Creating and Displaying Linked List

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

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

// Function to create a new node
Node* createNode(int value) {
   Node* newNode = new Node();
   newNode->data = value;
   newNode->next = nullptr;  // No next node initially
   return newNode;
}

// Function to display the linked list
void display(Node* head) {
   Node* temp = head;
   while (temp != nullptr) {
      cout << temp->data << " -> ";
      temp = temp->next;
   }
   cout << "nullptr" << endl;
}

int main() {
   // Create nodes
   Node* head = createNode(10);  // First node
   head->next = createNode(20);  // Second node
   head->next->next = createNode(30);  // Third node

   // Display linked list
   display(head);  // Output: 10 -> 20 -> 30 -> nullptr

   return 0;
}
```

Inserting Node

```
void insertAtBeginning(Node*& head, int data) {
   Node* newNode = new Node();
   newNode->data = data;
   newNode->next = head;
   head = newNode;
}
// Insert node at the middle of the linked list
void insertInMiddle(Node*& head, int data, int position) {
   Node* newNode = new Node();
   newNode->data = data;

   if (position == 1) {
      newNode->next = head;
      head = newNode;
      return;
   }

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

   if (temp == nullptr) {
      cout << "Position out of bounds\n";
      return;
   }

   newNode->next = temp->next;
   temp->next = newNode;
}
void insertAtEnd(Node*& head, int value) {
   Node* newNode = createNode(value);
   if (head == nullptr) {
      head = newNode;
      return;
   }
   Node* temp = head;
   while (temp->next != nullptr) {
      temp = temp->next;
   }
   temp->next = newNode;
}
```

Deletion

```
void deleteNode(Node*& head, int value) {
    if (head == nullptr) return;
    if (head->data == value) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr && temp->next->data != value) {
        temp = temp->next;
    }
    if (temp->next == nullptr) return;  // Value not found
    Node* toDelete = temp->next;
    temp->next = toDelete->next;
    delete toDelete;
}
```

Searching

```
bool search(Node* head, int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) return true;
        temp = temp->next;
    }
    return false;
}
```

Sorting

```
void sortList(Node* head) {
    Node* current = head;
    Node* index = nullptr;
    int temp;

    if (head == nullptr) return;  // List is empty

    while (current != nullptr) {
        index = current->next;
        while (index != nullptr) {
            if (current->data > index->data) {
                temp = current->data;
                current->data = index->data;
                index->data = temp;
            }
            index = index->next;
        }
        current = current->next;
    }
}
```

Doubly Linked List

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

// Node structure for doubly linked list
struct Node {
   int data;
   Node* next;
   Node* prev;
};
// Insert node at the beginning of a doubly linked list
void insertAtBeginning(DoublyNode*& head, int data) {
   DoublyNode* newNode = new DoublyNode();
   newNode->data = data;
   newNode->prev = nullptr;
   newNode->next = head;

   if (head != nullptr) {
      head->prev = newNode;
   }

   head = newNode;
}
// Insert node in the middle of a doubly linked list
void insertInMiddle(DoublyNode*& head, int data, int position) {
   DoublyNode* newNode = new DoublyNode();
   newNode->data = data;

   if (position == 1) {
      newNode->next = head;
      newNode->prev = nullptr;
      if (head != nullptr) head->prev = newNode;
      head = newNode;
      return;
   }

   DoublyNode* temp = head;
   for (int i = 1; i < position - 1 && temp != nullptr; i++) {
      temp = temp->next;
   }

   if (temp == nullptr) {
      cout << "Position out of bounds\n";
      return;
   }
```

```
    newNode->next = temp->next;
    newNode->prev = temp;

    if (temp->next != nullptr) {
        temp->next->prev = newNode;
    }

    temp->next = newNode;
}
// Function to insert at the end of a doubly linked list
void insertAtEnd(Node*& head, int value) {
    Node* newNode = createNode(value);
    if (head == nullptr) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;  // Link new node to the previous one
}

// Function to display the doubly linked list
void display(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " <-> ";
        temp = temp->next;
    }
    cout << "nullptr" << endl;
}

int main() {
    Node* head = nullptr;
    insertAtEnd(head, 10);
    insertAtEnd(head, 20);
    insertAtEnd(head, 30);

    display(head);  // Output: 10 <-> 20 <-> 30 <-> nullptr

    return 0;
}
```

Circular Linked list

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

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

// Function to create a new node
Node* createNode(int value) {
   Node* newNode = new Node();
   newNode->data = value;
   newNode->next = nullptr;
   return newNode;
}
void insertAtBeginning(CircularNode*& head, int data) {
   CircularNode* newNode = new CircularNode();
   newNode->data = data;

   if (head == nullptr) {
      newNode->next = newNode;
      head = newNode;
      return;
   }

   CircularNode* temp = head;
   while (temp->next != head) {
      temp = temp->next;
   }

   newNode->next = head;
   temp->next = newNode;
   head = newNode;
}
// Insert node in the middle of a circular linked list
void insertInMiddle(CircularNode*& head, int data, int position) {
   CircularNode* newNode = new CircularNode();
   newNode->data = data;

   if (position == 1) {
      insertAtBeginning(head, data);
      return;
   }
```

```
    CircularNode* temp = head;
    for (int i = 1; i < position - 1 && temp->next != head; i++) {
        temp = temp->next;
    }

    newNode->next = temp->next;
    temp->next = newNode;
}
// Function to insert at the end of a circular linked list
void insertAtEnd(Node*& head, int value) {
    Node* newNode = createNode(value);
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;  // Circular link
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = head;  // Circular link
}

// Function to display the circular linked list
void display(Node* head) {
    if (head == nullptr) return;
    Node* temp = head;
    do {
        cout << temp->data << " -> ";
        temp = temp->next;
    } while (temp != head);
    cout << "Head" << endl;
}

int main() {
    Node* head = nullptr;
    insertAtEnd(head, 10);
    insertAtEnd(head, 20);
    insertAtEnd(head, 30);

    display(head);  // Output: 10 -> 20 -> 30 -> Head

    return 0;
}
```

Doubly Circular Linked List

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

// Node structure for doubly circular linked list
struct Node {
    int data;
    Node* next;
    Node* prev;
};

// Function to create a new node
Node* createNode(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    newNode->prev = nullptr;
    return newNode;
}

// Function to insert at the end of a doubly circular linked list
void insertAtEnd(Node*& head, int value) {
    Node* newNode = createNode(value);
    if (head == nullptr) {
        head = newNode;
        head->next = head;
        head->prev = head;
        return;
    }
    Node* tail = head->prev;
    tail->next = newNode;
    newNode->prev = tail;
    newNode->next = head;
    head->prev = newNode;
}

// Function to display the doubly circular linked list
void display(Node* head) {
    if (head == nullptr) return;
    Node* temp = head;
    do {
        cout << temp->data << " <-> ";
        temp = temp->next;
    } while (temp != head);
    cout << "Head" << endl;
}
```

```
int main() {
   Node* head = nullptr;
   insertAtEnd(head, 10);
   insertAtEnd(head, 20);
   insertAtEnd(head, 30);

   display(head);  // Output: 10 <-> 20 <-> 30 <-> Head

   return 0;
}
```