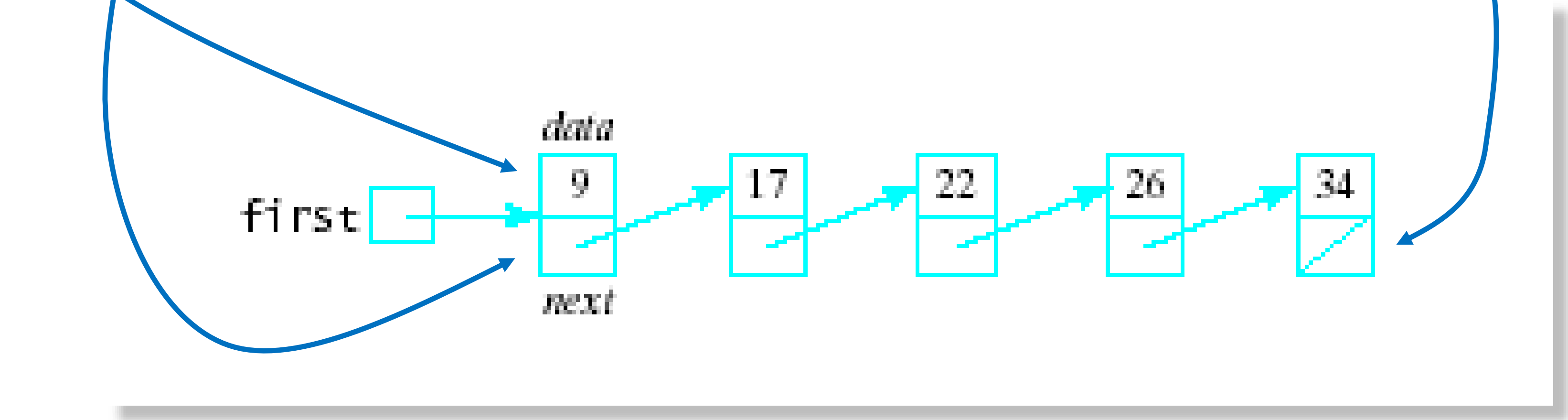
Data Structures

7. Linked List

Linked List Using Pointers-Based Implementation of Lists

Linked List

- Linked list nodes composed of two parts
  - Data part
    - Stores an element of the list
  - Next part
    - Stores link/pointer to next element
    - Stores Null value, when no next element



Simple Linked List Class (1)

- We use two classes: Node and List
- Declare Node class for the nodes
  - `data:` double-type data in this example
  - `next:` a pointer to the next node in the list

```
class Node {
   public:
      double   data;   // data
      Node*    next;   // pointer to next
};
```

Simple Linked List Class (2)

- Declare List, which contains
  - `head:` a pointer to the first node in the list
  - Since the list is empty initially, `head` is set to `NULL`

```
class List {
    public:
        List(void) { head = NULL; } // constructor
        ~List(void);                // destructor

        private:
        Node* head;
};
```

Simple Linked List Class (3)

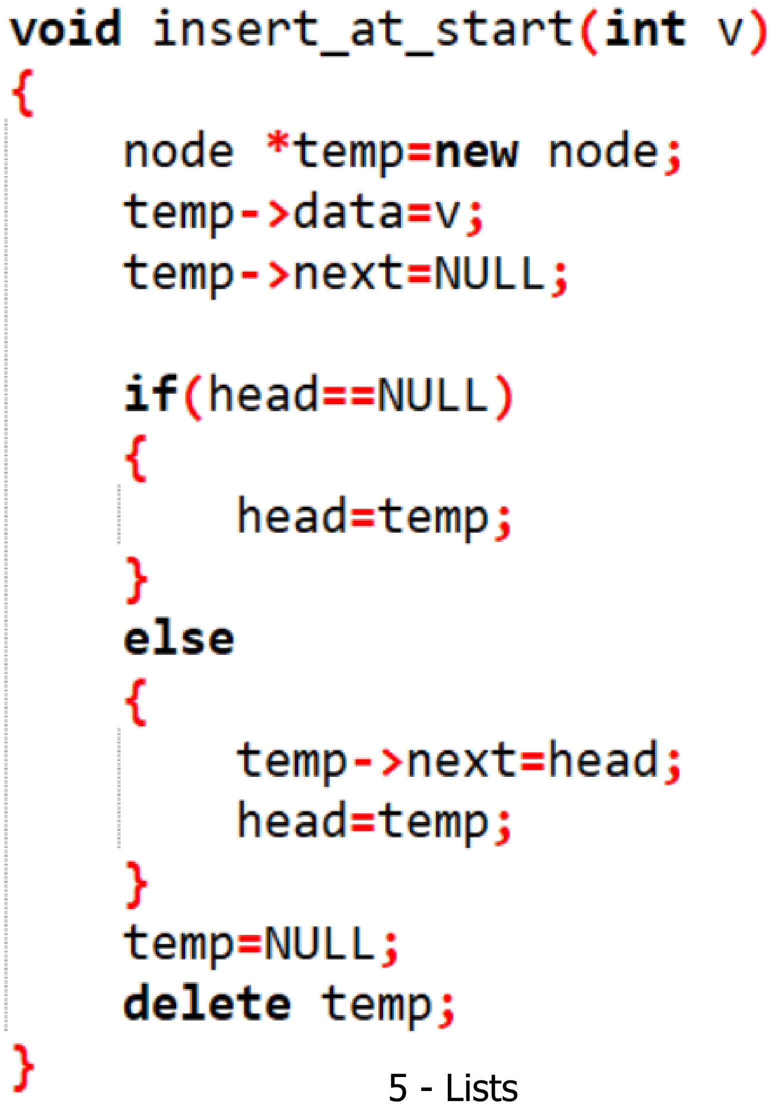
Operations of List

- `IsEmpty`: determine whether or not the list is empty
- `InsertNode`: insert a new node at a particular position
- `FindNode`: find a node with a given value
- `DeleteNode`: delete a node with a given value
- `DisplayList`: print all the nodes in the list

Inserting a New Node

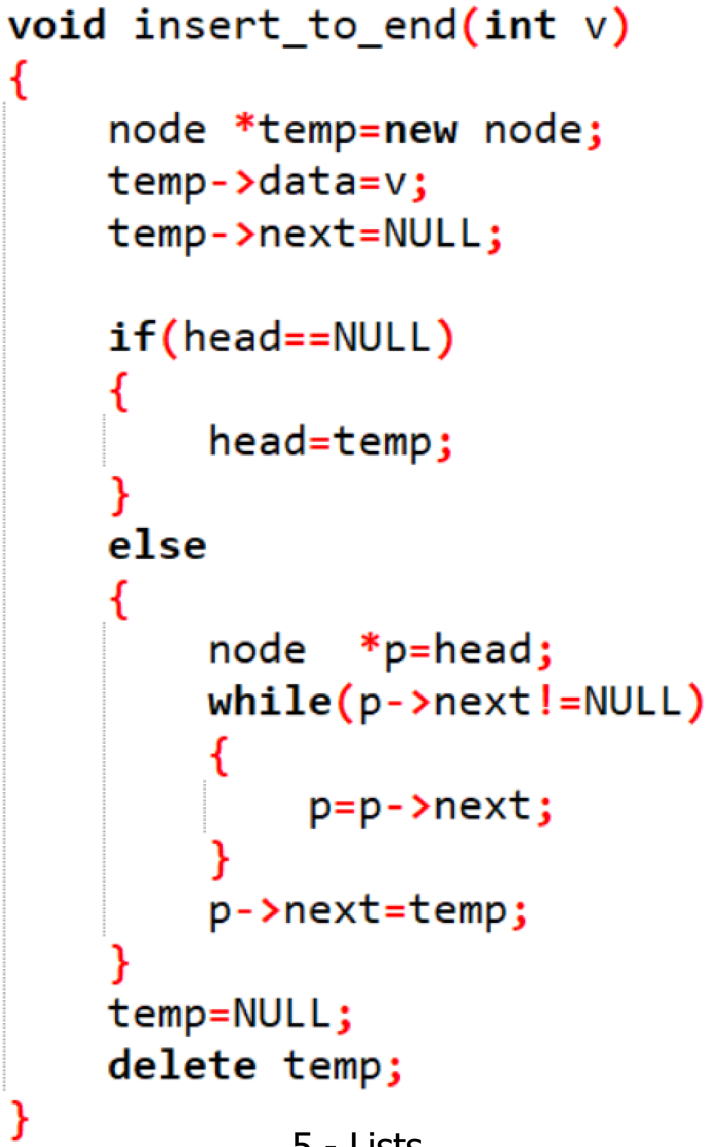
- Four Insertions
  - At start
  - At End
  - Before/After a Value
  - Before/After a Location

Insert at the start



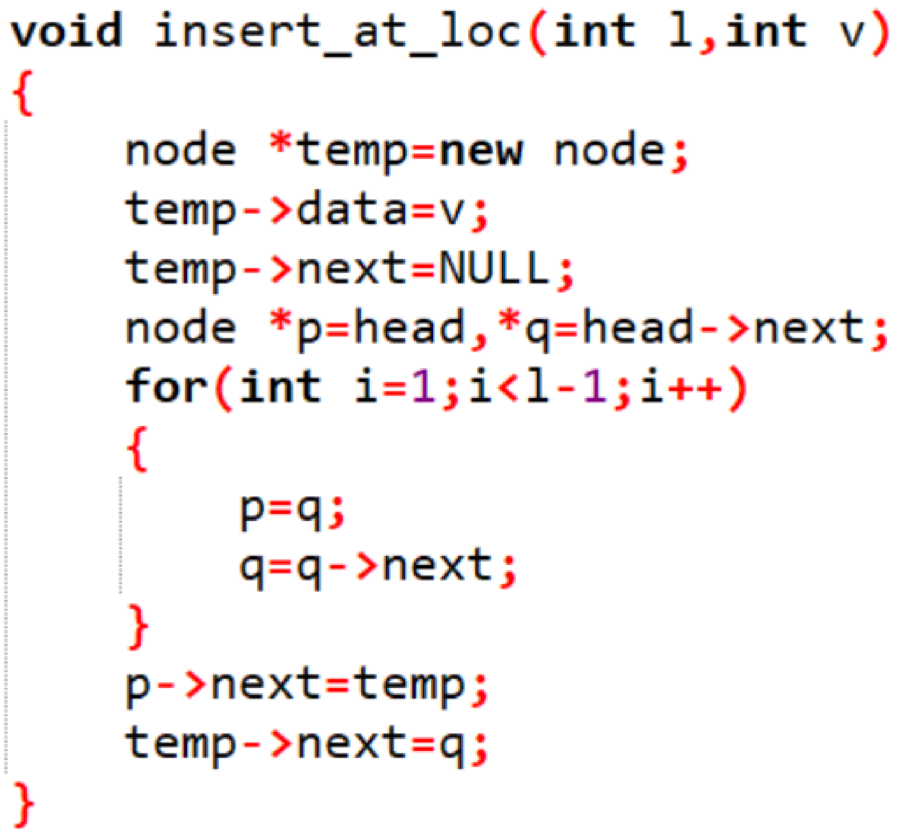
```
void insert_at_start(int v)
{
    node *temp=new node;
    temp->data=v;
    temp->next=NULL;

    if(head==NULL)
    {
        head=temp;
    }
    else
    {
        temp->next=head;
        head=temp;
    }
    temp=NULL;
    delete temp;
}
```

Insert at the end



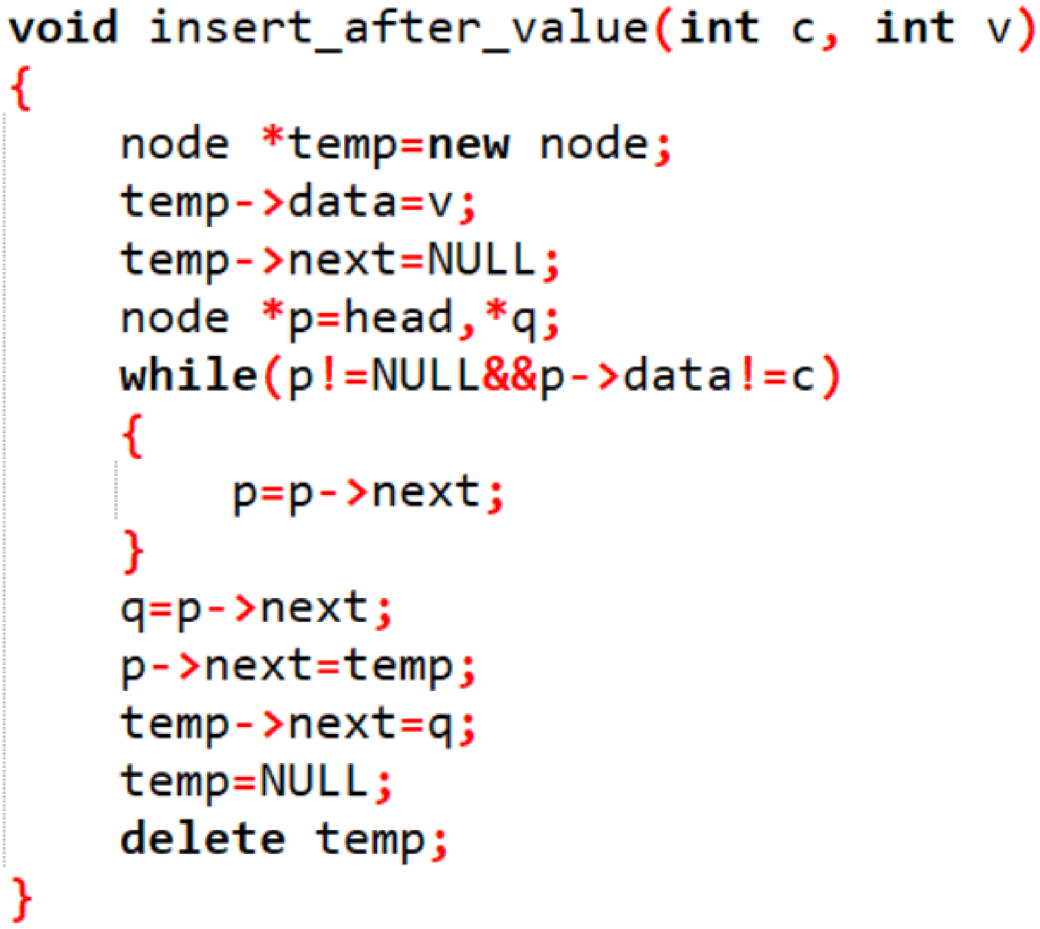
```
void insert_to_end(int v)
{
    node *temp=new node;
    temp->data=v;
    temp->next=NULL;

    if(head==NULL)
    {
        head=temp;
    }
    else
    {
        node  *p=head;
        while(p->next!=NULL)
        {
            p=p->next;
        }
        p->next=temp;
    }
    temp=NULL;
    delete temp;
}
```

Insert before/after location



```
void insert_at_loc(int l,int v)
{
    node *temp=new node;
    temp->data=v;
    temp->next=NULL;
    node *p=head,*q=head->next;
    for(int i=1;i<l-1;i++)
    {
        p=q;
        q=q->next;
    }
    p->next=temp;
    temp->next=q;
}
```

Insertion before/after value

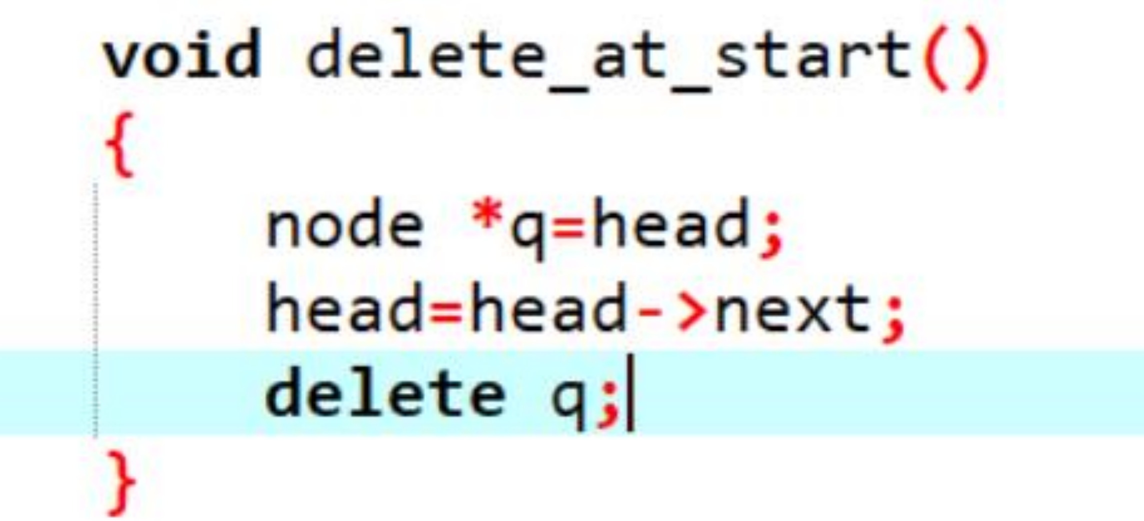


```
void insert_after_value(int c, int v)
{
    node *temp=new node;
    temp->data=v;
    temp->next=NULL;
    node *p=head,*q;
    while(p!=NULL&&p->data!=c)
    {
        p=p->next;
    }
    q=p->next;
    p->next=temp;
    temp->next=q;
    temp=NULL;
    delete temp;
}
```

Deleting a Node

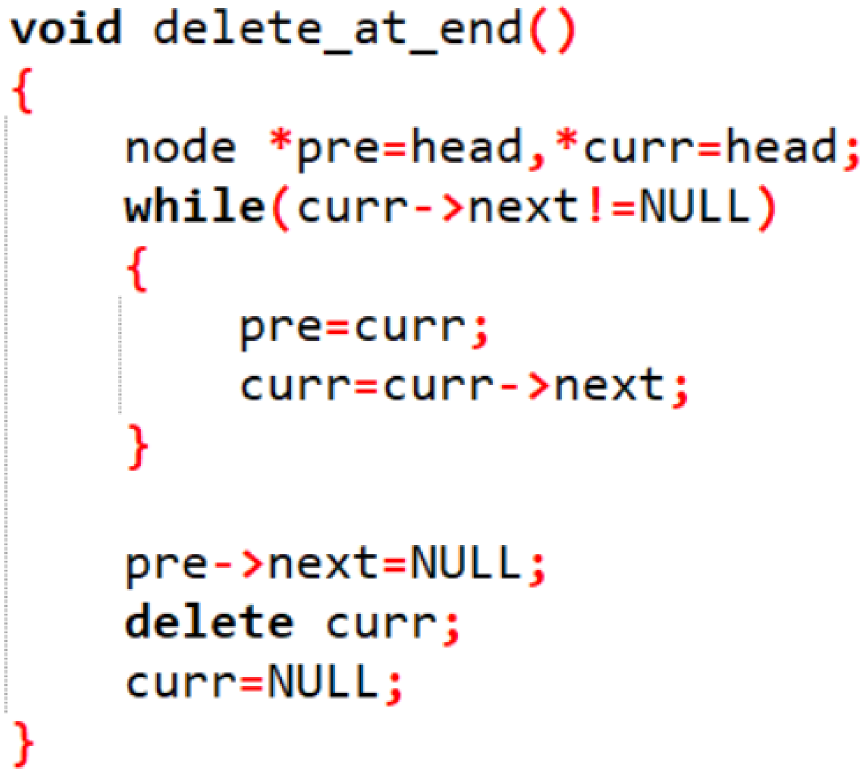
- Four Deletions
  - At start
  - At End
  - A Value
  - A Location

Delete at Start



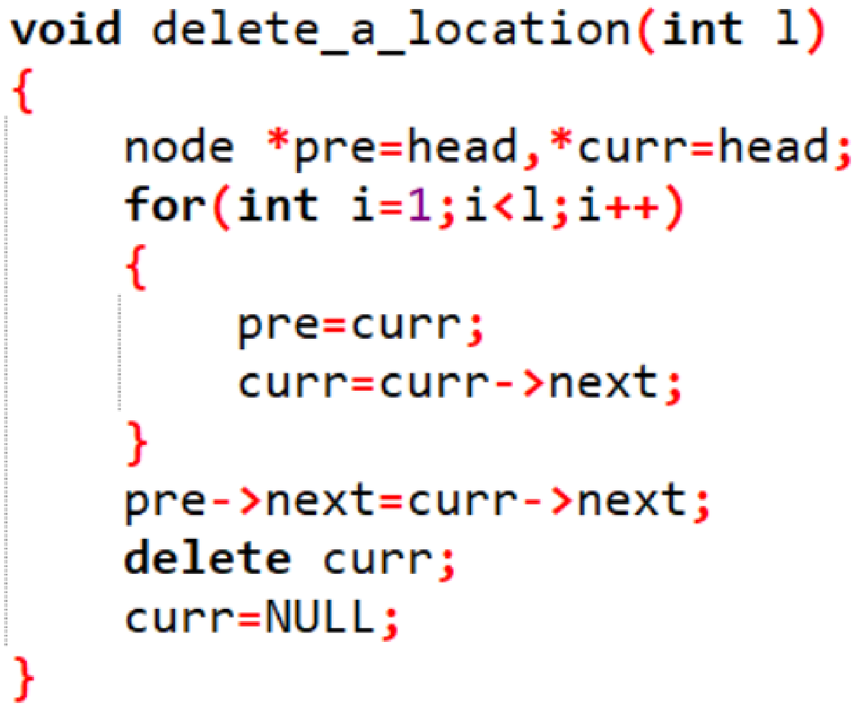
```
void delete_at_start()
{
    node *q=head;
    head=head->next;
    delete q;
}
```

Delete at end



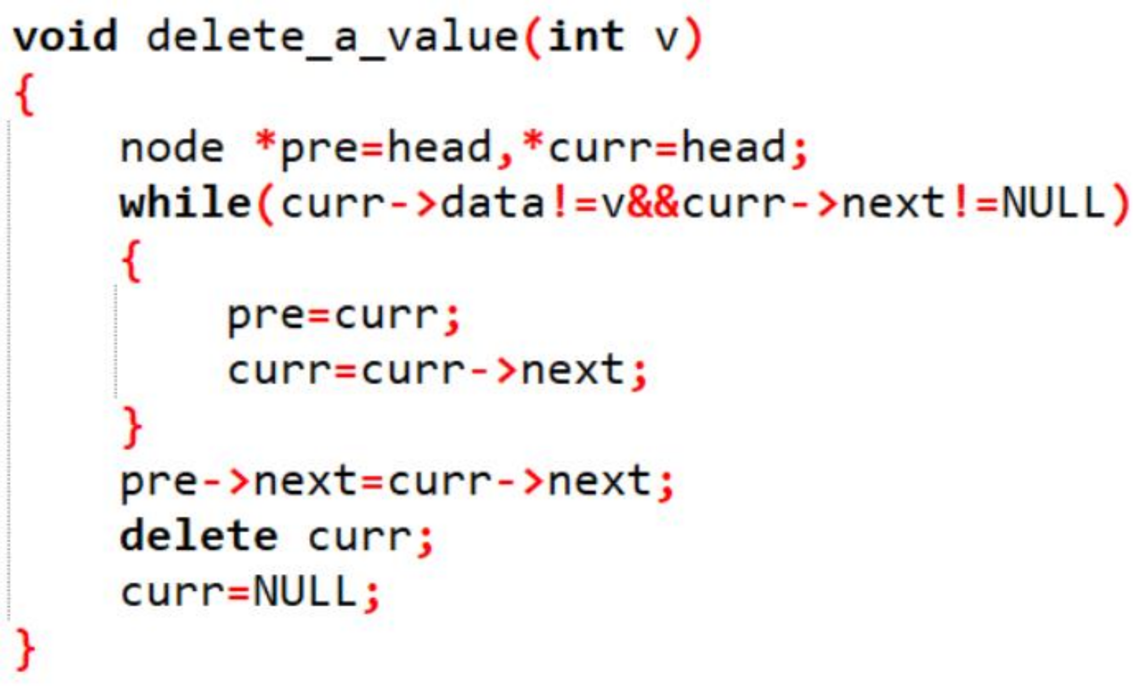
```
void delete_at_end()
{
    node *pre=head,*curr=head;
    while(curr->next!=NULL)
    {
        pre=curr;
        curr=curr->next;
    }

    pre->next=NULL;
    delete curr;
    curr=NULL;
}
```

Delete at Location



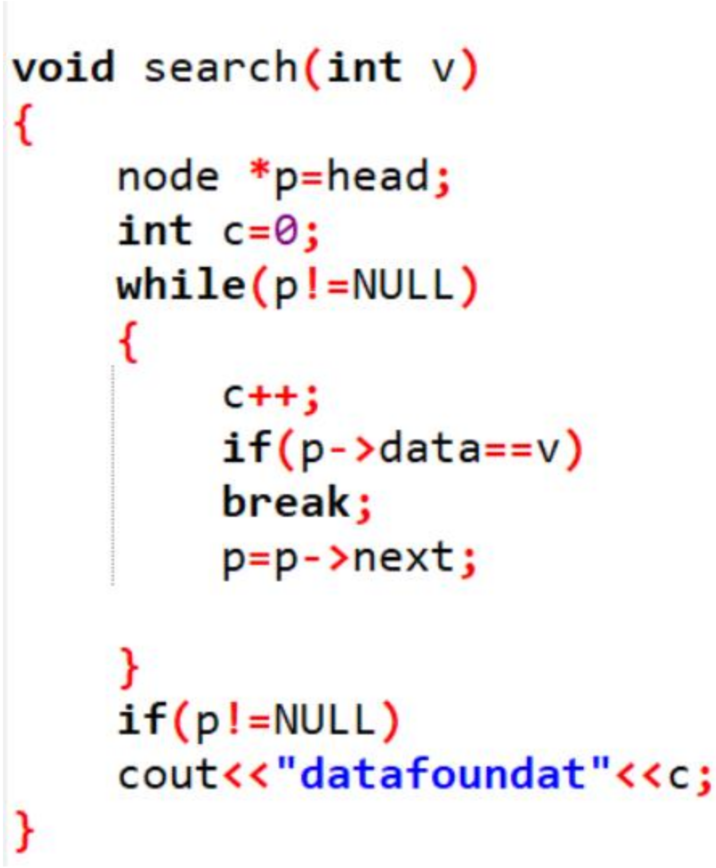
```
void delete_a_location(int l)
{
    node *pre=head,*curr=head;
    for(int i=1;i<l;i++)
    {
        pre=curr;
        curr=curr->next;
    }
    pre->next=curr->next;
    delete curr;
    curr=NULL;
}
```

Delete a Value



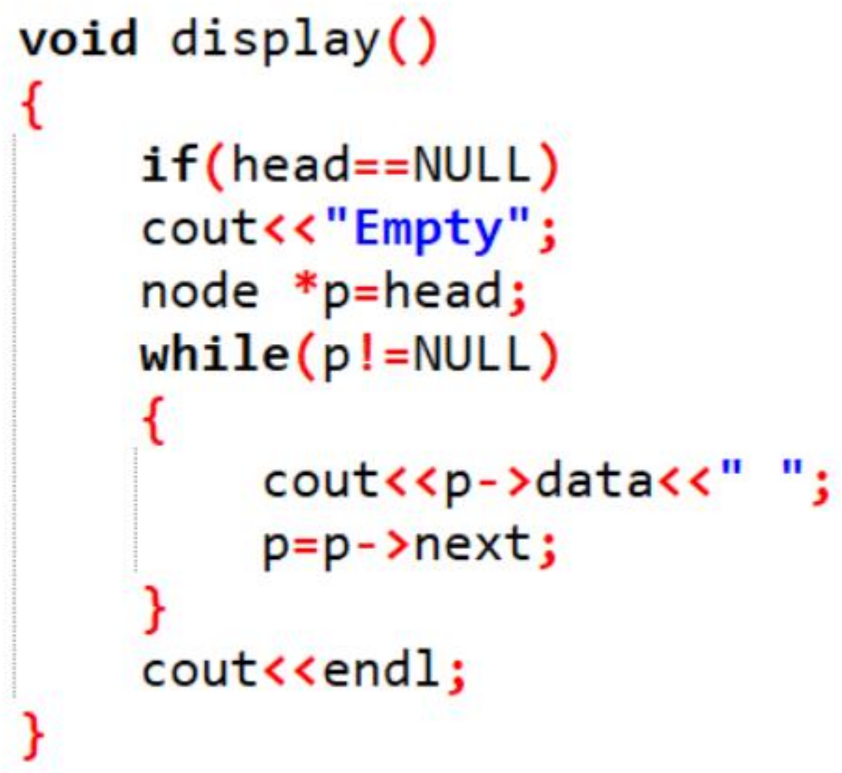
```
void delete_a_value(int v)
{
    node *pre=head,*curr=head;
    while(curr->data!=v&&curr->next!=NULL)
    {
        pre=curr;
        curr=curr->next;
    }
    pre->next=curr->next;
    delete curr;
    curr=NULL;
}
```

Search a Node



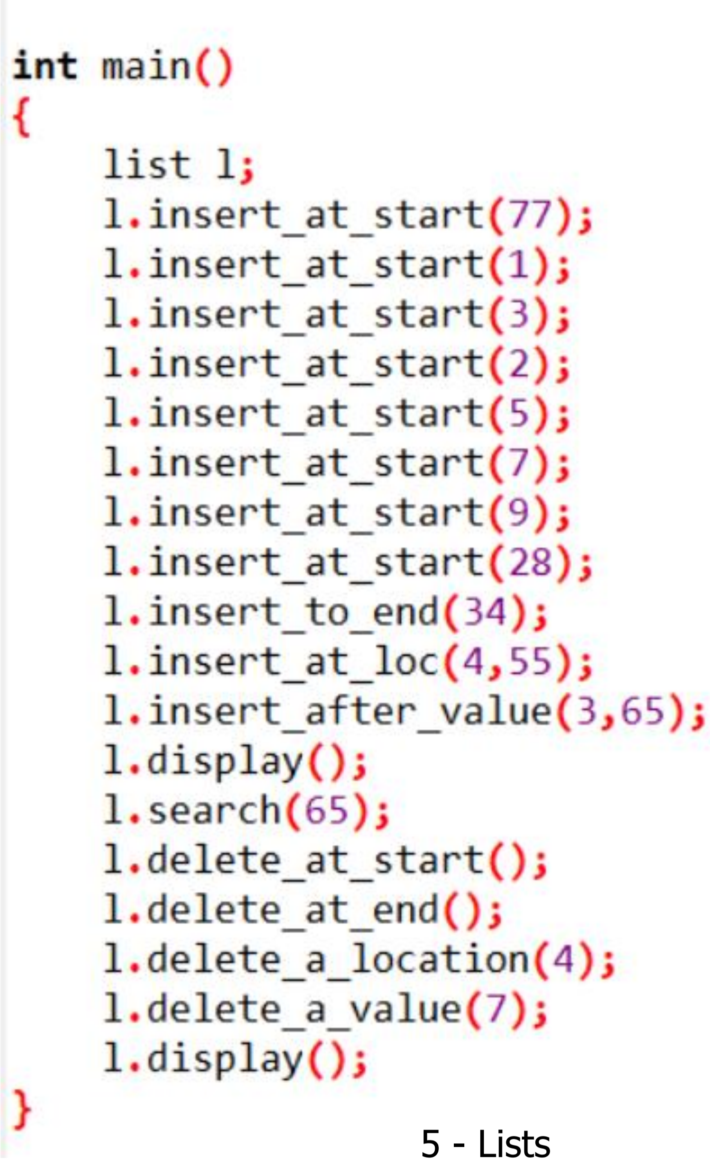
```
void search(int v)
{
    node *p=head;
    int c=0;
    while(p!=NULL)
    {
        c++;
        if(p->data==v)
        break;
        p=p->next;

    }
    if(p!=NULL)
    cout<<"datafoundat"<<c;
}
```

Print the list

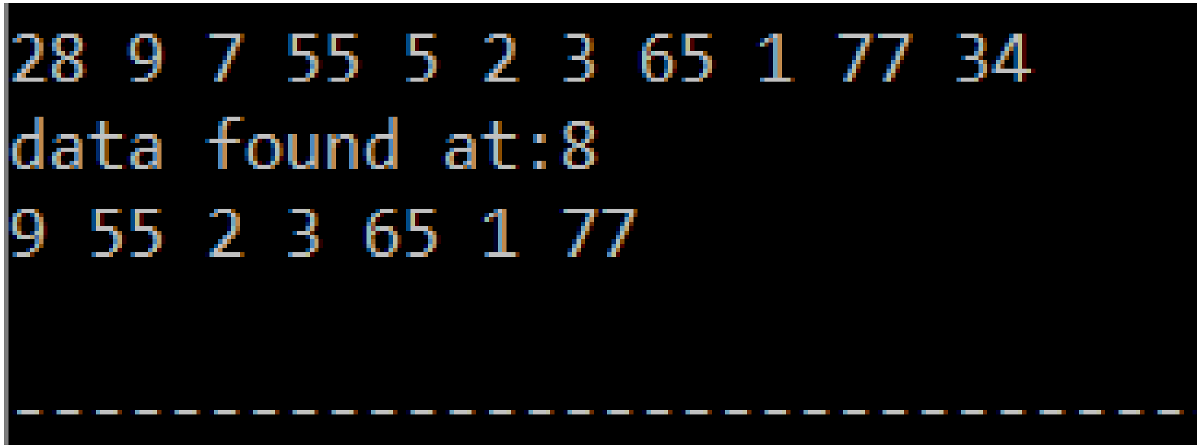


```
void display()
{
    if(head==NULL)
    cout<<"Empty";
    node *p=head;
    while(p!=NULL)
    {
        cout<<p->data<<" ";
        p=p->next;
    }
    cout<<endl;
}
```

Using List



```
int main()
{
    list l;
    l.insert_at_start(77);
    l.insert_at_start(1);
    l.insert_at_start(3);
    l.insert_at_start(2);
    l.insert_at_start(5);
    l.insert_at_start(7);
    l.insert_at_start(9);
    l.insert_at_start(28);
    l.insert_to_end(34);
    l.insert_at_loc(4,55);
    l.insert_after_value(3,65);
    l.display();
    l.search(65);
    l.delete_at_start();
    l.delete_at_end();
    l.delete_a_location(4);
    l.delete_a_value(7);
    l.display();
}
```



```
28 9 7 55 5 2 3 65 1 77 34
data found at:8
9 55 2 3 65 1 77



------------------------------
```

Any Question So Far?

Practice Task

- Write a C++ program to create a singly linked list of n nodes and count the number of nodes.
  - Original Linked list:
    13 11 9 7 5 3 1
    Number of nodes in the said Linked list:
    7
- Write a C++ program to find the middle element of a given Linked List.
  - Input List:
    5 7 2 9
    Middle element of the list:
    2
    Original list:
    5 6 2 9 4
    Middle element of the list:
    2