In [ ]:

```
#Adding a new node,
#include <iostream>
using namespace std;
struct Node {
   int data;
   struct Node *next;
};
struct Node* head = NULL;
void insert(int new_data) {
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = head;
   head = new_node;
}
void display() {
   struct Node* ptr;
   ptr = head;
   while (ptr != NULL) {
      cout<< ptr->data <<" ";
      ptr = ptr->next;
   }
}
int main() {
   insert(3);
   insert(1);
   insert(7);
   insert(2);
   insert(9);
   cout<<"The linked list is: ";
   display();
   return 0;
}

#Deleting a particular node (referenced by the location),
#Delete all the nodes from the list which contain a particular data say a number 5
#include <iostream>
using namespace std;

// A linked list node
class Node {
public:
    int data;
    Node* next;
};

// Given a reference (pointer to pointer) to
// the head of a list and an int inserts a
// new node on the front of the list.
void push(Node** head_ref, int new_data)
{
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

// Given a reference (pointer to pointer) to
// the head of a list and a position, deletes
// the node at the given position
void deleteNode(Node** head_ref, int position)
{

    // If linked list is empty
    if (*head_ref == NULL)
        return;

    // Store head node
    Node* temp = *head_ref;

    // If head needs to be removed
    if (position == 0) {

        // Change head
        *head_ref = temp->next;

        // Free old head
        free(temp);
        return;
    }

    // Find previous node of the node to be deleted
    for (int i = 0; temp != NULL && i < position - 2; i++)
        temp = temp->next;

    // If position is more than number of nodes
    if (temp == NULL || temp->next == NULL)
        return;

    // Node temp->next is the node to be deleted
    // Store pointer to the next of node to be deleted
    Node* next = temp->next->next;

    // Unlink the node from linked list
    free(temp->next); // Free memory

    // Unlink the deleted node from list
    temp->next = next;
}

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

// Driver code
int main()
{

    // Start with the empty list
    Node* head = NULL;

    push(&head, 7);
    push(&head, 1);
    push(&head, 3);
    push(&head, 2);
    push(&head, 8);

    cout << "Created Linked List: ";
    printList(head);
    deleteNode(&head, 4);
    cout << "\nLinked List after Deletion at position 4: ";
    printList(head);
    return 0;
}


#Delete the complete linked list
#include <iostream>
#include <vector>
using namespace std;

// A Linked List Node
class Node
{
  public:
    int data;                  // data field
    Node* next = nullptr;     // pointer to the next node

    Node() {}
    Node(int data): data(data) {}
    Node(int data, Node *next): data(data), next(next) {}
};

// Helper function to create a new node with the given data and
// pushes it onto the list's front
void push(Node* &head, int data)
{
    Node* newNode = new Node(data);
    newNode->next = head;
    head = newNode;
}

// Iterative function to delete a linked list
void deleteList(Node* &head)
{
    Node* prev = head;

    while (head)
    {
        head = head->next;

        cout << "Deleting " << prev->data << endl;
        delete(prev);
        prev = head;
    }
}

int main()
{
    // input keys
    vector<int> keys = { 1, 2, 3, 4, 5 };
    int n = keys.size();

    // points to the head node of the linked list
    Node* head = nullptr;

    // construct a linked list
    for (int i = n - 1; i >= 0; i--) {
        push(head, keys[i]);
    }

    deleteList(head);

    if (head == nullptr) {
        cout << "List deleted";
    }

    return 0;
}

#Display the linked list
#include <iostream>
using namespace std;
struct Node {
   int data;
   struct Node *next;
};
struct Node* head = NULL;
void insert(int new_data) {
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = head;
   head = new_node;
}
void display() {
   struct Node* ptr;
   ptr = head;
   while (ptr != NULL) {
      cout<< ptr->data <<" ";
      ptr = ptr->next;
   }
}
int main() {
   insert(3);
   insert(1);
   insert(7);
   insert(2);
   insert(9);
   cout<<"The linked list is: ";
   display();
   return 0;
}

#Display the inverted linked list
class Node:

    # Constructor to initialize the node object
    def __init__(self, data):
        self.data = data
        self.next = None


class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None

    # Function to reverse the linked list
    def reverse(self):
        prev = None
        current = self.head
        while(current is not None):
            next = current.next
            current.next = prev
            prev = current
            current = next
        self.head = prev

    # Function to insert a new node at the beginning
    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node

    # Utility function to print the linked LinkedList
    def printList(self):
        temp = self.head
        while(temp):
            print temp.data,
            temp = temp.next


# Driver code
llist = LinkedList()
llist.push(20)
llist.push(4)
llist.push(15)
llist.push(85)

print "Given Linked List"
llist.printList()
llist.reverse()
print "\nReversed Linked List"
llist.printList()
```