CPSC 256 – Midterm Test 4

The test consists of two questions. The first question is worth 15 points and the second is worth 25 points, for a total of 40 points for the entire test.

You should complete your solutions on the test paper itself. You are allowed pens, pencils, erasers, and blank scrap paper. No other aids are allowed.

Question 1: Inheritance and Polymorphism (15 points total)

You are building a program to manage different types of vehicles. The base class is Vehicle, and each derived class represents a specific vehicle (e.g., Car and Bike). The Vehicle class contains a virtual method honk() that is implemented differently in each derived class.

1. (4 points)

Write the declaration of the Vehicle base class, including:

• A protected member variable for the vehicle’s name.

• A public virtual method double honk() = 0; to make the class abstract.

• An initialization constructor which sets the internal name variable to the

provided vehicle name, and a default destructor.

1. (8 points)

Implement two derived classes:

• Car: It should override the honk() method to print the phrase “Beep Beep!”

• Bike: It should override the honk() method print the phrase “Ring Ring!”

1. Explain the role the “virtual” keyword in achieving polymorphism. What will happen if honk() is NOT declared virtual in the base class? Write your explanation in the space below:

int main() {

// Create an array of Vehicle pointers

Vehicle\* vehicles[2];

vehicles[0] = new Car(); // Car object

vehicles[1] = new Bike(); // Bike object

// Demonstrate polymorphism

for (int i = 0; i < 2; i++) {

vehicles[i]->honk(); // Calls the overridden honk() method

}

// Free dynamically allocated memory

for (int i = 0; i < 2; i++) {

delete vehicles[i];

}

return 0;

}

#include <iostream>

#include <string>

using namespace std;

// Abstract base class

class Vehicle {

protected:  
 string name; // Vehicle name  
  
public:  
 // Constructor  
 Vehicle(const string& vehicleName) : name(vehicleName) {}  
  
 // Pure virtual function  
 virtual void honk() const = 0;  
  
 // Virtual destructor  
 virtual ~Vehicle() {}

};

// Derived class: Car

class Car : public Vehicle{

public:  
 // Constructor  
 Car() : Vehicle("Car") {}  
  
 // Override the honk() method  
 void honk() const override {  
 cout << name << " honks: Beep Beep!" << endl;  
 }

};

// Derived class: Bike

class Bike : public Vehicle{

public:  
 // Constructor  
 Bike() : Vehicle("Bike") {}  
  
 // Override the honk() method  
 void honk() const override {  
 cout << name << " honks: Ring Ring!" << endl;  
 }

};

Question 2: Linked Lists (25 points total)

I asked ChatGPT to generate a question that would test your knowledge of linked lists in C++. Here is the question that it came up with , together with the solution that it generated:

You are tasked with implementing a singly linked list in C++. This linked list should support some basic operations and one advanced operation.

#include <iostream>  
using namespace std;  
  
struct Node {  
 int data; // TODO1: Data field  
 Node\* next; // TODO2: Pointer to the next node  
  
 // TODO3: Constructor for convenience  
 Node(int value) : data(value), next(nullptr) {}  
};  
  
class LinkedList {  
private:  
 Node\* head; // TODO4: Pointer to the first node in the list  
  
public:  
 // TODO5: Constructor  
 LinkedList() : head(nullptr) {}  
  
 // TODO6: Method to insert at the head of the list  
 void insertAtHead(int value) {  
 Node\* newNode = new Node(value); //TODO7: Create a new node  
 newNode->next = head; //TODO8: Point the new node to the current head  
 head = newNode; //TODO9: Update the head to the new node  
 }  
  
 // TODO10: Method to insert at a specific position  
 void insertAtPosition(int value, int position) {  
 Node\* newNode = new Node(value);  
  
 // TODO11: If inserting at the head  
 if (position == 0) {  
 newNode->next = head;  
 head = newNode;  
 return;  
 }  
  
 // TODO12: Traverse to the position  
 Node\* temp = head;  
 for (int i = 0; i < position - 1 && temp != nullptr; i++) {  
 temp = temp->next;  
 }  
  
 // TODO13: If position is invalid  
 if (temp == nullptr) {  
 cout << "Error: Position out of bounds!" << endl;  
 delete newNode;  
 return;  
 }  
  
 // TODO14: Insert the new node  
 newNode->next = temp->next;  
 temp->next = newNode;  
 }  
  
 // TODO15: Method to display the list  
 void display() {  
 Node\* temp = head;  
 while (temp != nullptr) {  
 cout << temp->data << " -> ";  
 temp = temp->next;  
 }  
 cout << "NULL" << endl;  
 }  
  
 // TODO16: Method to reverse the linked list  
 void reverse() {  
 Node\* prev = nullptr;  
 Node\* current = head;  
 Node\* next = nullptr;  
  
 while (current != nullptr) {  
 next = current->next; // TODO17: Save the next node  
 current->next = prev; // TODO18: Reverse the pointer  
 prev = current; // TODO19: Move prev one step forward  
 current = next; // TODO20: Move current one step forward  
 }  
  
 head = prev; // Update head to the new front  
 }  
  
 // TODO21: Destructor to clean up memory  
 ~LinkedList() {  
 Node\* temp;  
 while (head != nullptr) {  
 temp = head;  
 head = head->next;  
 delete temp;  
 }  
 }  
};

int main() {  
 LinkedList list;  
  
 // TODO22: Insert elements into the list  
 list.insertAtHead(10);  
 list.insertAtHead(20);  
 list.insertAtHead(30);  
  
 // TODO23: Display the list  
 cout << "Original list: ";  
 list.display();  
  
 // TODO24: Insert at specific position  
 list.insertAtPosition(25, 1);  
 cout << "After inserting 25 at position 1: ";  
 list.display();  
  
 // TODO25: Reverse the list  
 list.reverse();  
 cout << "Reversed list: ";  
 list.display();  
  
 return 0;  
}

Your task is to complete the comments in the above code (indicated in 25 places with TODO) so as to demonstrate that you understand how the code works.