     first = p;
     last = p;
    p–>next = NULL;
 }
```



If the list was not empty, then insertion follows different patterns depending on the position where the node is to be inserted. Thus, we have the following cases:

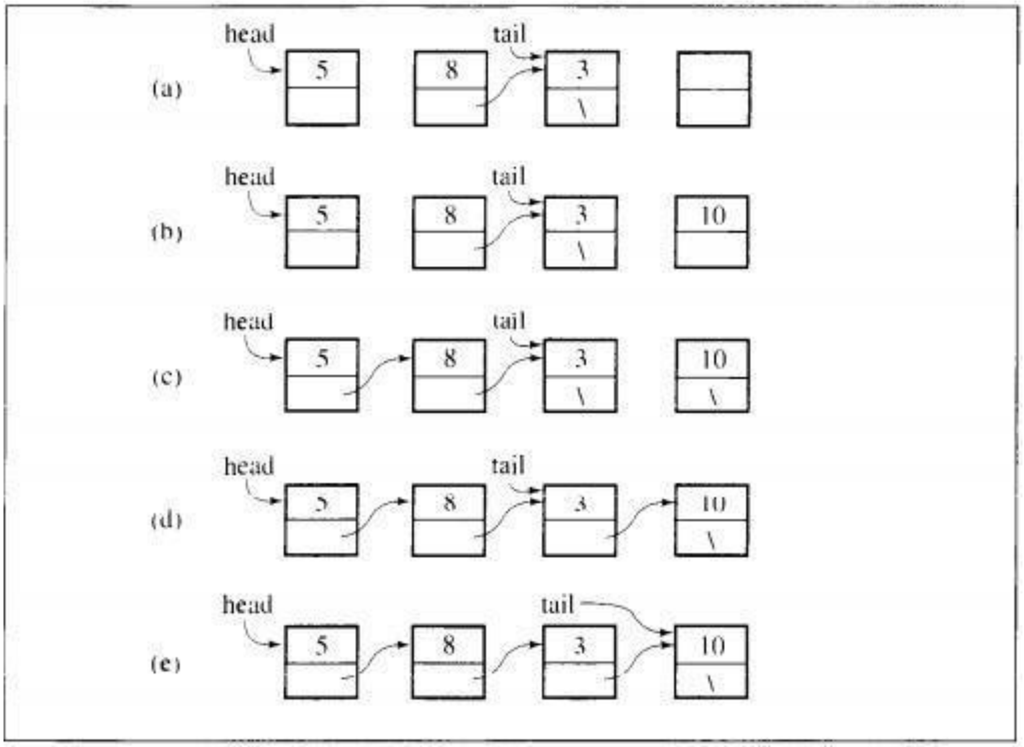
1. Insertion before the first node of the list:

```
if ( first != NULL )
{
     p–>next = first ;
    first = p;
}
```



2. Insertion after the last node of the list (this operation is also called *append*):

```
if ( last != NULL )
{
        p–>next = NULL;
       last–>next = p;
      last = p;
}
```

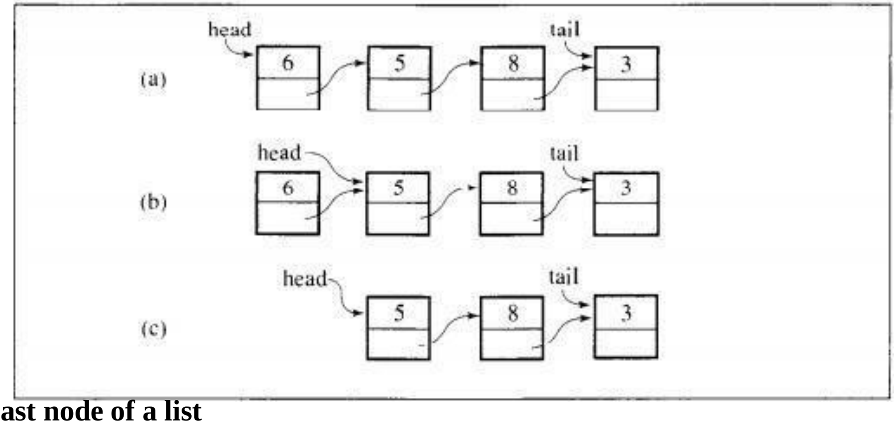


Removing a Node of a Singly-linked List

When we are to remove a node from a list, there are some aspects to take into account: (i) list may be empty; (ii) list may contain a single node; (iii) list has more than one node. And, also, deletion of the first, the last or a node given by its key may be required. Thus we will discuss some cases here others will be yours task.

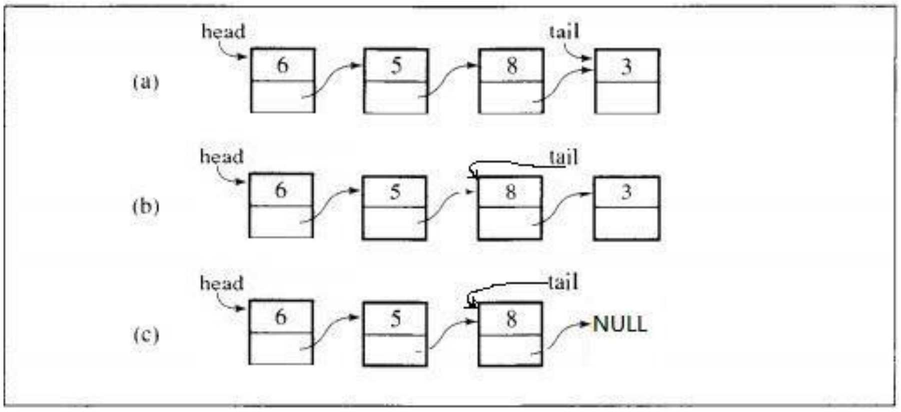
1. Removing the first node of a list

```
NodeT *p;
if ( first != NULL )
{ /* non−empty list */
          p = first ;
          first = first−>next ;
          free( p ) ; /* free up memory */
         if ( first == NULL ) /* list is now empty */
              last = NULL;
}
```



2. Removing the last node of a list

```
NodeT *q, *q1;
q1 = NULL; /* initialize */
q = first ;
if ( q != NULL )
{ /* non−empty list */
while ( q != last )
{ /* advance towards end */
     q1 = q;
     q = q−>next ;
 }
if ( q == first )
{ /* only one node */
    first = last = NULL;
}
else
{ /* more than one node */
    q1−>next = NULL;
    last = q1;
}
 free( q ) ;
}
```



Complete Deletion of a Singly-linked List

For a complete deletion of a list, we have to remove each node of that list, i.e.

```
NodeT *p;
while ( first != NULL )
{
     p = first ;
    first = first−>next ;
   free ( p ) ;
}
last = NULL;
```

Sample Code:

Inserting a Node in a Singly-linked List

The node to be inserted may be created. We will assume here that the node to insert is pointed to by p.

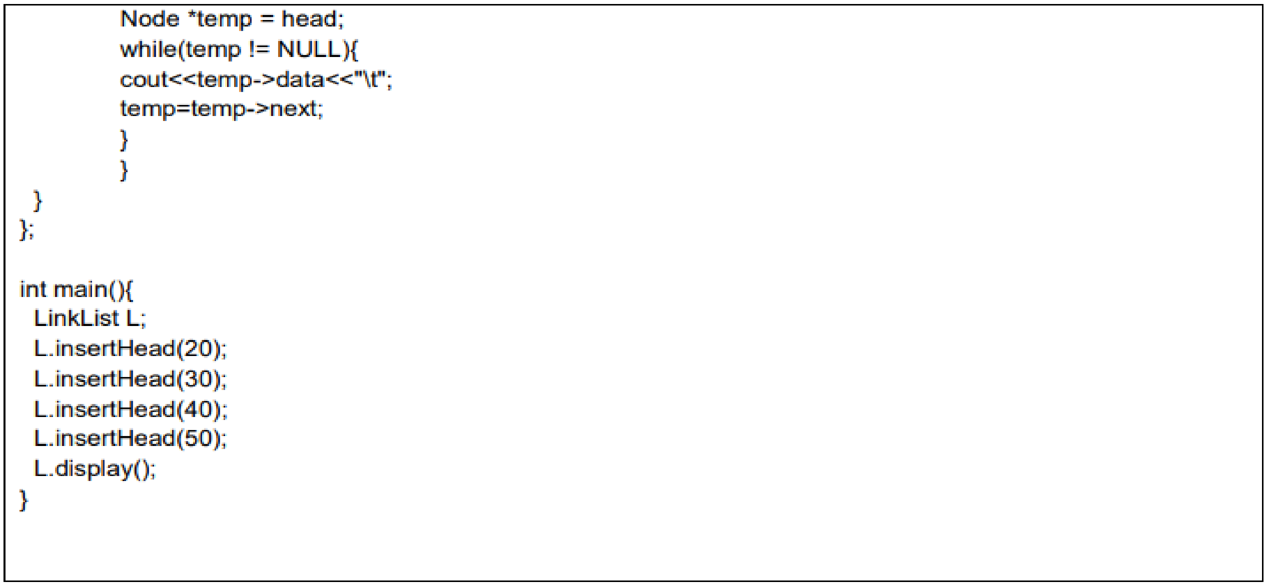
```
// Example program
#include <iostream>
using namespace std;
class Node {
public:
  int data;
  Node *next;

  Node (int val) {
          data = val;
          next = NULL;
  }
};
class LinkList {
public:
  Node* head;

  LinkList () {
          head = NULL;
  }

  void insertHead (int val) {
          Node *mynode= new Node(val);
          if(head == NULL){
          head=mynode;
          }
          else{
          mynode->next=head;
          head=mynode;
          }
  }

  void display(){
          if(head == NULL){
          cout<<"List is empty"<<endl;
          }
          else{
```



```
            Node *temp = head;
            while(temp != NULL){
            cout<<temp->data<<"\t";
            temp=temp->next;
            }
            }
  }
};

int main(){
  LinkList L;
  L.insertHead(20);
  L.insertHead(30);
  L.insertHead(40);
  L.insertHead(50);
  L.display();
}
```

Lab Tasks

Task 1:

Write a program that uses class LinkedList to create a linked list of float and then display it on console.

Task 2:

Perform the following operations.

1. Create your own class of link list which will have the following functions.
   a. Function called **InsertAtBegin** to add node at the begging of list.
   b. Function called **InsertAtEnd** to add node at the last of list.
   c. Function called Display to display list on console.
   d. Function called Search to search a specific value in list.
   e. Function called Update to update value of list.