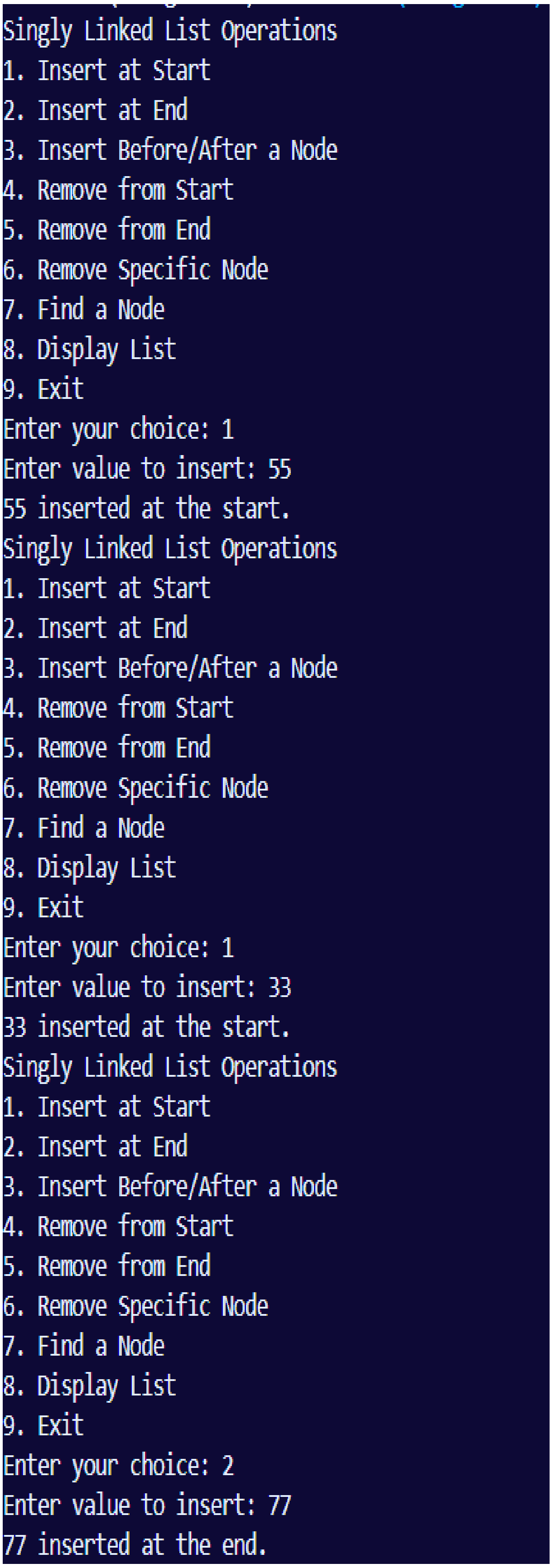
AQ1.cpp

```
#include <iostream>

using namespace std;


struct Node {

   int value;

   Node* nextNode;

};


Node* head = nullptr;


void insertAtStart(int num) {

   Node* newNode = new Node();

   newNode->value = num;

   newNode->nextNode = head;

   head = newNode;

   cout << num << " inserted at the start.\n";

}


void insertAtEnd(int num) {

   Node* newNode = new Node();

   newNode->value = num;

   newNode->nextNode = nullptr;


   if (head == nullptr) {

      head = newNode;

   } else {
        Node* currentNode = head;
        while (currentNode->nextNode != nullptr) {
            currentNode = currentNode->nextNode;
        }
        currentNode->nextNode = newNode;
    }
    cout << num << " inserted at the end.\n";
}

void insertBeforeOrAfter(int num, int target, bool insertAfter) {
    Node* newNode = new Node();
    newNode->value = num;

    if (head == nullptr) {
        cout << "List is empty. Cannot insert.\n";
        delete newNode;
        return;
    }

    Node* currentNode = head;
    Node* previousNode = nullptr;

    while (currentNode != nullptr && currentNode->value != target) {
        previousNode = currentNode;
        currentNode = currentNode->nextNode;
    }

    if (currentNode == nullptr) {
        cout << "Target node not found.\n";
        delete newNode;
        return;
    }

    if (insertAfter) {
        newNode->nextNode = currentNode->nextNode;
        currentNode->nextNode = newNode;
        cout << num << " inserted after " << target << ".\n";
    } else {
        if (previousNode == nullptr) {
            newNode->nextNode = head;
            head = newNode;
        } else {
            newNode->nextNode = currentNode;
            previousNode->nextNode = newNode;
        }
        cout << num << " inserted before " << target << ".\n";
    }
}

void removeFromStart() {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }
    Node* temp = head;
    head = head->nextNode;
    cout << temp->value << " removed from the start.\n";
    delete temp;
}

void removeFromEnd() {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }
    if (head->nextNode == nullptr) {
        cout << head->value << " removed from the end.\n";
        delete head;
        head = nullptr;
        return;
    }
    Node* currentNode = head;
    Node* previousNode = nullptr;
    while (currentNode->nextNode != nullptr) {
        previousNode = currentNode;
        currentNode = currentNode->nextNode;
    }
    previousNode->nextNode = nullptr;
    cout << currentNode->value << " removed from the end.\n";
    delete currentNode;
}

void removeSpecificNode(int num) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }
    Node* currentNode = head;
    Node* previousNode = nullptr;

    while (currentNode != nullptr && currentNode->value != num) {
        previousNode = currentNode;
        currentNode = currentNode->nextNode;
    }

    if (currentNode == nullptr) {
        cout << "Node " << num << " not found.\n";
        return;
    }

    if (previousNode == nullptr) {
        head = currentNode->nextNode;
    } else {
        previousNode->nextNode = currentNode->nextNode;
    }

    cout << "Node " << num << " removed.\n";
    delete currentNode;
}

void findNode(int num) {
    Node* currentNode = head;
    int position = 1;
    while (currentNode != nullptr) {
        if (currentNode->value == num) {
            cout << "Node " << num << " found at position " << position << ".\n";
            return;
        }
        currentNode = currentNode->nextNode;
        position++;
    }
    cout << "Node " << num << " not found.\n";
}

void showList() {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }
    Node* currentNode = head;
    cout << "Linked List: ";
    while (currentNode != nullptr) {
        cout << currentNode->value << " ";
        currentNode = currentNode->nextNode;
    }
    cout << endl;
}

int main() {
    int option, num, target;
```

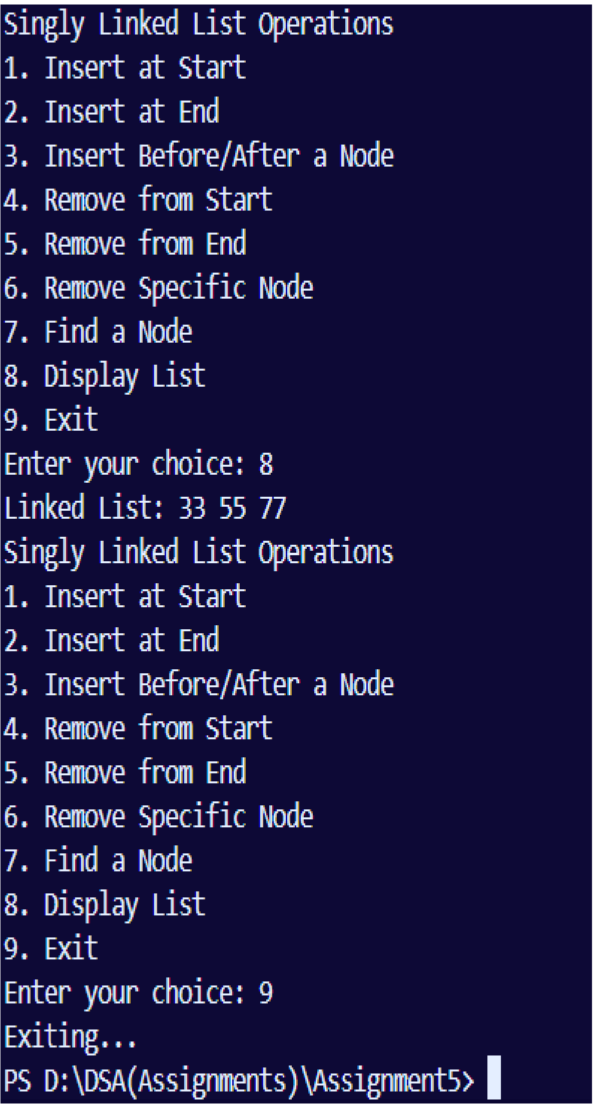
```
bool insertAfter;

do {
    cout << "Singly Linked List Operations\n";
    cout << "1. Insert at Start\n";
    cout << "2. Insert at End\n";
    cout << "3. Insert Before/After a Node\n";
    cout << "4. Remove from Start\n";
    cout << "5. Remove from End\n";
    cout << "6. Remove Specific Node\n";
    cout << "7. Find a Node\n";
    cout << "8. Display List\n";
    cout << "9. Exit\n";
    cout << "Enter your choice: ";
    cin >> option;

    switch (option) {
    case 1:
        cout << "Enter value to insert: ";
        cin >> num;
        insertAtStart(num);
        break;
    case 2:
        cout << "Enter value to insert: ";
        cin >> num;
        insertAtEnd(num);
        break;
    case 3:
```

```
    cout << "Enter value to insert: ";
    cin >> num;
    cout << "Enter target node value: ";
    cin >> target;
    cout << "Insert after target? (1 for yes, 0 for before): ";
    cin >> insertAfter;
    insertBeforeOrAfter(num, target, insertAfter);
    break;
case 4:
    removeFromStart();
    break;
case 5:
    removeFromEnd();
    break;
case 6:
    cout << "Enter value to remove: ";
    cin >> num;
    removeSpecificNode(num);
    break;
case 7:
    cout << "Enter value to find: ";
    cin >> num;
    findNode(num);
    break;
case 8:
    showList();
    break;
case 9:
```

```
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice. Please try again.\n";
        }
    } while (option != 9);

    return 0;
}
```



```
Singly Linked List Operations
1. Insert at Start
2. Insert at End
3. Insert Before/After a Node
4. Remove from Start
5. Remove from End
6. Remove Specific Node
7. Find a Node
8. Display List
9. Exit
Enter your choice: 1
Enter value to insert: 55
55 inserted at the start.
Singly Linked List Operations
1. Insert at Start
2. Insert at End
3. Insert Before/After a Node
4. Remove from Start
5. Remove from End
6. Remove Specific Node
7. Find a Node
8. Display List
9. Exit
Enter your choice: 1
Enter value to insert: 33
33 inserted at the start.
Singly Linked List Operations
1. Insert at Start
2. Insert at End
3. Insert Before/After a Node
4. Remove from Start
5. Remove from End
6. Remove Specific Node
7. Find a Node
8. Display List
9. Exit
Enter your choice: 2
Enter value to insert: 77
77 inserted at the end.
```



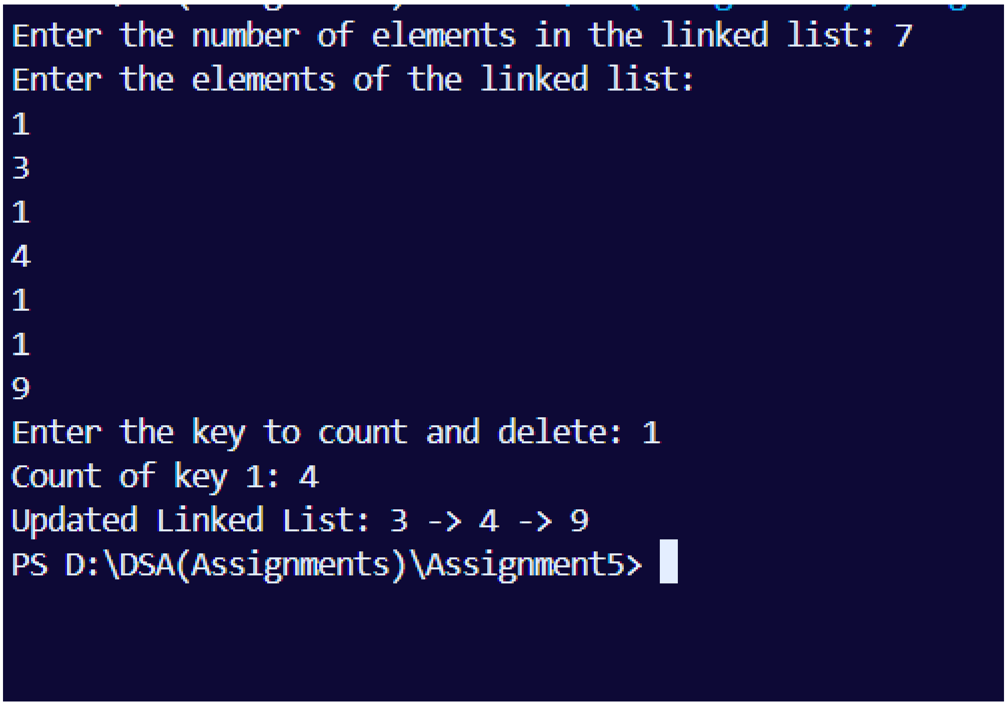
```
Singly Linked List Operations
1. Insert at Start
2. Insert at End
3. Insert Before/After a Node
4. Remove from Start
5. Remove from End
6. Remove Specific Node
7. Find a Node
8. Display List
9. Exit
Enter your choice: 8
Linked List: 33 55 77
Singly Linked List Operations
1. Insert at Start
2. Insert at End
3. Insert Before/After a Node
4. Remove from Start
5. Remove from End
6. Remove Specific Node
7. Find a Node
8. Display List
9. Exit
Enter your choice: 9
Exiting...
PS D:\DSA(Assignments)\Assignment5>
```

AQ2.cpp

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


struct Node {
    int value;
    Node* nextNode;
};


Node* head = nullptr;


void insertAtEnd(int num) {
    Node* newNode = new Node();
    newNode->value = num;
    newNode->nextNode = nullptr;


    if (head == nullptr) {
        head = newNode;
    } else {
        Node* currentNode = head;
        while (currentNode->nextNode != nullptr) {
            currentNode = currentNode->nextNode;
        }
        currentNode->nextNode = newNode;
    }
}
```

```
int countOccurrences(int key) {
    int count = 0;
    Node* currentNode = head;
    while (currentNode != nullptr) {
        if (currentNode->value == key) {
            count++;
        }
        currentNode = currentNode->nextNode;
    }
    return count;
}

void deleteAllOccurrences(int key) {
    while (head != nullptr && head->value == key) {
        Node* temp = head;
        head = head->nextNode;
        delete temp;
    }

    Node* currentNode = head;
    Node* previousNode = nullptr;

    while (currentNode != nullptr) {
        if (currentNode->value == key) {
            previousNode->nextNode = currentNode->nextNode;
            delete currentNode;
            currentNode = previousNode->nextNode;
        } else {
            previousNode = currentNode;
            currentNode = currentNode->nextNode;
        }
    }
}

void displayList() {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    Node* currentNode = head;
    while (currentNode != nullptr) {
        cout << currentNode->value;
        if (currentNode->nextNode != nullptr) {
            cout << " -> ";
        }
        currentNode = currentNode->nextNode;
    }
    cout << endl;
}

int main() {
    int n, val, key;
    cout << "Enter the number of elements in the linked list: ";
    cin >> n;

    cout << "Enter the elements of the linked list:\n";
```

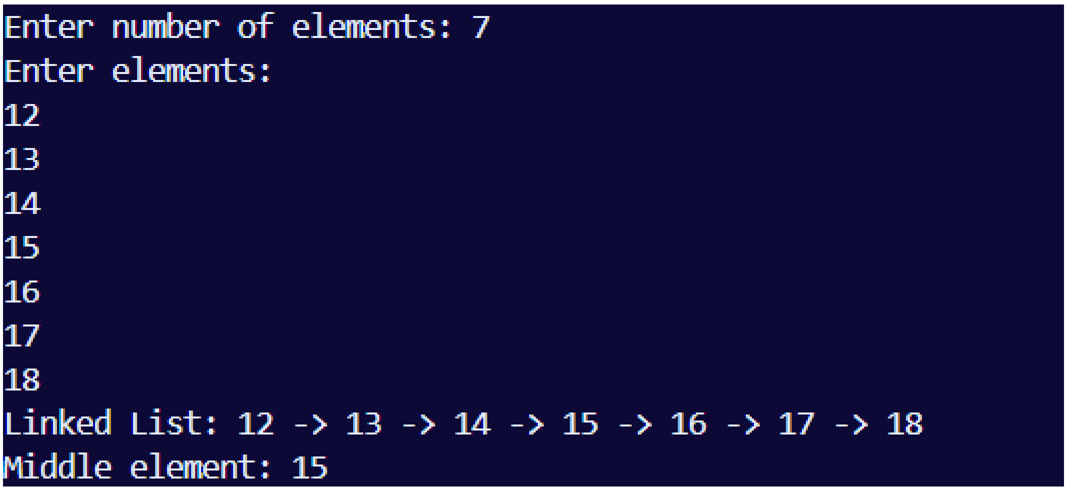
```
for (int i = 0; i < n; i++) {
    cin >> val;
    insertAtEnd(val);
}

cout << "Enter the key to count and delete: ";
cin >> key;

int count = countOccurrences(key);
cout << "Count of key " << key << ": " << count << endl;

deleteAllOccurrences(key);

cout << "Updated Linked List: ";
displayList();

return 0; }
```



```
Enter the number of elements in the linked list: 7
Enter the elements of the linked list:
1
3
1
4
1
1
9
Enter the key to count and delete: 1
Count of key 1: 4
Updated Linked List: 3 -> 4 -> 9
PS D:\DSA(Assignments)\Assignment5>
```

AQ3.cpp

```
#include <iostream>

using namespace std;


class Node {

public:

    int data;

    Node* next;


    Node(int x) {

        data = x;

        next = nullptr;

    }

};


int getLength(Node* head) {

    int length = 0;

    while (head) {

        length++;

        head = head->next;

    }

    return length;

}


int getMiddle(Node* head) {

    int length = getLength(head);

    int midIndex = length / 2;
    while (midIndex--) {
        head = head->next;
    }
    return head->data;
}

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

void displayList(Node* head) {
    while (head != nullptr) {
        cout << head->data;
        if (head->next != nullptr) cout << " -> ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    int n, val;

    cout << "Enter number of elements: ";
    cin >> n;
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++) {
        cin >> val;
        insertAtEnd(head, val);
    }

    cout << "Linked List: ";
    displayList(head);

    cout << "Middle element: " << getMiddle(head) << endl;

    return 0;
}
```

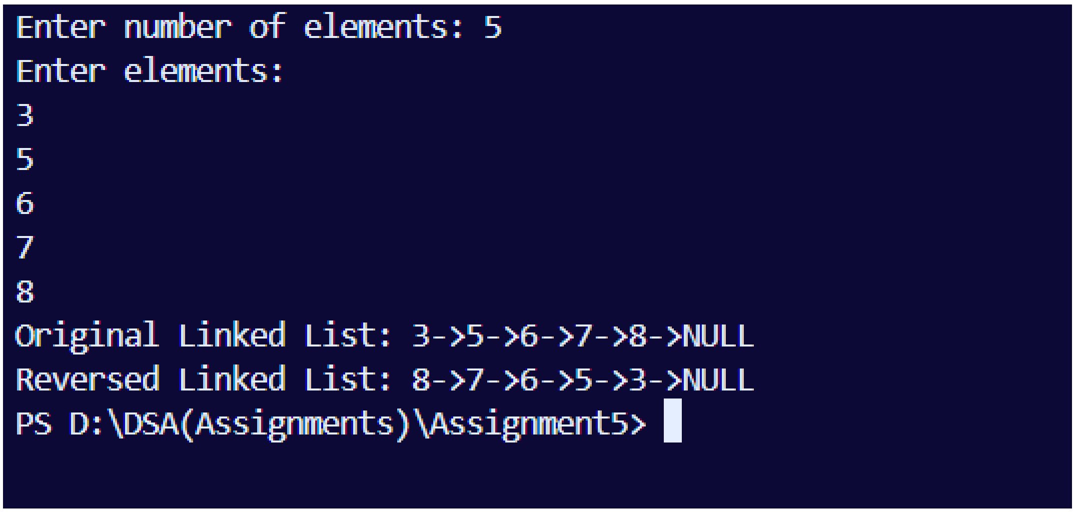


```
Enter number of elements: 7
Enter elements:
12
13
14
15
16
17
18
Linked List: 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18
Middle element: 15
```

AQ4.cpp

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

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

   Node(int x) {
       data = x;
       next = nullptr;
   }
};

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

```
void displayList(Node* head) {
    while (head != nullptr) {
        cout << head->data;
        if (head->next != nullptr)
            cout << "->";
        head = head->next;
    }
    cout << "->NULL" << endl;
}

void reverseList(Node*& head) {
    Node* prev = nullptr;
    Node* curr = head;
    Node* nextNode = nullptr;

    while (curr != nullptr) {
        nextNode = curr->next;  // Save the next node
        curr->next = prev;      // Reverse the current node's pointer
        prev = curr;            // Move prev and curr one step forward
        curr = nextNode;
    }
    head = prev;  // Make prev the new head of the reversed list
}

int main() {
    Node* head = nullptr;
    int n, val;
    cout << "Enter number of elements: ";
    cin >> n;
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++) {
        cin >> val;
        insertAtEnd(head, val);
    }

    cout << "Original Linked List: ";
    displayList(head);

    reverseList(head);

    cout << "Reversed Linked List: ";
    displayList(head);

    return 0;
}
```



```
Enter number of elements: 5
Enter elements:
3
5
6
7
8
Original Linked List: 3->5->6->7->8->NULL
Reversed Linked List: 8->7->6->5->3->NULL
PS D:\DSA(Assignments)\Assignment5>
```