Tabish Ahmad

23103145

Assignment 7 – Advanced Linked List

Question 1)

```
#include <iostream>

using namespace std;
class Node {
public:
   int data;
   Node* next;
   Node(int val) {
      data = val;
      next = nullptr;}};
class LinkedList {
public:
   Node* head;
   LinkedList() {
      head = nullptr;}
   void insertAtEnd(int val) {
      Node* newNode = new Node(val);
      if (head == nullptr) {
         head = newNode;
      } else {
         Node* temp = head;
         while (temp->next != nullptr) {
            temp = temp->next;}
         temp->next = newNode;}}
   void display() {
      Node* temp = head;
      while (temp != nullptr) {
         cout << temp->data << " -> ";
         temp = temp->next;}
      cout << "NULL" << endl;}};
void concatenate(LinkedList& list1, LinkedList& list2) {
   if (list1.head == nullptr) {
      list1.head = list2.head;
      return;}
   Node* temp = list1.head;
   while (temp->next != nullptr) {
      temp = temp->next;}
   temp->next = list2.head;}
int main() {
   LinkedList list1, list2;
   list1.insertAtEnd(1);
   list1.insertAtEnd(2);
   list1.insertAtEnd(3);
   list2.insertAtEnd(4);
   list2.insertAtEnd(5);
   list2.insertAtEnd(6);
   cout << "List 1: ";
   list1.display();
```

```
List 1: 1 -> 2 -> 3 -> NULL
List 2: 4 -> 5 -> 6 -> NULL
Concatenated List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
```

```
  cout << "List 2: ";
  list2.display();
  concatenate(list1, list2);
  cout << "Concatenated List: ";
  list1.display();
  return 0;}
```

Question 2)

```
#include <iostream>
using namespace std;
class Node {
public:
  int data;
  Node* next;
  Node(int val) {
    data = val;
    next = nullptr;}};
class LinkedList {
public:
  Node* head;
  LinkedList() {
    head = nullptr;}
  void insertAtEnd(int val) {
    Node* newNode = new Node(val);
    if (head == nullptr) {
      head = newNode;
    } else {
      Node* temp = head;
      while (temp->next != nullptr) {
        temp = temp->next;}
      temp->next = newNode;}}
  void display() {
    Node* temp = head;
    while (temp != nullptr) {
      cout << temp->data << " -> ";
      temp = temp->next;}
    cout << "NULL" << endl;}
  void split(LinkedList& list1, LinkedList& list2) {
    if (head == nullptr) {
      return;}
    Node* slow = head;
    Node* fast = head;
    while (fast->next != nullptr && fast->next->next != nullptr) {
      slow = slow->next;
      fast = fast->next->next;}
    list1.head = head;
    list2.head = slow->next;
    slow->next = nullptr;}};
int main() {
  LinkedList originalList;
  originalList.insertAtEnd(1);
  originalList.insertAtEnd(2);
  originalList.insertAtEnd(3);
  originalList.insertAtEnd(4);
  originalList.insertAtEnd(5);
  cout << "Original List: ";
  originalList.display();
```

```
Original List: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
First Half: 1 -> 2 -> 3 -> NULL
Second Half: 4 -> 5 -> NULL
```

```
  LinkedList list1, list2;
  originalList.split(list1, list2);
  cout << "First Half: ";
  list1.display();
  cout << "Second Half: ";
  list2.display();
  return 0;}
```

Question 3)

```
#include <iostream>
using namespace std;
class Node {
public:
  int data;
  Node* next;

  Node(int val) {
    data = val;
    next = nullptr;}};
class LinkedList {
public:
  Node* head;
  LinkedList() {
    head = nullptr;}
  void insertAtEnd(int val) {
    Node* newNode = new Node(val);
    if (head == nullptr) {
      head = newNode;
    } else {
      Node* temp = head;
      while (temp->next != nullptr) {
        temp = temp->next;}
      temp->next = newNode;}}
  void display() {
    Node* temp = head;
    while (temp != nullptr) {
      cout << temp->data << " -> ";
      temp = temp->next;}
    cout << "NULL" << endl;}
  Node* reverseKGroup(Node* head, int k) {
    if (head == nullptr || k <= 1) return head;
    Node* current = head;
    Node* prev = nullptr;
    Node* next = nullptr;
    int count = 0;
    Node* temp = head;
    while (temp != nullptr && count < k) {
      temp = temp->next;
      count++;}
    if (count < k) return head;
    current = head;
    count = 0;
    while (count < k && current != nullptr) {
      next = current->next;
      current->next = prev;
      prev = current;
      current = next;
      count++;}
```

```
    if (next != nullptr) {
        head->next = reverseKGroup(next, k);}
    return prev;}
  void reverseInGroups(int k) {
    head = reverseKGroup(head, k);}};
int main() {
  LinkedList list;
  list.insertAtEnd(1);
  list.insertAtEnd(2);
  list.insertAtEnd(3);
  list.insertAtEnd(4);
  list.insertAtEnd(5);
  list.insertAtEnd(6);
  list.insertAtEnd(7);
  list.insertAtEnd(8);
  cout << "Original List: ";
  list.display();
  int k = 3;
  list.reverseInGroups(k);
  cout << "Reversed List in Groups of " << k << ": ";
  list.display();
  return 0;}
```

```
Original List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> NULL
Reversed List in Groups of 3: 3 -> 2 -> 1 -> 6 -> 5 -> 4 -> 7 -> 8 ->
    NULL
```

Question 4)

```
#include <iostream>
using namespace std;
class Node {
public:
  int data;
  Node* next;
  Node(int val) {
    data = val;
    next = nullptr;}};
class LinkedList {
public:
  Node* head;
  LinkedList() {
    head = nullptr;}
  void insertAtEnd(int val) {
    Node* newNode = new Node(val);
    if (head == nullptr) {
      head = newNode;
    } else {
      Node* temp = head;
      while (temp->next != nullptr) {
        temp = temp->next;}
      temp->next = newNode;}}
  int countNodes() {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
      count++;
      temp = temp->next;
    }
    return count;}
  void display() {
    Node* temp = head;
    while (temp != nullptr) {
```

```
        cout << temp->data << " -> ";
        temp = temp->next;}
    cout << "NULL" << endl;}};
int main() {
  LinkedList list;
  list.insertAtEnd(10);
  list.insertAtEnd(20);
  list.insertAtEnd(30);
  list.insertAtEnd(40);
  list.insertAtEnd(50);
  cout << "Linked List: ";
  list.display();
  int nodeCount = list.countNodes();
  cout << "Number of nodes in the linked list: " << nodeCount << endl;
  return 0;}
```

```
Original List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> NULL
Reversed List in Groups of 3: 3 -> 2 -> 1 -> 6 -> 5 -> 4 -> 7 -> 8 ->
    NULL
```

Question 5)

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
class Node {
public:
  int data;
  Node* next;
  Node(int val) {
    data = val;
    next = nullptr;}};
class LinkedList {
public:
  Node* head;
  LinkedList() {
    head = nullptr;}
  void insert(int val) {
    Node* newNode = new Node(val);
    if (head == nullptr) {
      head = newNode;
    } else {
      Node* temp = head;
      while (temp->next != nullptr) {
        temp = temp->next;}
      temp->next = newNode;}}
  void display() {
    Node* temp = head;
    while (temp != nullptr) {
      cout << temp->data << " -> ";
      temp = temp->next;}
    cout << "NULL" << endl;}};
Node* mergeKLists(vector<Node*>& lists) {
  auto compare = [](Node* a, Node* b) {
    return a->data > b->data;};
  priority_queue<Node*, vector<Node*>, decltype(compare)> minHeap(compare);
  for (Node* list : lists) {
    if (list != nullptr) {
      minHeap.push(list);}}
  Node* dummy = new Node(0);
  Node* tail = dummy;
```

```
    while (!minHeap.empty()) {
        Node* minNode = minHeap.top();
        minHeap.pop();
        tail->next = minNode;
        tail = tail->next;
        if (minNode->next != nullptr) {
            minHeap.push(minNode->next);}}
    return dummy->next;}
int main() {
    vector<Node*> lists;
    LinkedList list1;
    list1.insert(1);
    list1.insert(4);
    list1.insert(5);
    lists.push_back(list1.head);
    LinkedList list2;
    list2.insert(1);
    list2.insert(3);
    list2.insert(4);
    lists.push_back(list2.head);
    LinkedList list3;
    list3.insert(2);
    list3.insert(6);
    lists.push_back(list3.head);
    Node* mergedHead = mergeKLists(lists);
    cout << "Merged Linked List: ";
    Node* temp = mergedHead;
    while (temp != nullptr) {
        cout << temp->data << " -> ";
        temp = temp->next;}
    cout << "NULL" << endl;
    return 0;}
```

```
Merged Linked List: 1 -> 1 -> 2 -> 3 -> 4 -> 4 -> 5 -> 6 -> NULL


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

```
Question 6)
#include <iostream>
#include <string>
using namespace std;
class Node {
public:
    string data;
    Node* next;
    Node(string val) {
        data = val;
        next = nullptr;}};
class SinglyLinkedList {
public:
    Node* head;
    SinglyLinkedList() {
        head = nullptr;}
    void addString(string val) {
        Node* newNode = new Node(val);
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;}
            temp->next = newNode;}}
```

```
Merged Linked List: 1 -> 1 -> 2 -> 3 -> 4 -> 4 -> 5 -> 6 -> NULL


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

```
  bool searchString(string val) {
    Node* temp = head;
    while (temp != nullptr) {
      if (temp->data == val) {
        return true;}
      temp = temp->next;}
    return false;}
  void displayStrings() {
    Node* temp = head;
    while (temp != nullptr) {
      cout << temp->data << " -> ";
      temp = temp->next;}
    cout << "NULL" << endl;}};
int main() {
  SinglyLinkedList list;
  list.addString("Hello");
  list.addString("World");
  list.addString("Linked");
  list.addString("List");
  cout << "Strings in the list: ";
  list.displayStrings();
  string searchItem = "World";
  if (list.searchString(searchItem)) {
    cout << searchItem << " found in the list." << endl;
  } else {
    cout << searchItem << " not found in the list." << endl;}
  return 0;}
```

Question 7)

```
#include <iostream>
#include <string>
using namespace std;
class Node {
public:
  string data;
  Node* next;
  Node(string val) {
    data = val;
    next = nullptr;}};
class SinglyLinkedList {
public:
  Node* head;
  SinglyLinkedList() {
    head = nullptr;}
  void addString(string val) {
    Node* newNode = new Node(val);
    if (head == nullptr) {
      head = newNode;
    } else {
      Node* temp = head;
      while (temp->next != nullptr) {
        temp = temp->next;}
      temp->next = newNode;}}
  string findLongestString() {
    if (head == nullptr) {
```

```
        return "";  }
      Node* temp = head;
      string longest = temp->data;
      while (temp != nullptr) {
        if (temp->data.length() > longest.length()) {
          longest = temp->data;
        }
        temp = temp->next;}
      return longest;}};
int main() {
   SinglyLinkedList list;
   list.addString("apple");
   list.addString("banana");
   list.addString("strawberry");
   list.addString("grape");
   string longestString = list.findLongestString();
   if (!longestString.empty()) {
      cout << "The longest string in the list is: " << longestString << endl;
   } else {
      cout << "The list is empty." << endl;}
   return 0;}
```

```
The longest string in the list is: strawberry


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

```
Question 8)
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
class Node {
public:
   string data;
   Node* next;
   Node(string val) {
      data = val;
      next = nullptr;}};
class SinglyLinkedList {
public:
   Node* head;
   SinglyLinkedList() {
      head = nullptr;}
   void addString(string val) {
      Node* newNode = new Node(val);
      if (head == nullptr) {
         head = newNode;
      } else {
         Node* temp = head;
         while (temp->next != nullptr) {
            temp = temp->next;}
         temp->next = newNode;}}
   void reverseStringsInNodes() {
      Node* temp = head;
      while (temp != nullptr) {
         reverse(temp->data.begin(), temp->data.end());
         temp = temp->next;}}
   void displayList() {
      Node* temp = head;
      while (temp != nullptr) {
         cout << temp->data << " ";
```

```
        temp = temp->next;}
    cout << endl;}};
int main() {
  SinglyLinkedList list;
  list.addString("apple");
  list.addString("banana");
  list.addString("grape");
  cout << "Original list: ";
  list.displayList();
  list.reverseStringsInNodes();
  cout << "List after reversing each string: ";
  list.displayList();
  return 0;}
```

```
Original list: apple banana grape
List after reversing each string: elppa ananab eparg


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

Question 9)

```
#include <iostream>
#include <string>
using namespace std;
class Node {
public:
  string data;
  Node* next;

  Node(string val) {
    data = val;
    next = nullptr;}};
class SinglyLinkedList {
public:
  Node* head;
  SinglyLinkedList() {
    head = nullptr;}
  void addString(string val) {
    Node* newNode = new Node(val);
    if (head == nullptr) {
      head = newNode;
    } else {
      Node* temp = head;
      while (temp->next != nullptr) {
        temp = temp->next;}
      temp->next = newNode;}}
  bool isVowel(char ch) {
    ch = tolower(ch);
    return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u');}
  int countVowelStrings() {
    Node* temp = head;
    int count = 0;
    while (temp != nullptr) {
      string str = temp->data;
      if (!str.empty() && isVowel(str.front()) && isVowel(str.back())) {
        count++;}
      temp = temp->next;}
    return count;}};
int main() {
  SinglyLinkedList list;
  list.addString("apple");
  list.addString("banana");
  list.addString("orange");
```

```
Number of strings that start and end with a vowel: 3


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

```
  list.addString("umbrella");
  int result = list.countVowelStrings();
  cout << "Number of strings that start and end with a vowel: " << result << endl;
  return 0;}
```

Question 10)

```
#include <iostream>
#include <string>
using namespace std;
class Node {
public:
  string data;
  Node* next;
  Node(string val) {
    data = val;
    next = nullptr;}};
class SinglyLinkedList {
public:
  Node* head;
  SinglyLinkedList() {
    head = nullptr;}
  void addString(string val) {
    Node* newNode = new Node(val);
    if (head == nullptr) {
      head = newNode;
    } else {
      Node* temp = head;
      while (temp->next != nullptr) {
        temp = temp->next;}
      temp->next = newNode;}}
  string concatenateStrings() {
    Node* temp = head;
    string result = "";
    while (temp != nullptr) {
      result += temp->data;
      temp = temp->next;}
    return result;}};
int main() {
  SinglyLinkedList list;
  list.addString("Hello");
  list.addString(" ");
  list.addString("World");
  list.addString("!");
  string concatenated = list.concatenateStrings();
  cout << "Concatenated string: " << concatenated << endl;
  return 0;}
```

```
Concatenated string: Hello World!


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