Tabish Ahmad – 23103145

Assignment 6 – Linked List

Question 1)

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;};
void insert(Node** head, int newData) {
   Node* newNode = new Node();
   newNode->data = newData;
   newNode->next = (*head);
   (*head) = newNode;}
bool detectCycle(Node* head) {
   Node* slow = head;
   Node* fast = head;
   while (fast != nullptr && fast->next != nullptr) {
      slow = slow->next;
      fast = fast->next->next;
      if (slow == fast) {
         return true;}}
   return false;}
int main() {
   Node* head = nullptr;
   insert(&head, 10);
   insert(&head, 20);
   insert(&head, 30);
   insert(&head, 40);
   head->next->next->next->next = head->next;
   if (detectCycle(head)) {
      cout << "Cycle detected" << endl;
   } else {
      cout << "No cycle detected" << endl;}
   return 0;}
```

```
Cycle detected


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

Question 2)

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;};
void insert(Node** head, int newData) {
   Node* newNode = new Node();
   newNode->data = newData;
   newNode->next = (*head);
   (*head) = newNode;}
void segregateEvenOdd(Node** head) {
   Node* evenStart = nullptr;
   Node* evenEnd = nullptr;
   Node* oddStart = nullptr;
```

```
  Node* oddEnd = nullptr;
  Node* current = *head;
  while (current != nullptr) {
    int val = current->data;
    if (val % 2 == 0) {
      if (evenStart == nullptr) {
        evenStart = current;
        evenEnd = evenStart;
      } else {
        evenEnd->next = current;
        evenEnd = evenEnd->next;}
    } else {
      if (oddStart == nullptr) {
        oddStart = current;
        oddEnd = oddStart;
      } else {
        oddEnd->next = current;
        oddEnd = oddEnd->next;}}
    current = current->next;}
  if (evenStart == nullptr || oddStart == nullptr)
    return;
  evenEnd->next = oddStart;
  oddEnd->next = nullptr;
  *head = evenStart;}
void printList(Node* head) {
  while (head != nullptr) {
    cout << head->data << " ";
    head = head->next;}}
int main() {
  Node* head = nullptr;
  insert(&head, 11);
  insert(&head, 10);
  insert(&head, 9);
  insert(&head, 6);
  insert(&head, 3);
  insert(&head, 2);
  segregateEvenOdd(&head);
  printList(head);
  return 0;}
```

```
2 6 10 3 9 11

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

Question 3)

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  Node* next;};
int getLength(Node* head) {
  int len = 0;
  while (head != nullptr) {
    len++;
    head = head->next;}
  return len;}
Node* getIntersectionNode(Node* head1, Node* head2) {
  int len1 = getLength(head1);
  int len2 = getLength(head2);
  int diff = abs(len1 - len2);
  if (len1 > len2) {
```

```
    while (diff--) {
        head1 = head1->next;}
    } else {
        while (diff--) {
            head2 = head2->next;}}
    while (head1 != nullptr && head2 != nullptr) {
        if (head1 == head2) {
            return head1;}
        head1 = head1->next;
        head2 = head2->next;}
    return nullptr;}
int main() {
    Node* newNode;
    Node* head1 = new Node();
    head1->data = 10;
    Node* head2 = new Node();
    head2->data = 3;
    newNode = new Node();
    newNode->data = 6;
    head2->next = newNode;
    newNode = new Node();
    newNode->data = 9;
    head2->next->next = newNode;
    newNode = new Node();
    newNode->data = 15;
    head1->next = newNode;
    head2->next->next->next = newNode;
    newNode = new Node();
    newNode->data = 30;
    head1->next->next = newNode;
    head1->next->next->next = nullptr;
    Node* intersection = getIntersectionNode(head1, head2);
    if (intersection != nullptr)
        cout << "Intersection point is " << intersection->data;
    else
        cout << "No intersection point";
    return 0;}
```

```
Intersection point is 15

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

Question 4)

```
#include <iostream>
#include <unordered_set>
using namespace std;
struct Node {
    int data;
    Node* next;};
void removeDuplicates(Node* head) {
    unordered_set<int> seen;
    Node* current = head;
    Node* prev = nullptr;

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

```
    current = prev->next;}}
void insert(Node** head, int new_data) {
   Node* new_node = new Node();
   new_node->data = new_data;
   new_node->next = (*head);
   (*head) = new_node;}
void printList(Node* node) {
   while (node != nullptr) {
      cout << node->data << " ";
      node = node->next;}} int main() {
   Node* head = nullptr;
   insert(&head, 10);
   insert(&head, 12);
   insert(&head, 11);
   insert(&head, 11);
   insert(&head, 12);
   insert(&head, 11);
   insert(&head, 10);
   removeDuplicates(head);
   printList(head);
   return 0;}
```

```
10 11 12

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

Question 5)

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;};
void rotate(Node** head, int k) {
   if (!*head || k == 0) return;
   Node* current = *head;
   int count = 1;
   while (count < k && current) {
      current = current->next;
      count++;}
   if (!current) return;
   Node* kthNode = current;
   while (current->next) {
      current = current->next;}
   current->next = *head;
   *head = kthNode->next;
   kthNode->next = nullptr;}
void insert(Node** head, int new_data) {
   Node* new_node = new Node();
   new_node->data = new_data;
   new_node->next = (*head);
   (*head) = new_node;}
void printList(Node* node) {
   while (node != nullptr) {
      cout << node->data << " ";
      node = node->next;}}
int main() {
   Node* head = nullptr;
   insert(&head, 60);
   insert(&head, 50);
   insert(&head, 40);
```

```
   insert(&head, 30);
   insert(&head, 20);
   insert(&head, 10);
   int k = 2;
   rotate(&head, k);
   printList(head);
   return 0;}
```

```
30 40 50 60 10 20

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

Question 6)

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;};
void insert(Node** head, int new_data) {
   Node* new_node = new Node();
   new_node->data = new_data;
   new_node->next = (*head);
   (*head) = new_node;}
void sortList(Node** head) {
   if (!*head) return;
   Node* zeroD = new Node();
   Node* oneD = new Node();
   Node* twoD = new Node();
   Node* zero = zeroD;
   Node* one = oneD;
   Node* two = twoD;
   Node* curr = *head;
   while (curr) {
      if (curr->data == 0) {
         zero->next = curr;
         zero = zero->next;
      } else if (curr->data == 1) {
         one->next = curr;
         one = one->next;
      } else {
         two->next = curr;
         two = two->next;}
      curr = curr->next;}
   zero->next = (oneD->next) ? (oneD->next) : (twoD->next);
   one->next = twoD->next;
   two->next = nullptr;
   *head = zeroD->next;
   delete zeroD;
   delete oneD;
   delete twoD;}
void printList(Node* node) {
   while (node != nullptr) {
      cout << node->data << " ";
      node = node->next;}}
int main() {
   Node* head = nullptr;
   insert(&head, 2);
   insert(&head, 1);
   insert(&head, 0);
   insert(&head, 1);
```

```
  insert(&head, 2);
  insert(&head, 0);
  sortList(&head);
  printList(head);
  return 0;}
```

```
0 0 1 1 2 2

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

Question 7)

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  Node* next;};
void insert(Node** head, int new_data) {
  Node* new_node = new Node();
  new_node->data = new_data;
  new_node->next = (*head);
  (*head) = new_node;}
Node* reverse(Node* head) {
  Node* prev = nullptr;
  Node* curr = head;
  Node* next = nullptr;
  while (curr != nullptr) {
    next = curr->next;
    curr->next = prev;
    prev = curr;
    curr = next;}
  return prev;}
bool isPalindrome(Node* head) {
  if (head == nullptr || head->next == nullptr)
    return true;
  Node* slow = head;
  Node* fast = head;
  while (fast != nullptr && fast->next != nullptr) {
    slow = slow->next;
    fast = fast->next->next;}
  slow = reverse(slow);
  Node* secondHalf = slow;
  Node* firstHalf = head;
  while (secondHalf != nullptr) {
    if (firstHalf->data != secondHalf->data)
      return false;
    firstHalf = firstHalf->next;
    secondHalf = secondHalf->next;}
  return true;}
int main() {
  Node* head = nullptr;
  insert(&head, 1);
  insert(&head, 2);
  insert(&head, 3);
  insert(&head, 2);
  insert(&head, 1);
  if (isPalindrome(head))
    cout << "The linked list is a palindrome.";
  else
    cout << "The linked list is not a palindrome.";
  return 0;}
```

```
The linked list is a palindrome.

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

Question 8)

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;};
void insert(Node** head, int new_data) {
   Node* new_node = new Node();
   new_node->data = new_data;
   new_node->next = (*head);
   (*head) = new_node;}
void findNthFromEnd(Node* head, int n) {
   Node* main_ptr = head;
   Node* ref_ptr = head;
   int count = 0;
   while (count < n) {
      if (ref_ptr == nullptr) {
         cout << "List has fewer than " << n << " nodes." << endl;
         return;}
      ref_ptr = ref_ptr->next;
      count++;}
   while (ref_ptr != nullptr) {
      main_ptr = main_ptr->next;
      ref_ptr = ref_ptr->next;}
   cout << "The " << n << "nd node from the end is: " << main_ptr->data << endl;}
int main() {
   Node* head = nullptr;
   insert(&head, 20);
   insert(&head, 4);
   insert(&head, 15);
   insert(&head, 35);
   int n = 2;
   findNthFromEnd(head, n);
   return 0;}
```

```
The 2nd node from the end is: 4


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

Question 9)

```
#include<iostream>
using namespace std;
class Student {
public:
   string name;
   int age;
   float score;
   Student* next;
   Student(string studentName, int studentAge, float studentScore) {
      name = studentName;
      age = studentAge;
      score = studentScore;
      next = nullptr;}};
class StudentList {
private:
   Student* head;
public:
   StudentList() {
      head = nullptr;}
   void addStudent(string name, int age, float score) {
      Student* newStudent = new Student(name, age, score);
      if (head == nullptr) {
```

```
        head = newStudent;
    } else {
        Student* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;}
        temp->next = newStudent;}}
  void displayStudents() {
    if (head == nullptr) {
        cout << "No students in the list." << endl;
        return;}
    Student* temp = head;
    while (temp != nullptr) {
        cout << "Name: " << temp->name << ", Age: " << temp->age << ", Score: " << temp->score << endl;
        temp = temp->next;}}
  void deleteStudent(string name) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;}
    if (head->name == name) {
        Student* toDelete = head;
        head = head->next;
        delete toDelete;
        cout << "Student " << name << " deleted." << endl;
        return;}
    Student* temp = head;
    while (temp->next != nullptr && temp->next->name != name) {
        temp = temp->next;
    }
    if (temp->next == nullptr) {
        cout << "Student not found." << endl;
    } else {
        Student* toDelete = temp->next;
        temp->next = temp->next->next;
        delete toDelete;
        cout << "Student " << name << " deleted." << endl;}}};

int main() {
  StudentList list;
  list.addStudent("Tabish", 21, 85.5);
  list.addStudent("Ahmad", 20, 90.2);
  list.addStudent("Sameer", 22, 76.8);
  cout << "Student list:" << endl;
  list.displayStudents();
  list.deleteStudent("Sara");
  cout << "\nUpdated student list after deletion:" << endl;
  list.displayStudents();
  return 0;}
```

```
Student list:
Name: Tabish, Age: 21, Score: 85.5
Name: Ahmad, Age: 20, Score: 90.2
Name: Sameer, Age: 22, Score: 76.8
Student not found.

Updated student list after deletion:
Name: Tabish, Age: 21, Score: 85.5
Name: Ahmad, Age: 20, Score: 90.2
Name: Sameer, Age: 22, Score: 76.8
```

Question 10)

```
#include <iostream>
using namespace std;
class Student {
public:
  string name;
  int age;
  float score;
  Student* next;
```

```
    Student(string studentName, int studentAge, float studentScore) {
        name = studentName;
        age = studentAge;
        score = studentScore;
        next = nullptr;}};
class StudentList {
private:
    Student* head;
public:
    StudentList() {
        head = nullptr;}
    void addStudent(string name, int age, float score) {
        Student* newStudent = new Student(name, age, score);
        if (head == nullptr) {
            head = newStudent;
        } else {
            Student* temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;}
            temp->next = newStudent;}}
    void displayStudents() {
        if (head == nullptr) {
            cout << "No students in the list." << endl;
            return;}
        Student* temp = head;
        while (temp != nullptr) {
            cout << "Name: " << temp->name << ", Age: " << temp->age << ", Score: " << temp->score << endl;
            temp = temp->next;}}
    void deleteStudent(string name) {
        if (head == nullptr) {
            cout << "List is empty." << endl;
            return;}
        if (head->name == name) {
            Student* toDelete = head;
            head = head->next;
            delete toDelete;
            cout << "Student " << name << " deleted." << endl;
            return;}
        Student* temp = head;
        while (temp->next != nullptr && temp->next->name != name) {
            temp = temp->next;}
        if (temp->next == nullptr) {
            cout << "Student not found." << endl;
        } else {
            Student* toDelete = temp->next;
            temp->next = temp->next->next;
            delete toDelete;
            cout << "Student " << name << " deleted." << endl;}}};
int main() {
    StudentList list;
    list.addStudent("Amit", 21, 85.5);
    list.addStudent("Sara", 20, 90.2);
    list.addStudent("Ravi", 22, 76.8);
    cout << "Student list:" << endl;
    list.displayStudents();
    list.deleteStudent("Sara");
    cout << "\nUpdated student list after deletion:" << endl;
    list.displayStudents();
```

```
Student list:
Name: Amit, Age: 21, Score: 85.5
Name: Sara, Age: 20, Score: 90.2
Name: Ravi, Age: 22, Score: 76.8
Student Sara deleted.

Updated student list after deletion:
Name: Amit, Age: 21, Score: 85.5
Name: Ravi, Age: 22, Score: 76.8
```

```
  return 0;}
```

Question 11)

```
#include <iostream>
using namespace std;
class Product {
public:
  int productID;
  string productName;
  int quantity;
  float price;
  Product* next;
  Product(int id, string name, int qty, float prc) {
    productID = id;
    productName = name;
    quantity = qty;
    price = prc;
    next = nullptr;
  }};
class Inventory {
private:
  Product* head;
public:
  Inventory() {
    head = nullptr;}
  void addProduct(int id, string name, int qty, float price) {
    Product* newProduct = new Product(id, name, qty, price);
    if (head == nullptr) {
      head = newProduct;
    } else {
      Product* temp = head;
      while (temp->next != nullptr) {
        temp = temp->next;}
      temp->next = newProduct;}}
  void displayInventory() {
    if (head == nullptr) {
      cout << "No products in the inventory." << endl;
      return;}
    Product* temp = head;
    while (temp != nullptr) {
      cout << "Product ID: " << temp->productID << ", Name: " << temp->productName
        << ", Quantity: " << temp->quantity << ", Price: $" << temp->price << endl;
      temp = temp->next;}}
  void deleteProduct(int id) {
    if (head == nullptr) {
      cout << "Inventory is empty." << endl;
      return;}
    if (head->productID == id) {
      Product* toDelete = head;
      head = head->next;
      delete toDelete;
      cout << "Product with ID " << id << " deleted." << endl;
      return;}
    Product* temp = head;
    while (temp->next != nullptr && temp->next->productID != id) {
      temp = temp->next;}
    if (temp->next == nullptr) {
      cout << "Product not found." << endl;
```

```
Inventory list:
Product ID: 101, Name: Laptop, Quantity: 5, Price: $750.99
Product ID: 102, Name: Smartphone, Quantity: 10, Price: $499.5
Product ID: 103, Name: Headphones, Quantity: 20, Price: $25.75
Product with ID 102 restocked. New quantity: 15
Product with ID 103 deleted.

Updated inventory list:
Product ID: 101, Name: Laptop, Quantity: 5, Price: $750.99
Product ID: 102, Name: Smartphone, Quantity: 15, Price: $499.5
```

```
    } else {
        Product* toDelete = temp->next;
        temp->next = temp->next->next;
        delete toDelete;
        cout << "Product with ID " << id << " deleted." << endl;}}
  void restockProduct(int id, int qty) {
    if (head == nullptr) {
        cout << "Inventory is empty." << endl;
        return;
    }
    Product* temp = head;
    while (temp != nullptr) {
        if (temp->productID == id) {
            temp->quantity += qty;
            cout << "Product with ID " << id << " restocked. New quantity: " << temp->quantity << endl;
            return;}
        temp = temp->next;}
    cout << "Product not found." << endl;}};
int main() {
  Inventory storeInventory;
  storeInventory.addProduct(101, "Laptop", 5, 750.99);
  storeInventory.addProduct(102, "Smartphone", 10, 499.50);
  storeInventory.addProduct(103, "Headphones", 20, 25.75);
  cout << "Inventory list:" << endl;
  storeInventory.displayInventory();
  storeInventory.restockProduct(102, 5);
  storeInventory.deleteProduct(103);
  cout << "\nUpdated inventory list:" << endl;
  storeInventory.displayInventory();
  return 0;}

Question 12)
#include <iostream>
using namespace std;
class Patient {
public:
  int patientID;
  string patientName;
  int emergencyLevel;
  Patient* next;
  Patient(int id, string name, int level) {
    patientID = id;
    patientName = name;
    emergencyLevel = level;
    next = nullptr;}};
class EmergencyRoom {
private:
  Patient* head;
public:
  EmergencyRoom() {
    head = nullptr;}
  void addPatient(int id, string name, int level) {
    Patient* newPatient = new Patient(id, name, level);
    if (head == nullptr || head->emergencyLevel < level) {
        newPatient->next = head;
        head = newPatient;
    } else {
        Patient* temp = head;
```

```
        while (temp->next != nullptr && temp->next->emergencyLevel >= level) {
            temp = temp->next;}
        newPatient->next = temp->next;
        temp->next = newPatient;}}
    void treatNextPatient() {
        if (head == nullptr) {
            cout << "No patients in the queue." << endl;
            return;}
        Patient* toTreat = head;
        cout << "Treating patient ID: " << toTreat->patientID << ", Name: " << toTreat->patientName << ", Emergency Level: " << toTreat-
>emergencyLevel << endl;
        head = head->next;
        delete toTreat;}
    void displayQueue() {
        if (head == nullptr) {
            cout << "No patients in the queue." << endl;
            return;}
        Patient* temp = head;
        while (temp != nullptr) {
            cout << "Patient ID: " << temp->patientID << ", Name: " << temp->patientName << ", Emergency Level: " << temp-
>emergencyLevel << endl;
            temp = temp->next;}}};
int main() {
    EmergencyRoom er;
    er.addPatient(101, "Alice", 2);
    er.addPatient(102, "Bob", 5);
    er.addPatient(103, "Charlie", 3);
    cout << "Current queue:" << endl;
    er.displayQueue();
    er.treatNextPatient();
    cout << "\nQueue after treating the next patient:" << endl;
    er.displayQueue();
    er.addPatient(104, "David", 4);
    cout << "\nQueue after adding a new patient:" << endl;
    er.displayQueue();
    return 0;}
```

```
Current queue:
Patient ID: 102, Name: Bob, Emergency Level: 5
Patient ID: 103, Name: Charlie, Emergency Level: 3
Patient ID: 101, Name: Alice, Emergency Level: 2
Treating patient ID: 102, Name: Bob, Emergency Level: 5

Queue after treating the next patient:
Patient ID: 103, Name: Charlie, Emergency Level: 3
Patient ID: 101, Name: Alice, Emergency Level: 2

Queue after adding a new patient:
Patient ID: 104, Name: David, Emergency Level: 4
Patient ID: 103, Name: Charlie, Emergency Level: 3
Patient ID: 101, Name: Alice, Emergency Level: 2
```