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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  public:
    Node* head;
  public:
    //constructor to create an empty LinkedList
    LinkedList(){
      head = NULL;
    }

    //display the content of the list
    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  //create an empty LinkedList
  LinkedList MyList;

  //Add first node.
  Node* first = new Node();
  first->data = 10;
  first->next = NULL;
  //linking with head node
  MyList.head = first;

  //Add second node.
  Node* second = new Node();
```

```
  second->data = 20;
  second->next = NULL;
  //linking with first node
  first->next = second;

  //Add third node.
  Node* third = new Node();
  third->data = 30;
  third->next = NULL;
  //linking with second node
  second->next = third;

  //print the content of list
  MyList.PrintList();
  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30
```

Linked List - Insert a new node at the start

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the start of the list
```

```
    void push_front(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = head;
      head = newNode;
    }

    //display the content of the list
    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add three elements at the start of the list.
  MyList.push_front(10);
  MyList.push_front(20);
  MyList.push_front(30);
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 30 20 10
```

Linked List - Insert a new node at the end

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    //display the content of the list
    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
```

```
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add three elements at the end of the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30
```

Linked List - Insert a new node at the given position

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
```

```
    Node* newNode = new Node();
    newNode->data = newElement;
    newNode->next = NULL;
    if(head == NULL) {
      head = newNode;
    } else {
      Node* temp = head;
      while(temp->next != NULL)
        temp = temp->next;
      temp->next = newNode;
    }
  }

  //Inserts a new element at the given position
  void push_at(int newElement, int position) {
    Node* newNode = new Node();
    newNode->data = newElement;
    newNode->next = NULL;

    if(position < 1) {
      cout<<"\nposition should be >= 1.";
    } else if (position == 1) {
      newNode->next = head;
      head = newNode;
    } else {

      Node* temp = head;
      for(int i = 1; i < position-1; i++) {
        if(temp != NULL) {
          temp = temp->next;
        }
      }

      if(temp != NULL) {
        newNode->next = temp->next;
        temp->next = newNode;
      } else {
        cout<<"\nThe previous node is null.";
      }
    }
  }

  //display the content of the list
  void PrintList() {
    Node* temp = head;
    if(temp != NULL) {
      cout<<"The list contains: ";
      while(temp != NULL) {
```

```
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add three elements in the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.PrintList();

  //Insert an element at position 2
  MyList.push_at(100, 2);
  MyList.PrintList();

  //Insert an element at position 1
  MyList.push_at(200, 1);
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30
The list contains: 10 100 20 30
The list contains: 200 10 100 20 30
```

Linked List - Delete the first node

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

//node structure
```

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

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    //Delete first node of the list
    void pop_front() {
      if(head != NULL) {
        Node* temp = head;
        head = head->next;
        free(temp);
      }
    }

    //display the content of the list
    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
```

```
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add four elements in the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(40);
  MyList.PrintList();

  //Delete the first node
  MyList.pop_front();
  MyList.PrintList();
  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 40
The list contains: 20 30 40
```

Linked List - Delete the last node

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
```

```
public:
  LinkedList(){
    head = NULL;
  }

  //Add new element at the end of the list
  void push_back(int newElement) {
    Node* newNode = new Node();
    newNode->data = newElement;
    newNode->next = NULL;
    if(head == NULL) {
      //for first element in the list
      head = newNode;
    } else {
      Node* temp = head;
      while(temp->next != NULL)
        temp = temp->next;
      temp->next = newNode;
    }
  }

  //Delete last node of the list
  void pop_back() {
    if(head != NULL) {
      if(head->next == NULL) {
        head = NULL;
      } else {
        Node* temp = head;
        while(temp->next->next != NULL)
          temp = temp->next;
        Node* lastNode = temp->next;
        temp->next = NULL;
        free(lastNode);
      }
    }
  }

  //display the content of the list
  void PrintList() {
    Node* temp = head;
    if(temp != NULL) {
      cout<<"The list contains: ";
      while(temp != NULL) {
        cout<<temp->data<<" ";
        temp = temp->next;
      }
      cout<<"\n";
    } else {
```

```
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add four elements in the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(40);
  MyList.PrintList();

  //Delete the last node
  MyList.pop_back();
  MyList.PrintList();
  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 40
The list contains: 10 20 30
```

Linked List - Delete a node at the given position

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
```

```
public:
  LinkedList(){
    head = NULL;
  }

  //Add new element at the end of the list
  void push_back(int newElement) {
    Node* newNode = new Node();
    newNode->data = newElement;
    newNode->next = NULL;
    if(head == NULL) {
      head = newNode;
    } else {
      Node* temp = head;
      while(temp->next != NULL)
        temp = temp->next;
      temp->next = newNode;
    }
  }

  //Delete an element at the given position
  void pop_at(int position) {
    if(position < 1) {
      cout<<"\nposition should be >= 1.";
    } else if (position == 1 && head != NULL) {
      Node* nodeToDelete = head;
      head = head->next;
      free(nodeToDelete);
    } else {
      Node* temp = head;
      for(int i = 1; i < position-1; i++) {
        if(temp != NULL) {
          temp = temp->next;
        }
      }
      if(temp != NULL && temp->next != NULL) {
          Node* nodeToDelete = temp->next;
          temp->next = temp->next->next;
          free(nodeToDelete);
      } else {
        cout<<"\nThe node is already null.";
      }
    }
  }

  //display the content of the list
  void PrintList() {
    Node* temp = head;
```

```
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add three elements at the end of the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.PrintList();

  //Delete an element at position 2
  MyList.pop_at(2);
  MyList.PrintList();

  //Delete an element at position 1
  MyList.pop_at(1);
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30
The list contains: 10 30
The list contains: 30
```

Linked List - Delete all nodes

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    //delete all nodes of the list
    void deleteAllNodes() {
      Node* temp = new Node();
      while(head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
      }
      cout<<"All nodes are deleted successfully.\n";
    }

    //display the content of the list
    void PrintList() {
```

```
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add four elements in the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(40);

  //Display the content of the list.
  MyList.PrintList();

  //delete all nodes of the list
  MyList.deleteAllNodes();

  //Display the content of the list.
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 40
All nodes are deleted successfully.
The list is empty.
```

Linked List - Count nodes

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    //count nodes in the list
    int countNodes() {
      Node* temp = head;
      int i = 0;
      while(temp != NULL) {
        i++;
        temp = temp->next;
      }
      return i;
    }
```

```
    //display the content of the list
    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add four elements in the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(40);

  //Display the content of the list.
  MyList.PrintList();

  //number of nodes in the list
  cout<<"No. of nodes: "<<MyList.countNodes();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 40
No. of nodes: 4
```

Linked List - Delete even nodes

```
#include <iostream>
```

```
using namespace std;

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    //delete even nodes of the list
    void deleteEvenNodes() {
      if(head != NULL) {
        Node* oddNode = head;
        Node* evenNode = head->next;
        while(oddNode != NULL && evenNode != NULL) {
          oddNode->next = evenNode->next;
          free(evenNode);
          oddNode = oddNode->next;
          if(oddNode != NULL)
            evenNode = oddNode->next;
        }
      }
    }

    //display the content of the list
    void PrintList() {
```

```
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add five elements in the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(40);
  MyList.push_back(50);

  //Display the content of the list.
  MyList.PrintList();

  //delete even nodes of the list
  MyList.deleteEvenNodes();

  cout<<"After deleting even nodes.\n";
  //Display the content of the list.
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 40 50
After deleting even nodes.
The list contains: 10 30 50
```

Linked List - Delete odd nodes

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    //delete odd nodes of the list
    void deleteOddNodes() {
      if(head != NULL) {
        Node* temp = head;
        head = head->next;
        free(temp);
        if(head != NULL) {
          Node* evenNode = head;
          Node* oddNode = head->next;
          while(evenNode != NULL && oddNode != NULL) {
            evenNode->next = oddNode->next;
            free(oddNode);
            evenNode = evenNode->next;
```

```
            if(evenNode != NULL)
              oddNode = evenNode->next;
          }
        }
      }
    }

    //display the content of the list
    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add five elements in the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(40);
  MyList.push_back(50);

  //Display the content of the list.
  MyList.PrintList();

  //delete odd nodes of the list
  MyList.deleteOddNodes();

  cout<<"After deleting odd nodes.\n";
  //Display the content of the list.
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 40 50
After deleting odd nodes.
The list contains: 20 40
```

Linked List - Search an element

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    //Search an element in the list
    void SearchElement(int searchValue) {
      Node* temp = head;
      int found = 0;
```

```
      int i = 0;

      if(temp != NULL) {
        while(temp != NULL) {
          i++;
          if(temp->data == searchValue) {
            found++;
            break;
          }
          temp = temp->next;
        }
        if (found == 1) {
          cout<<searchValue<<" is found at index = "<<i<<".\n";
        } else {
          cout<<searchValue<<" is not found in the list.\n";
        }
      } else {
        cout<<"The list is empty.\n";
      }
    }

    //display the content of the list
    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add three elements at the end of the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);

  //traverse to display the content of the list.
  MyList.PrintList();
```

```
  //search for element in the list
  MyList.SearchElement(10);
  MyList.SearchElement(15);
  MyList.SearchElement(20);

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30
10 is found at index = 1.
15 is not found in the list.
20 is found at index = 2.
```

Linked List - Delete first node by key

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
```

```
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    //Delete first node by key
    void pop_first(int key) {
      Node* temp = head;
      if(temp != NULL) {
        if(temp->data == key) {
          Node* nodeToDelete = head;
          head = head->next;
          free(nodeToDelete);
        } else {
          while(temp->next != NULL) {
            if(temp->next->data == key) {
              Node* nodeToDelete = temp->next;
              temp->next = temp->next->next;
              free(nodeToDelete);
              break;
            }
            temp = temp->next;
          }
        }
      }
    }

    //display the content of the list
    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;
```

```
  //Add five elements at the end of the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.PrintList();

  //Delete first occurrence of 20
  MyList.pop_first(20);
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 10 20
The list contains: 10 30 10 20
```

Linked List - Delete last node by key

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
```

```
    if(head == NULL) {
      head = newNode;
    } else {
      Node* temp = head;
      while(temp->next != NULL)
        temp = temp->next;
      temp->next = newNode;
    }
  }

  //Delete last node by key
  void pop_last(int key) {
    if(head != NULL) {
      Node *previousToLast, *lastNode, *temp;
      previousToLast = NULL;
      lastNode = NULL;

      if(head->data == key)
        lastNode = head;

      temp = head;
      while(temp->next != NULL) {
        if(temp->next->data == key) {
          previousToLast = temp;
          lastNode = temp->next;
        }
        temp = temp->next;
      }

      if(lastNode != NULL) {
        if(lastNode == head) {
          head = head->next;
          free(lastNode);
        } else {
          previousToLast->next = lastNode->next;
          free(lastNode);
        }
      }
    }
  }

  //display the content of the list
  void PrintList() {
    Node* temp = head;
    if(temp != NULL) {
      cout<<"The list contains: ";
      while(temp != NULL) {
        cout<<temp->data<<" ";
```

```
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add five elements at the end of the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(20);
  MyList.push_back(40);
  MyList.PrintList();

  //Delete last occurrence of 20
  MyList.pop_last(20);
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 20 40
The list contains: 10 20 30 40
```

Linked List - Delete all nodes by key

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

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
```

```
private:
  Node* head;
public:
  LinkedList(){
    head = NULL;
  }

  //Add new element at the end of the list
  void push_back(int newElement) {
    Node* newNode = new Node();
    newNode->data = newElement;
    newNode->next = NULL;
    if(head == NULL) {
      head = newNode;
    } else {
      Node* temp = head;
      while(temp->next != NULL)
        temp = temp->next;
      temp->next = newNode;
    }
  }

  //Delete all nodes by key
  void pop_all(int key) {
    Node* nodeToDelete;
    while(head != NULL && head->data == key) {
      nodeToDelete = head;
      head = head->next;
      free(nodeToDelete);
    }

    Node* temp = head;
    if(temp != NULL) {
      while(temp->next != NULL) {
        if(temp->next->data == key) {
          nodeToDelete = temp->next;
          temp->next = temp->next->next;
          free(nodeToDelete);
        } else {
          temp = temp->next;
        }
      }
    }
  }

  //display the content of the list
  void PrintList() {
    Node* temp = head;
```

```
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add five elements at the end of the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.PrintList();

  //Delete all occurrences of 20
  MyList.pop_all(20);
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 10 20
The list contains: 10 30 10
```

Linked List - Reverse the List

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

//node structure
struct Node {
    int data;
```

```
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    //reverse the list
    void reverseList() {
      if(head != NULL) {
        Node* prevNode = head;
        Node* tempNode = head;
        Node* curNode = head->next;

        prevNode->next = NULL;

        while(curNode != NULL) {
          tempNode = curNode->next;
          curNode->next = prevNode;
          prevNode = curNode;
          curNode = tempNode;
        }

        head = prevNode;
      }
    }

    //display the content of the list
    void PrintList() {
```

```
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add five elements in the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(40);
  MyList.push_back(50);

  //Display the content of the list.
  MyList.PrintList();

  //Reversing the list.
  MyList.reverseList();

  //Display the content of the list.
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 40 50
The list contains: 50 40 30 20 10
```

Linked List - Swap node values

```
#include <iostream>
```

```
using namespace std;

//node structure
struct Node {
    int data;
    Node* next;
};

class LinkedList {
  private:
    Node* head;
  public:
    LinkedList(){
      head = NULL;
    }

    //Add new element at the end of the list
    void push_back(int newElement) {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL) {
        head = newNode;
      } else {
        Node* temp = head;
        while(temp->next != NULL)
          temp = temp->next;
        temp->next = newNode;
      }
    }

    //swap node values
    void swapNodeValues(int node1, int node2) {

      Node* temp = head;
      int N = 0;
      while(temp != NULL) {
        N++;
        temp = temp->next;
      }

      if(node1 < 1 || node1 > N || node2 < 1 || node2 > N)
        return;

      Node* pos1 = head;
      Node* pos2 = head;
      for(int i = 1; i < node1; i++) {
        pos1 = pos1->next;
```

```
      }
      for(int i = 1; i < node2; i++) {
        pos2 = pos2->next;
      }

      int val = pos1->data;
      pos1->data = pos2->data;
      pos2->data = val;
    }

    //display the content of the list
    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<"\n";
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  LinkedList MyList;

  //Add five elements in the list.
  MyList.push_back(10);
  MyList.push_back(20);
  MyList.push_back(30);
  MyList.push_back(40);
  MyList.push_back(50);

  //Display the content of the list.
  MyList.PrintList();

  //swap values of node=1 and node=4
  MyList.swapNodeValues(1, 4);

  //Display the content of the list.
  MyList.PrintList();

  return 0;
}
```

The above code will give the following output:

```
The list contains: 10 20 30 40 50
The list contains: 40 20 30 10 50
```

Traverse a Linked List

Traversing through a linked list is very easy. It requires creating a temp node pointing to the head of the list. If the temp node is not null, display its content and move to the next node using temp next. Repeat the process till the temp node becomes null. If the temp node is empty at the start, then the list contains no item.

```
//C++ Code
//Display the content of the list
void PrintList() {

  //1. create a temp node pointing to head
  Node* temp = head;

  //2. if the temp node is not null continue
  //   displaying the content and move to the
  //   next node till the temp becomes null
  if(temp != NULL) {
    cout<<"\nThe list contains: ";
    while(temp != NULL) {
      cout<<temp->data<<" ";
      temp = temp->next;
    }
  } else {

    //3. If the temp node is null at the start,
    //   the list is empty
    cout<<"\nThe list is empty.";
  }
}
```

Insert a new node in Linked List

A new node can be inserted into a list in three ways:

- Insert a node at the start

- Insert a node at the given position
- Insert a node at the end

Insert a new node at the start

In this method, a new node is inserted at the beginning of the linked list. For example - if the given list is 10->20->30 and a new element 100 is added at the start, the list becomes 100->10->20->30.

Inserting a new node at the beginning of the Linked List is very easy. First, a new node with given element is created. It is then added before the head of the given list that makes the newly added node to new head of the list by changing the head pointer to point to the new node.

```
//C++ Code
//Inserts a new node at the start
void push_front(int newElement) {

  //1. allocate a new node
  Node* newNode = new Node();

  //2. assign data element
  newNode->data = newElement;

  //3. make next node of new node as head
  newNode->next = head;

  //4. make new node as head
  head = newNode;
}
```

Insert a new node at the given position

In this method, a new element is inserted at the specified position in the linked list. For example - if the given list is 10->20->30 and a new element 100 is added at position 2, the list becomes 10->100->20->30.

First, a new node with given element is created. If the insert position is 1, then the new node is made to head. Otherwise, traverse to the node that is previous to the insert position and check if it is null or not. In case of null, the specified position does not exist. In other case, assign next of the new node as next of the previous node and next of previous node as new node.

```
//C++ Code
//Inserts a new node at the given position
void push_at(int newElement, int position) {

  //1. allocate node to new element
```

```
  Node* newNode = new Node();
  newNode->data = newElement;
  newNode->next = NULL;

  //2. check if the position is > 0
  if(position < 1) {
    cout<<"\nposition should be >= 1.";
  } else if (position == 1) {

  //3. if the position is 1, make next of the
  //   new node as head and new node as head
    newNode->next = head;
    head = newNode;
  } else {

   //4. Else, make a temp node and traverse to the
   //   node previous to the position
    Node* temp = head;
    for(int i = 1; i < position-1; i++) {
      if(temp != NULL) {
        temp = temp->next;
      }
    }

    //5. If the previous node is not null, make
    //   newNode next as temp next and temp next
    //   as newNode.
    if(temp != NULL) {
      newNode->next = temp->next;
      temp->next = newNode;
    } else {

      //6. When the previous node is null
      cout<<"\nThe previous node is null.";
    }
  }
}
```

Insert a new node at the end

In this method, a new node is inserted at the end of the linked list. For example - if the given list is 10->20->30 and a new element 100 is added at the end, the list becomes 10->20->30->100.

Inserting a new node at the end of the Linked List is very easy. First, a new node with given element is created. It is then added at the end of the list by linking the last node to the new node.

```
//C++ Code
//Inserts a new node at the end
void push_back(int newElement) {

  //1. allocate node
  Node* newNode = new Node();

  //2. assign data element
  newNode->data = newElement;

  //3. assign null to the next of new node
  newNode->next = NULL;

  //4. Check the list is empty or not,
  //   if empty make the new node as head
  if(head == NULL) {
    head = newNode;
  } else {

    //5. Else, traverse to the last node
    Node* temp = head;
    while(temp->next != NULL)
      temp = temp->next;

    //6. Change the next of last node to new node
    temp->next = newNode;
  }
}
```

Delete a Node from Linked List

A node can be deleted from a list in three ways:

- Delete the first node
- Delete the node at given position
- Delete the last node

Delete the first node
In this method, the first node of the linked list is deleted. For example - if the given list is 10->20->30->40 and the first node is deleted, the list becomes 20->30->40.

Deleting the first node of the linked list is very easy. If the head is not null then create a temp node pointing to head and move head to the next of head. Then delete the temp node.

```
//C++ Code
//Deletes the first node
```

```
void pop_front() {
  if(head != NULL) {

    //1. if head is not null, create a
    //   temp node pointing to head
    Node* temp = head;

    //2. move head to next of head
    head = head->next;

    //3. delete temp node
    free(temp);
  }
}
```

Delete a node from middle

In this method, a node at the specified position in the linked list is deleted. For example - if the given list is 10->20->30 and the 2nd node is deleted, the list becomes 10->20.

First, the specified position must be greater than equal to 1. If the specified position is 1 and head is not null, then make the head to head next. Else, traverse to the node that is previous to the specified position. If the specified node and previous to the specified node are not null then adjust the link and delete the node at specified position. In other case, the specified node will be already null.

```
//C++ Code
//Deletes the node at given position
void pop_at(int position) {

  //1. check if the position is > 0
  if(position < 1) {
    cout<<"\nposition should be >= 1.";
  } else if (position == 1 && head != NULL) {

    //2. if the position is 1 and head is not null,
    //   make head to head next and delete the
    //   previous head
    Node* nodeToDelete = head;
    head = head->next;
    free(nodeToDelete);
  } else {

    //3. Else, make a temp node and traverse to the
    //   node previous to the position
    Node* temp = head;
    for(int i = 1; i < position-1; i++) {
```

```
      if(temp != NULL) {
        temp = temp->next;
      }
    }

    //4. If the previous node and next of the previous
    //   is not null, adjust links
    if(temp != NULL && temp->next != NULL) {
        Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
        free(nodeToDelete);
    } else {

      //5. Else the given node will be empty.
      cout<<"\nThe node is already null.";
    }
  }
}
```

Delete the last node

In this method, the last node of the linked list is deleted. For example - if the given list is 10->20->30->40 and the last node is deleted, the list becomes 10->20->30.

Deleting the last node of the Linked List involves checking the head for empty. If it is not empty, then check the head next for empty. If the head next is empty, then release the head, else traverse to the second last node of the list. Then, link the next of second last node to NULL and delete the last node.

```
//C++ Code
//Deletes the last node
void pop_back() {
  if(head != NULL) {

    //1. if head in not null and next of head
    //   is null, release the head
    if(head->next == NULL) {
      head = NULL;
    } else {

      //2. Else, traverse to the second last
      //   element of the list
      Node* temp = head;
      while(temp->next->next != NULL)
        temp = temp->next;

      //3. Change the next of the second
```

```
      //   last node to null and delete the
      //   last node
      Node* lastNode = temp->next;
      temp->next = NULL;
      free(lastNode);
    }
  }
}
```