Lab Assignment 5 Single Linked List

Shefali Sharma

1024030284

2C35

1. Develop a menu driven program for the following operations of on a Singly Linked List. (a) Insertion at the beginning. (b) Insertion at the end. (c) Insertion in between (before or after a node having a specific value, say 'Insert a new Node 35 before/after the Node 30'). (d) Deletion from the beginning. (e) Deletion from the end. (f) Deletion of a specific node, say 'Delete Node 60'). (g) Search for a node and display its position from head. (h) Display all the node values.

A1)

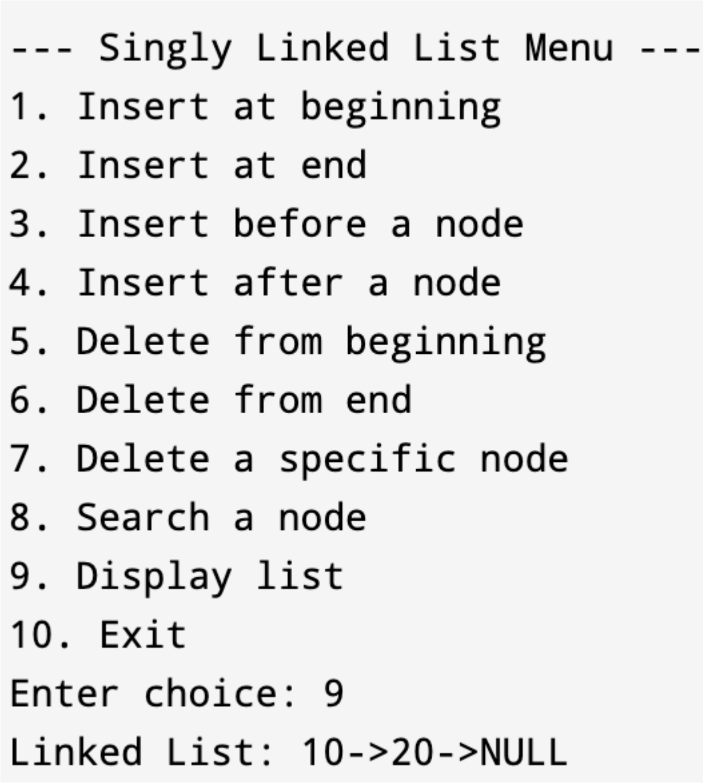
```
#include <iostream>
using namespace std;

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

class LinkedList {
  Node* head;

public:
  LinkedList() { head = NULL; }

  // Insertion at beginning
  void insertAtBeginning(int val) {
    Node* newNode = new Node{val, head};
    head = newNode;
```

```
}

// Insertion at end
void insertAtEnd(int val) {
    Node* newNode = new Node{val, NULL};
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = newNode;
}

// Insertion before/after a specific value
void insertAtPosition(int key, int val, bool before = true) {
    if (!head) return;

    if (before && head->data == key) {
        insertAtBeginning(val);
        return;
    }

    Node* temp = head;
    while (temp->next && temp->next->data != key) temp = temp->next;

    if (!temp->next && before) {
        cout << "Key not found!\n";
        return;
    }

    Node* newNode = new Node{val, NULL};
    if (before) {
        newNode->next = temp->next;
        temp->next = newNode;
    } else {
        newNode->next = temp->next->next;
        temp->next->next = newNode;
    }
}

// Delete from beginning
void deleteFromBeginning() {
    if (!head) return;
    Node* temp = head;
    head = head->next;
    delete temp;
}

// Delete from end
void deleteFromEnd() {
    if (!head) return;
    if (!head->next) {
        delete head;
        head = NULL;
        return;
```

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

// Delete specific node
void deleteNode(int key) {
   if (!head) return;
   if (head->data == key) {
      deleteFromBeginning();
      return;
   }
   Node* temp = head;
   while (temp->next && temp->next->data != key) temp = temp->next;
   if (!temp->next) {
      cout << "Node not found!\n";
      return;
   }
   Node* del = temp->next;
   temp->next = del->next;
   delete del;
}

// Search node
void searchNode(int key) {
   Node* temp = head;
```

```
        int pos = 1;
        while (temp) {
            if (temp->data == key) {
                cout << "Node found at position: " << pos << endl;
                return;
            }
            temp = temp->next;
            pos++;
        }
        cout << "Node not found!\n";
    }

    // Display all
    void display() {
        Node* temp = head;
        cout << "Linked List: ";
        while (temp) {
            cout << temp->data << "->";
            temp = temp->next;
        }
        cout << "NULL\n";
    }
};

int main() {
    LinkedList list;
    int choice, val, key;
```

```cpp
    do {
        cout << "\n--- Singly Linked List Menu ---\n";
        cout << "1. Insert at beginning\n2. Insert at end\n3. Insert before a node\n4. Insert after a node\n";
        cout << "5. Delete from beginning\n6. Delete from end\n7. Delete a specific node\n";
        cout << "8. Search a node\n9. Display list\n10. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1: cout << "Enter value: "; cin >> val; list.insertAtBeginning(val); break;
            case 2: cout << "Enter value: "; cin >> val; list.insertAtEnd(val); break;
            case 3: cout << "Enter key & new value: "; cin >> key >> val; list.insertAtPosition(key, val, true); break;
            case 4: cout << "Enter key & new value: "; cin >> key >> val; list.insertAtPosition(key, val, false); break;
            case 5: list.deleteFromBeginning(); break;
            case 6: list.deleteFromEnd(); break;
            case 7: cout << "Enter value to delete: "; cin >> key; list.deleteNode(key); break;
            case 8: cout << "Enter value to search: "; cin >> key; list.searchNode(key); break;
            case 9: list.display(); break;
            case 10: cout << "Exiting...\n"; break;
            default: cout << "Invalid choice!\n";
        }
    } while (choice != 10);

    return 0;
}
```



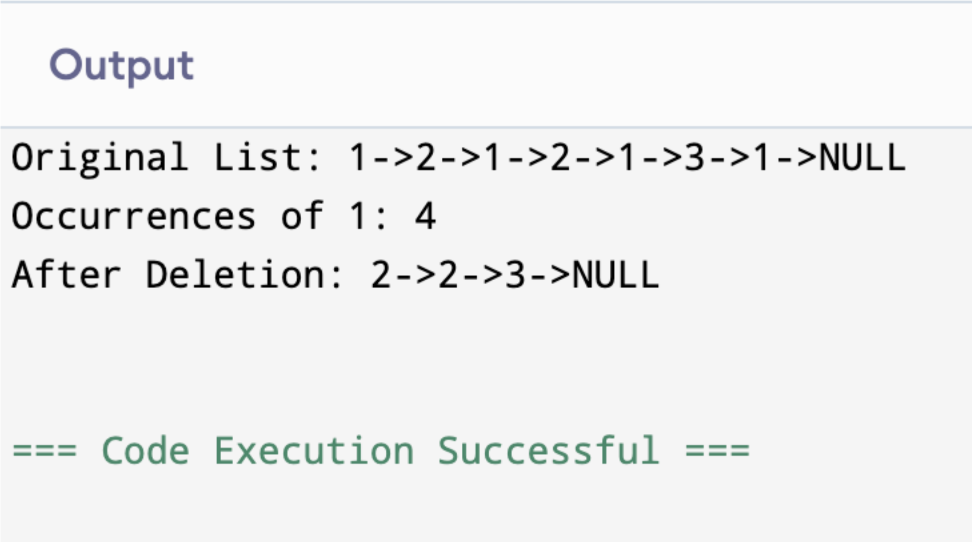
```
--- Singly Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert before a node
4. Insert after a node
5. Delete from beginning
6. Delete from end
7. Delete a specific node
8. Search a node
9. Display list
10. Exit
Enter choice: 9
Linked List: 10->20->NULL
```

2. Write a program to count the number of occurrences of a given key in a singly linked list and then delete all the occurrences. For example, if given linked list is 1->2->1- >2- >1->3->1 and given key is 1, then output should be 4. After deletion of all the occurrences of 1, the linked list is 2->2->3.

A2)

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

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

void deleteKeyOccurrences(Node*& head, int key) {
  int count = 0;
```

```
    // Delete occurrences at head
    while (head && head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
        count++;
    }

    Node* curr = head;
    while (curr && curr->next) {
        if (curr->next->data == key) {
            Node* temp = curr->next;
            curr->next = temp->next;
            delete temp;
            count++;
        } else {
            curr = curr->next;
        }
    }

    cout << "Occurrences of " << key << ": " << count << endl;
}

void display(Node* head) {
    while (head) {
        cout << head->data << "->";
        head = head->next;
    }
    cout << "NULL\n";
}

int main() {
    Node* head = new Node{1, new Node{2, new Node{1, new Node{2, new Node{1, new Node{3, new Node{1, NULL}}}}}}};
    cout << "Original List: ";
    display(head);

    int key = 1;
    deleteKeyOccurrences(head, key);

    cout << "After Deletion: ";
    display(head);
    return 0;
}
```



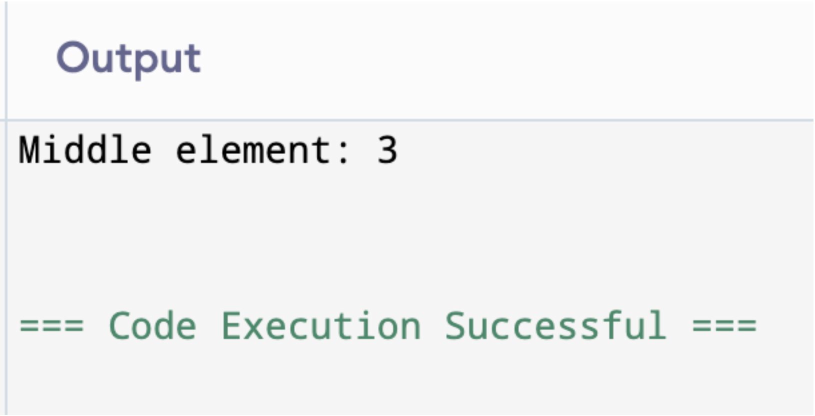
Output

```
Original List: 1->2->1->2->1->3->1->NULL
Occurrences of 1: 4
After Deletion: 2->2->3->NULL


=== Code Execution Successful ===
```

3. Write a program to find the middle of a linked list Input: 1->2->3->4->5 Output- 3

A3) #include <iostream>

```
using namespace std;

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

int findMiddle(Node* head) {
    Node* slow = head;
    Node* fast = head;

    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow->data;
}

int main() {
    Node* head = new Node{1, new Node{2, new Node{3, new Node{4, new Node{5, NULL}}}}};
    cout << "Middle element: " << findMiddle(head) << endl;
    return 0;
}
```



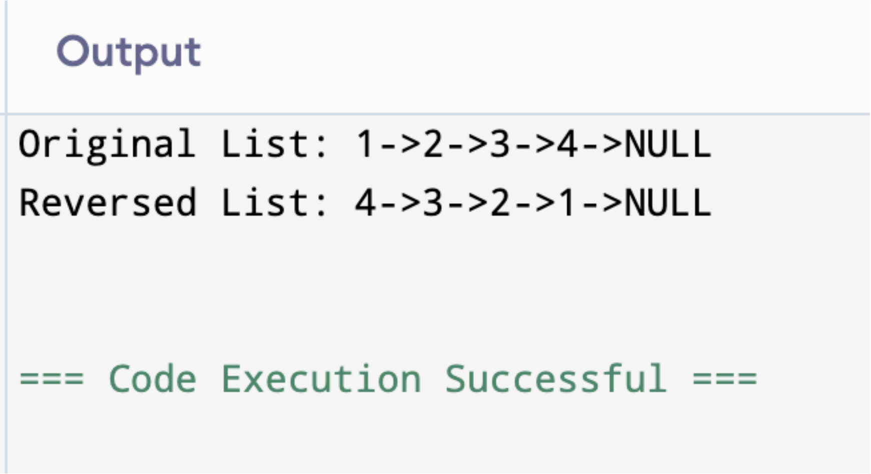
Output

```
Middle element: 3


=== Code Execution Successful ===
```

4. Write a program to reverse a linked list. Input: 1->2->3->4->NULL Output: 4->3->2->1->NULL

A4)

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

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

Node* reverseList(Node* head) {
  Node* prev = NULL;
  Node* curr = head;
  Node* next = NULL;

  while (curr) {
    next = curr->next;
    curr->next = prev;
    prev = curr;
    curr = next;
    }
    return prev;
}

void display(Node* head) {
    while (head) {
        cout << head->data << "->";
        head = head->next;
    }
    cout << "NULL\n";
}

int main() {
    Node* head = new Node{1, new Node{2, new Node{3, new Node{4, NULL}}}};
    cout << "Original List: ";
    display(head);

    head = reverseList(head);

    cout << "Reversed List: ";
    display(head);
    return 0;
}
```



```
Original List: 1->2->3->4->NULL
Reversed List: 4->3->2->1->NULL


=== Code Execution Successful ===
```