Tabish Ahmad – 23103145

Assignment 5 – Linked List

Question 1)

```
#include<iostream>
using namespace std;
struct Node {
   int data;
   Node* next;};
class SinglyLinkedList {
public:
   Node* head;
   SinglyLinkedList() {
      head = nullptr;}
   void insertAtBeginning(int newData) {
      Node* newNode = new Node();
      newNode->data = newData;
      newNode->next = head;
      head = newNode;}     void insertAtEnd(int newData) {
      Node* newNode = new Node();
      newNode->data = newData;
      newNode->next = nullptr;
      if (head == nullptr) {
         head = newNode;
         return;}
      Node* temp = head;
      while (temp->next != nullptr) {
         temp = temp->next;}
      temp->next = newNode;}
   void insertAtPosition(int newData, int position) {
      Node* newNode = new Node();
      newNode->data = newData;
      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) {
         newNode->next = temp->next;
         temp->next = newNode;
      } else {
         cout << "Position out of range!" << endl;}}
   void deleteAtBeginning() {
      if (head == nullptr) {
         cout << "List is empty!" << endl;
         return;}
      Node* temp = head;
      head = head->next;
      delete temp;}
   void deleteAtEnd() {
      if (head == nullptr) {
         cout << "List is empty!" << endl;
         return;}
```

```
    if (head->next == nullptr) {
        delete head;
        head = nullptr;
        return;}
    Node* temp = head;
    while (temp->next->next != nullptr) {
        temp = temp->next;}
    delete temp->next;
    temp->next = nullptr;}
void deleteAtPosition(int position) {
    if (head == nullptr) {
        cout << "List is empty!" << endl;
        return;}
    if (position == 1) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;}
    Node* temp = head;
    for (int i = 1; i < position - 1 && temp != nullptr; i++) {
        temp = temp->next;}
    if (temp == nullptr || temp->next == nullptr) {
        cout << "Position out of range!" << endl;
        return;}
    Node* nodeToDelete = temp->next;
    temp->next = nodeToDelete->next;
    delete nodeToDelete;}
bool search(int value) {
    Node* temp = head;
    int position = 1;
    while (temp != nullptr) {
        if (temp->data == value) {
            cout << "Element " << value << " found at position " << position << endl;
            return true;}
        temp = temp->next;
        position++;}
    cout << "Element " << value << " not found in the list." << endl;
    return false;}
void display() {
    if (head == nullptr) {
        cout << "List is empty!" << endl;
        return;}
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " -> ";
        temp = temp->next;}
    cout << "NULL" << endl;}};
int main() {
    SinglyLinkedList list;
    list.insertAtEnd(1120);
    list.insertAtEnd(20);
    list.insertAtBeginning(5);
    list.insertAtPosition(15, 3);
    list.display();
    list.deleteAtBeginning();
    list.display();
    list.deleteAtEnd();
    list.display();
```

```
5 -> 1120 -> 15 -> 20 -> NULL
1120 -> 15 -> 20 -> NULL
1120 -> 15 -> NULL
Element 1120 found at position 1
1120 -> NULL


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

```
   list.search(1120);
   list.deleteAtPosition(2);
   list.display();
   return 0;
}
```

Question 2)

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;};
void insertAtEnd(Node*& head, int data) {
   Node* newNode = new Node;
   newNode->data = data;
   newNode->next = nullptr;
   if (!head) {
      head = newNode;
      return;}
   Node* temp = head;
   while (temp->next) {
      temp = temp->next;}
   temp->next = newNode;}
void reverseList(Node*& head) {
   Node* prev = nullptr;
   Node* curr = head;
   Node* next = nullptr;
   while (curr) {
      next = curr->next;
      curr->next = prev;
      prev = curr;
      curr = next;}
   head = prev;}
void printList(Node* head) {    while (head) {
      cout << head->data << " ";
      head = head->next;}
   cout << endl;}
int main() {
   Node* head = nullptr;
   insertAtEnd(head, 1);
   insertAtEnd(head, 2);
   insertAtEnd(head, 3);
   insertAtEnd(head, 4);
   insertAtEnd(head, 5);
   cout << "Original List: ";
   printList(head);
   reverseList(head);
   cout << "Reversed List: ";
   printList(head);
   return 0;}
```

```
Original List: 1 2 3 4 5
Reversed List: 5 4 3 2 1
```

Question 3)

```
#include <iostream>
using namespace std;
struct Node {
   int data;
```

```
    Node* next;};
void insertAtEnd(Node*& head, int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;
    if (!head) {
        head = newNode;
        return;}
    Node* temp = head;
    while (temp->next) {
        temp = temp->next;}
    temp->next = newNode;}
Node* mergeSortedLists(Node* head1, Node* head2) {
    if (!head1) return head2;
    if (!head2) return head1;
    Node* mergedHead = nullptr;
    if (head1->data <= head2->data) {
        mergedHead = head1;
        head1 = head1->next;
    } else {
        mergedHead = head2;
        head2 = head2->next;}
    Node* mergedTail = mergedHead;
    while (head1 && head2) {
        if (head1->data <= head2->data) {
            mergedTail->next = head1;
            head1 = head1->next;
        } else {
            mergedTail->next = head2;
            head2 = head2->next;}
        mergedTail = mergedTail->next;}
    if (head1) {
        mergedTail->next = head1;
    } else {
        mergedTail->next = head2;}
    return mergedHead;}
void printList(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;}
    cout << endl;}
int main() {
    Node* list1 = nullptr;
    Node* list2 = nullptr;
    insertAtEnd(list1, 1);
    insertAtEnd(list1, 3);
    insertAtEnd(list1, 5);
    insertAtEnd(list2, 2);
    insertAtEnd(list2, 4);
    insertAtEnd(list2, 6);
    cout << "List 1: ";
    printList(list1);
    cout << "List 2: ";
    printList(list2);
    Node* mergedList = mergeSortedLists(list1, list2);
    cout << "Merged List: ";
    printList(mergedList);
    return 0;}
```

```
List 1: 1 3 5
List 2: 2 4 6
Merged List: 1 2 3 4 5 6
```