LIMKED LIST PROBLEMS BY SAZIA MA'AM

Implement a program that allows users to perform the following operations on a singly linked list:

1. Insert at first
2. Insert at last
3. Insert after any node in the list
4. Delete the first item
5. Delete the last item
6. Delete any item from the list

For each operation, example input and output:

1. Insert at first:

● Input: Linked list: 1 -> 2 -> 3 -> 4, Value to insert: 0

● Output: Linked list: 0 -> 1 -> 2 -> 3 -> 4

2. Insert at last:

● Input: Linked list: 1 -> 2 -> 3 -> 4, Value to insert: 5

● Output: Linked list: 1 -> 2 -> 3 -> 4 -> 5

3. Insert after any node in the list:

● Input: Linked list: 1 -> 2 -> 3 -> 4,

○ Node to insert after Node with value: 2, Value to insert: 2.5

● Output: Linked list: 1 -> 2 -> 2.5 -> 3 -> 4

4. Delete the first item:

● Input: Linked list: 1 -> 2 -> 3 -> 4

● Output: Linked list: 2 -> 3 -> 4

5. Delete last item:

● Input: Linked list: 1 -> 2 -> 3 -> 4

● Output: Linked list: 1 -> 2 -> 3

6. Delete any item from the list:

● Input: Linked list: 1 -> 2 -> 3 -> 4, Node to delete: Node with value 2

● Output: Linked list: 1 -> 3 -> 4

Note: The program should handle cases where the list is empty or the specified node does not exist.

SOLN:

```
#include <iostream>

class Node {
public:
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    void insertAtFirst(int value) {
        Node* newNode = new Node(value);
        newNode->next = head;
        head = newNode;
    }

    void insertAtLast(int value) {
```

```
    Node* newNode = new Node(value);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* current = head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void insertAfterNode(int afterValue, int value) {
    Node* newNode = new Node(value);
    Node* current = head;
    while (current != nullptr) {
        if (current->data == afterValue) {
            newNode->next = current->next;
            current->next = newNode;
            break;
        }
        current = current->next;
    }
}

void deleteFirst() {
    if (head != nullptr) {
        Node* temp = head;
        head = head->next;
```

```
        delete temp;
    }
}

void deleteLast() {
    if (head != nullptr) {
        if (head->next == nullptr) {
            delete head;
            head = nullptr;
        } else {
            Node* current = head;
            while (current->next->next != nullptr) {
                current = current->next;
            }
            delete current->next;
            current->next = nullptr;
        }
    }
}

void deleteNode(int value) {
    if (head != nullptr) {
        if (head->data == value) {
            deleteFirst();
        } else {
            Node* current = head;
            while (current->next != nullptr) {
                if (current->next->data == value) {
                    Node* temp = current->next;
```

```
                current->next = current->next->next;
                delete temp;
                break;
            }
            current = current->next;
        }
    }
  }

  void display() {
    Node* current = head;
    while (current != nullptr) {
      std::cout << current->data << " -> ";
      current = current->next;
    }
    std::cout << "nullptr" << std::endl;
  }
};

int main() {
  LinkedList list;

  list.insertAtFirst(4);
  list.insertAtFirst(3);
  list.insertAtFirst(2);
  list.insertAtFirst(1);

  list.display();
```

```
    list.insertAtFirst(0);
    list.display();

    list.insertAtLast(5);
    list.display();

    list.insertAfterNode(2, 2.5);
    list.display();

    list.deleteFirst();
    list.display();

    list.deleteLast();
    list.display();

    list.deleteNode(2);
    list.display();

    return 0;
}
```

Formulate a program to take two linked lists as input and make a new link list with the average value of each index of those two linked lists.

Sample Input

Size of the list 1: 3

Items in List 1: 1 2 3

Size of List 2: 3

Items in List 2: 1 2 3

Sample Output

Output: 1 2 3

Sample Input

Size of the list 1: 5

Items in List 1: 10 20 30 40 50

Size of List 2: 6

Items in List 2: 20 40 60 80 100 120

Sample Output

Output: 15 30 45 60 75 60

SOLN:

```
#include <iostream>

class Node {
public:
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

class LinkedList {
```

```
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    void insertAtLast(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* current = head;
            while (current->next != nullptr) {
                current = current->next;
            }
            current->next = newNode;
        }
    }

    void calculateAverage(LinkedList& list1, LinkedList& list2) {
        Node* current1 = list1.head;
        Node* current2 = list2.head;

        while (current1 != nullptr && current2 != nullptr) {
            int average = (current1->data + current2->data) / 2;
            insertAtLast(average);

            current1 = current1->next;
            current2 = current2->next;
```

```
        }
    }

    void display() {
        Node* current = head;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }
};

int main() {
    int size1, size2;
    std::cout << "Size of List 1: ";
    std::cin >> size1;

    LinkedList list1;
    int item;
    std::cout << "Items in List 1: ";
    for (int i = 0; i < size1; i++) {
        std::cin >> item;
        list1.insertAtLast(item);
    }

    std::cout << "Size of List 2: ";
    std::cin >> size2;
```

```
    LinkedList list2;
    std::cout << "Items in List 2: ";
    for (int i = 0; i < size2; i++) {
        std::cin >> item;
        list2.insertAtLast(item);
    }

    LinkedList resultList;
    resultList.calculateAverage(list1, list2);

    std::cout << "Output: ";
    resultList.display();

    return 0;
}
```

Formulate a program to take two linked lists and merge them in another linked list in sorted order.

Sample Input

Size of the list 1: 5

Items in List 1: 1 9 2 4 10

Size of List 2: 3

Items in List 2: 3 1 5

Sample Output

Output: 1 2 3 4 5 9 10

Sample Input

Size of the list 1: 3

Items in List 1: 90 60 40

Size of List 2: 2

Items in List 2: 10 20

Sample Output

Output: 10 20 40 60 90

Soln:

```
#include <iostream>

class Node {
public:
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    void insertAtLast(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* current = head;
            while (current->next != nullptr) {
                current = current->next;
            }
```

```
        current->next = newNode;
    }
}

void mergeSorted(LinkedList& list1, LinkedList& list2) {
    Node* current1 = list1.head;
    Node* current2 = list2.head;

    while (current1 != nullptr && current2 != nullptr) {
        if (current1->data < current2->data) {
            insertAtLast(current1->data);
            current1 = current1->next;
        } else {
            insertAtLast(current2->data);
            current2 = current2->next;
        }
    }

    while (current1 != nullptr) {
        insertAtLast(current1->data);
        current1 = current1->next;
    }

    while (current2 != nullptr) {
        insertAtLast(current2->data);
        current2 = current2->next;
    }
}
```

```
    void display() {
        Node* current = head;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }
};

int main() {
    int size1, size2;
    std::cout << "Size of List 1: ";
    std::cin >> size1;

    LinkedList list1;
    int item;
    std::cout << "Items in List 1: ";
    for (int i = 0; i < size1; i++) {
        std::cin >> item;
        list1.insertAtLast(item);
    }

    std::cout << "Size of List 2: ";
    std::cin >> size2;

    LinkedList list2;
    std::cout << "Items in List 2: ";
    for (int i = 0; i < size2; i++) {
```

```
        std::cin >> item;
        list2.insertAtLast(item);
    }

    LinkedList resultList;
    resultList.mergeSorted(list1, list2);

    std::cout << "Output: ";
    resultList.display();

    return 0;
}
```

Formulate a program to take a linked list and delete the odd values.

Sample Input

Size of the list: 6

Items: 11 12 13 14 15 16

Sample Output

Number of remaining items: 3

Items: 12 14 16

Sample Input

Size of the list: 3

Items: 31 41 43

Sample Output

Number of remaining items: 0

Items: NONE

```
#include <iostream>

class Node {
public:
```

```
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    void insertAtLast(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* current = head;
            while (current->next != nullptr) {
                current = current->next;
            }
            current->next = newNode;
        }
    }

    void deleteOddValues() {
        Node* current = head;
        Node* prev = nullptr;
```

```
        while (current != nullptr) {
            if (current->data % 2 != 0) {
                if (prev == nullptr) {
                    head = current->next;
                } else {
                    prev->next = current->next;
                }
                Node* temp = current;
                current = current->next;
                delete temp;
            } else {
                prev = current;
                current = current->next;
            }
        }
    }

    void display() {
        Node* current = head;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }
};

int main() {
```

```
    int size;
    std::cout << "Size of the list: ";
    std::cin >> size;

    LinkedList list;
    int item;
    std::cout << "Items: ";
    for (int i = 0; i < size; i++) {
        std::cin >> item;
        list.insertAtLast(item);
    }

    list.deleteOddValues();

    std::cout << "Number of remaining items: " << size - list.getSize() << std::endl;
    std::cout << "Items: ";
    if (list.getSize() > 0) {
        list.display();
    } else {
        std::cout << "NONE" << std::endl;
    }

    return 0;
}
```

Formulate a program to Delete duplicate values from the list.

Sample Input

Size of the list: 10

Items: 1 2 3 4 5 2 6 3 7 8

Sample Output

Number of remaining items: 8

Items: 1 2 3 4 5 6 7 8

Sample Input

Size of the list: 6

Items: 1 2 2 1 4 4

Sample Output

Number of remaining items: 0

Items: NONE

Soln:

```
#include <iostream>
#include <unordered_set>

class Node {
public:
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

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

```
public:
    LinkedList() : head(nullptr) {}

    void insertAtLast(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* current = head;
            while (current->next != nullptr) {
                current = current->next;
            }
            current->next = newNode;
        }
    }

    void deleteDuplicates() {
        if (head == nullptr) {
            return;
        }

        std::unordered_set<int> seen;
        Node* current = head;
        Node* prev = nullptr;

        while (current != nullptr) {
            if (seen.find(current->data) != seen.end()) {
                prev->next = current->next;
                Node* temp = current;
```

```
            current = current->next;
            delete temp;
        } else {
            seen.insert(current->data);
            prev = current;
            current = current->next;
        }
    }
}

void display() {
    Node* current = head;
    while (current != nullptr) {
        std::cout << current->data << " ";
        current = current->next;
    }
    std::cout << std::endl;
}
};

int main() {
    int size;
    std::cout << "Size of the list: ";
    std::cin >> size;

    LinkedList list;
    int item;
    std::cout << "Items: ";
    for (int i = 0; i < size; i++) {
```

```
        std::cin >> item;
        list.insertAtLast(item);
    }

    list.deleteDuplicates();

    std::cout << "Number of remaining items: " << size - list.getSize() << std::endl;
    std::cout << "Items: ";
    if (list.getSize() > 0) {
        list.display();
    } else {
        std::cout << "NONE" << std::endl;
    }

    return 0;
}
```

Formulate a program to take two linked list as input. The program shall delete all the elements of the first linked list, which is contained in the second linked list.

Sample Input

Size of the list 1: 5

Items in List 1: 1 9 2 4 10

Size of List 2: 3

Items in List 2: 4 1 9

Sample Output

After Deletion:

List 1: 2 10

List 2: 4 1 9

Sample Input

Size of the list 1: 3

Items in List 1: 2 4 6

Size of List 2: 3

Items in List 2: 1 3 5

Sample Output

No elements of the first linked list are contained

in the second linked list.

Soln:

```
#include <iostream>
#include <unordered_set>

class Node {
public:
   int data;
   Node* next;

   Node(int value) : data(value), next(nullptr) {}
};

class LinkedList {
private:
   Node* head;

public:
   LinkedList() : head(nullptr) {}

   void insertAtLast(int value) {
      Node* newNode = new Node(value);
      if (head == nullptr) {
         head = newNode;
```

```
    } else {
        Node* current = head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void deleteCommonElements(const LinkedList& otherList) {
    if (head == nullptr || otherList.head == nullptr) {
        return;
    }

    std::unordered_set<int> elementsInSecondList;
    Node* current = otherList.head;
    while (current != nullptr) {
        elementsInSecondList.insert(current->data);
        current = current->next;
    }

    Node* prev = nullptr;
    current = head;

    while (current != nullptr) {
        if (elementsInSecondList.find(current->data) != elementsInSecondList.end()) {
            if (prev == nullptr) {
                head = current->next;
            } else {
```

```
                prev->next = current->next;
            }
            Node* temp = current;
            current = current->next;
            delete temp;
        } else {
            prev = current;
            current = current->next;
        }
    }
}

void display() {
    Node* current = head;
    while (current != nullptr) {
        std::cout << current->data << " ";
        current = current->next;
    }
    std::cout << std::endl;
}
};

int main() {
    int size1, size2;
    std::cout << "Size of the list 1: ";
    std::cin >> size1;

    LinkedList list1;
    int item;
```

```
    std::cout << "Items in List 1: ";
    for (int i = 0; i < size1; i++) {
        std::cin >> item;
        list1.insertAtLast(item);
    }

    std::cout << "Size of List 2: ";
    std::cin >> size2;

    LinkedList list2;
    std::cout << "Items in List 2: ";
    for (int i = 0; i < size2; i++) {
        std::cin >> item;
        list2.insertAtLast(item);
    }

    list1.deleteCommonElements(list2);

    std::cout << "After Deletion:" << std::endl;
    std::cout << "List 1: ";
    list1.display();
    std::cout << "List 2: ";
    list2.display();

    return 0;
}
```