Lab 06

Single link list, Double link list, Link list operations

Objective:

- To understand Single and double link lists.
- To Implement link lists in C++

Introduction

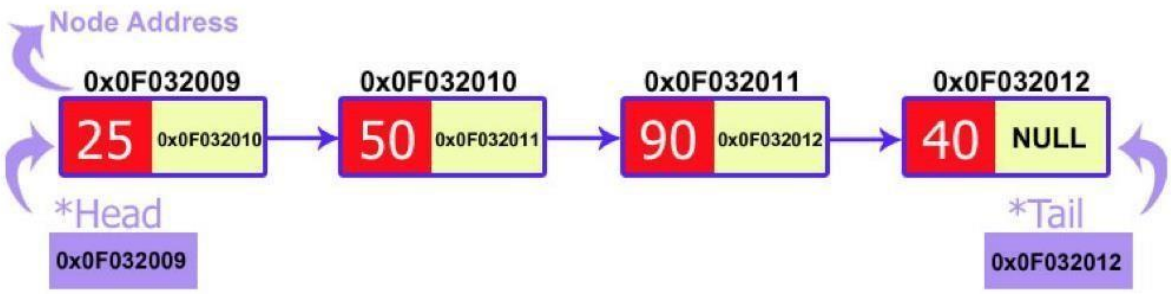
A linked list is just a chain of nodes, with each subsequent node being a child of the previous one. Many programs rely on linked lists for their storage because these don't have any evident restrictions. For example, the array list we did earlier could not grow or shrink, but node based ones can! This means there is no limit (other than the amount of memory) on the number of elements they can store.

A. Singly Linked Lists Basics:

- A singly linked list is a concrete data structure consisting of a sequence of nodes
- It has a head (or first) node pointer indicating the first node in list
- It could have optionally a tail pointer node indication the last node in list
- Each node stores
- Element (data)
- Link to the next node
- The null object (at the end) is denoted as ∅.

The elements of a linked list are called the **nodes**. A node has two fields i.e. **data** and **next**. The data field contains the data being stored in that specific node. It cannot just be a single variable. There may be many variables presenting the **data** section of a node. The **next** field contains the address of the next node. So this is the place where the link between nodes is established.

No matter how many nodes are present in the linked list, the very first node is called head and the last node is called the tail. If there is just one node created then it is called both head and tail.



Implementation of Linked List Using C++

As linked list consists of nodes, we need to declare a structure which defines a single node. Our structure should have at least one variable for data section and a pointer for the next node. In C++, our code would look like this:

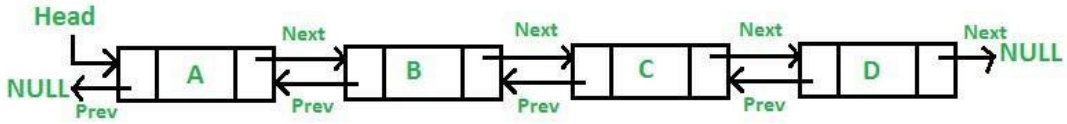
```
struct node
  {     int data;
    node *next;
  };

node *head, *tail;
```

B. Doubly Linked Lists

Basics:

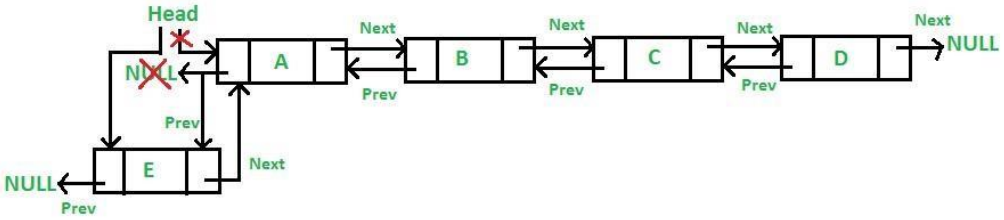
The doubly linked list allows us to go in both directions in a linked list:

- Forward
- Reverse
- A variety of quick update operations, including insertion and removal at both ends, and in the middle.
- A node in a doubly linked list stores two references A next link, which points to the next node in the list.
- A prev link, which points to the previous node in the list.



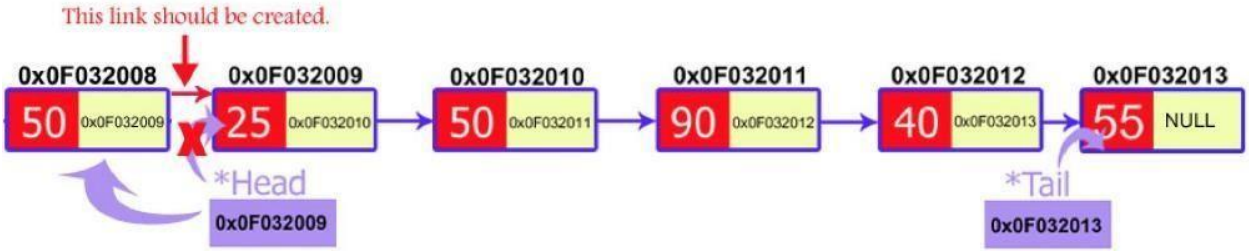

```
   struct Node
{
  int data;
  Node *next; // Pointer to next node in DLL
  Node *prev; // Pointer to previous node in DLL  };
Node *head, *tail;
```

Insertion at the front:



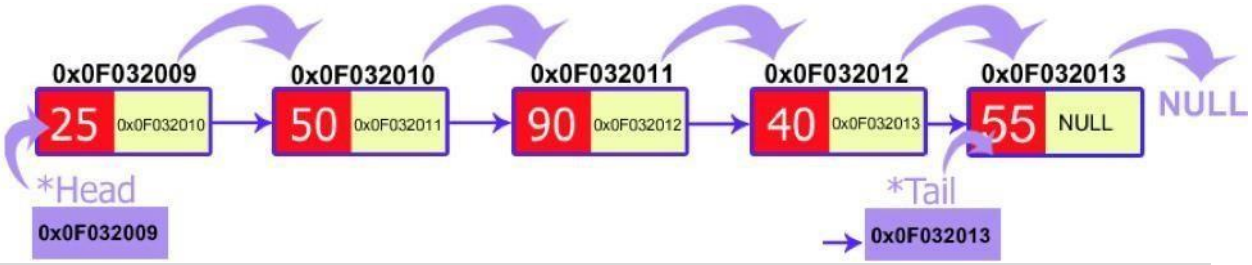

```
    void InsertAtHead(int value)
        {
          node *temp=new node;
    temp->data=value;              temp-
    >next=head;         temp->prev=null;
    if(head!=NULL)
          {
       head->previous = temp;
    }
          if(head==NULL)
          {
            head=temp;         tail=temp;       }else{  head=temp;  }
    }
```

Activity 1: Insert at Head




```
void InsertAtHead(int value)
    {
      node *temp=new node;
temp->data=value;        temp-
>next=head;        if(head==NULL)
      {         head=temp;
tail=temp;
       }
            else
{
          head=temp;
}
      }
```

Activity 2: Display Linked List

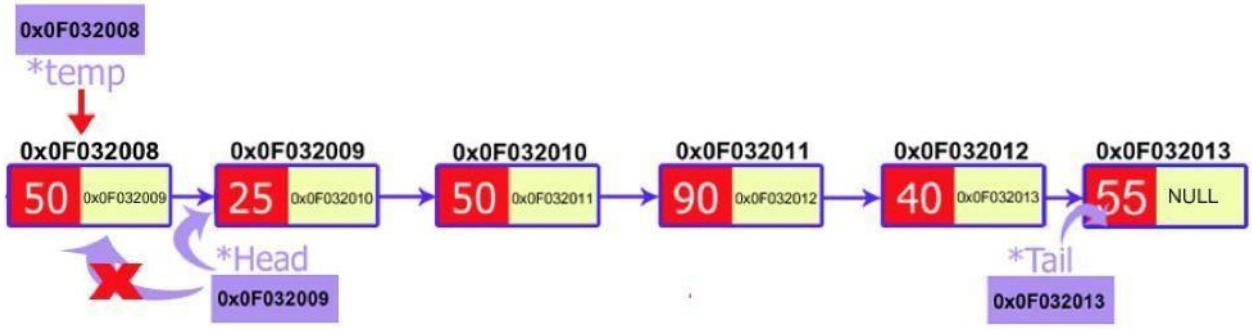



```
void display()
    {  node *temp=new
      node;
      temp=head;
      while(temp!=NULL)
       {
        cout<<temp->data<<",";          temp=temp-
      >next;
          }
       cout<<"\b \b";  // to remove last extra comma   }
```

Activity 3: Insert at Tail

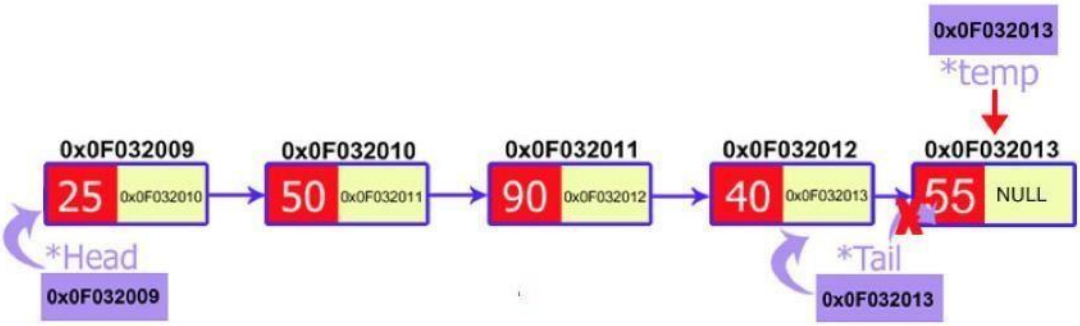
```
void InsertAtTail(int value)
      {
      node *temp = new node;
      temp -> data = value;   temp
      -> next = NULL;
       if (head == NULL) //if list is empty
       {
         head = tail = temp;    return;
       }
       tail->next = temp;  tail = temp;  }
```

Activity 4: Remove from Head



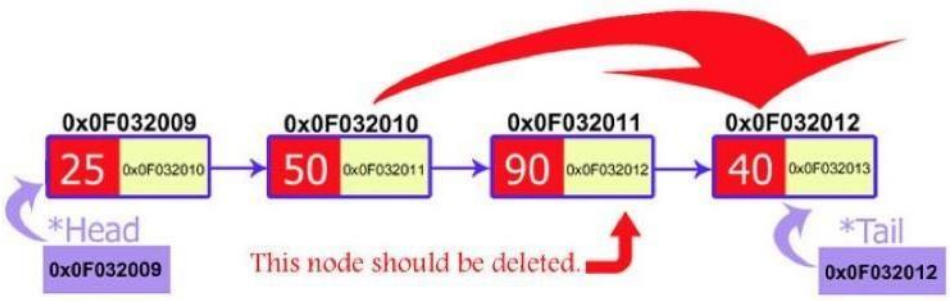
```
void RemoveFromHead(void)  {
if (head == NULL) return;
node *temp = new node;  temp
= head;  head = head ->
next;
 if (head == NULL) tail = NULL;   delete temp;
}
```

Activity 5: Remove from Tail




```
void RemoveFromTail(void)
  {      node *current=new
node;           node
*previous=new node;
current=head;
while(current->next!=NULL)
    {
previous=current;
current=current->next;
    }
       tail=previous;
previous->next=NULL;
delete current;
  }
```

Activity 6: Remove Specific Node



```
void DeleteSpecificNode(int value)
{  node *current, *previous;  current
= previous = head;
 while (current!=NULL && current->data!=value) //find address of node to
delete
 {
         previous = current;
         current = current->next;
 }
 if (current == NULL) //not found in list
 {
   cout<<"\nElement to delete not found in the list";
      return;
 }
 if (current == head) //node to delete is first node    head
= head->next;
 else if (current == tail) //node to delete is last node
 {
         tail = previous;
         previous->next = NULL;
 }
 else //node to delete is in middle of list        previous-
>next = current->next;
 if (head == NULL) tail=NULL; //there was only one element in list
 //and after deletion list is now empty  delete current;
}
```

Lab Tasks:

Task 1: Write and test a method public int size ( ) to count the number of nodes in the linked list.

Task 2: Write and test a method public Boolean search (int n) to find out whether the given data exists or not in the linked list.

Task 3. Swap nodes in a linked list without swapping data.

Home Tasks:

.

Q1. Write a function that counts the number of times a given int occurs in a Linked List.

Q2. Write a function Linked list traversal using recursion in c++.

Q3. There is a garage where the access road can accommodate any number of trucks at one time. The garage is build such a way that only the last truck entered can be moved out. Each of the trucks is identified by a positive integer (a truck_id). Write a program to handle truck moves, allowing for the following commands:

a) On_road (truck_id);
b) Enter_garage (truck_ id);
c) Exit_garage (truck_id);
d) Show_trucks (garage or road);

If an attempt is made to get out a truck which is not the closest to the garage entry, the error message Truck x not near garage door.

Q4. Simulate a game, using two circular doubly linked list. The game involves a number of children, and you are supposed to read their names from a file. The children whose names are on lines numbered by prime numbers should be placed on the first list, and the others on the second list. Starting with the child whose name is on the line in the middle (or ⌊numberOfChildren/2⌋) of the second list, children on that list are counted clockwise. Every mth child, where m is the number of elements in the first list is eliminated from that list. Counting goes on with the next child. Repeat this counting m times or till the second list gets empty. Your program should output the initial lists and the final second list.