National University of Computer and Emerging Sciences

Islamabad Campus

# Object Oriented Programming (CS1004)

**Course Instructor(s):**
Ms. Marium Hida
**Section(s): CS-A&B**

# Sessional-II Exam

**Total Time (Hrs):** 1
**Total Marks:** 60
**Total Questions:** 4

**Date:** Nov 4, 2024

Model Solution Sessional-II F-24

**Roll No** **Course Section** **Student Signature**

**Do not write below this line.**

**Attempt all the questions.**

**Instructions:**

1. Any answer provided on question paper will not be considered for marking. Therefore, avoid writing anything on question paper as final answer
2. Use permanent ink pens only. Any part done using soft pencil will not be marked and cannot be claimed for rechecking.
3. Ensure that you do not have any electronic gadget (like mobile phone, smart watch, etc.) with you.
4. We're not looking for flawless handwriting, but if your responses are challenging to read, a brief transcription would be helpful.
5. For additional 5 marks, please address all parts of a question in a single response. And mention the question and part no for each solution clearly on top of each answer. Write your roll number clearly on both the answer sheet and question paper.
6. Use a permanent pen to clearly cross out any rough work.

**[CLO 1: Model an algorithmic solution for a given problem using OOP]**

**Q1:** You are tasked with building a **3D graphics engine** that handles various geometrical shapes and transformations. One of the key elements in this engine is a class that represents a **3D Vector** in space, which has three coordinates: x, y, and z. This Vector3D class needs to support various operations commonly used in computer graphics. Write complete classes including all necessary functions for successful execution of program. Given the class

```
Class Vector
{
        private:
                int * dim;
        public:
                //add your functions here
};
```

**Your task is to interpret the given main() and implement the Vector3D class and overload the operators.** **[30 Marks]**

```
int main()
{
  Vector V1,V2(3,4,5); // initialize V1 with value=1 for each dimension and V2 with given set of values
  Vector V3(V2); //initialize V3 using V2
  Vector V4=V2+V3; //Add two vectors
  cin>>V1; //update magnitude of V1
  Vector V5;
  V5=V3++;
  V5=V4-V3; //Difference of 2 vectors
  cout<<V3^V2; //angle between 2 vectors. Assume math.h is included for this function only to call
               //pow() and cos-1 [ term acos() is used ] see help for this function below
  Vector V6=5*V5; //scaler product of vector
  Vector V7=V3*V4; // dot product of vectors
  Vector V8=V3%V4; //Cross product of vectors
  if(V4==V7)
  {
    cout<<V4;
  }
  else
    cout<<V4<<V7;
}
```

**Additional Constraints:**

- **Angle Calculation**: Implement the custom ^ operator to calculate the angle between two vectors using the dot product formula:

$$\Theta = \cos^{-1} \frac{(V2 * V3)}{V2().V3()}$$

Where: V2() calculates magnitude of said vector. Magnitude is calculated as: $\sqrt{(x^2+y^2+z^2)}$
Sqrt function use is not allowed. You may use pow()

**Calculating the Cross Product of 2 Vectors Ex.**

$\vec{A} = 3\hat{i} - 2\hat{j} - 5\hat{k} \quad \vec{B} = \hat{i} + 4\hat{j} - 4\hat{k}$

$$\vec{A} \times \vec{B} = \vec{C} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ 3 & -2 & -5 \\ 1 & 4 & -4 \end{vmatrix}$$

$$= \hat{i}(a_2b_3 - a_3b_2) - \hat{j}(a_1b_3 - a_3b_1) + \hat{k}(a_1b_2 - a_2b_1)$$

$$\boxed{\vec{A} \times \vec{B} = 28\hat{i} + 7\hat{j} + 14\hat{k}}$$

$$\begin{bmatrix} 2 \\ 7 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 8 \\ 2 \\ 8 \end{bmatrix} \quad \begin{matrix} 2 \cdot 8 \\ 7 \cdot 2 \\ 1 \cdot 8 \end{matrix}$$

**Dot product**

**Solution:**

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

class vector {
        int* dim;

public:
        vector()
        {
                dim = new int[3];
                for (int i = 0; i < 3; i++)
                        dim[i] = 1;
        }

        vector(int x, int y, int z)
        {
                dim = new int[3];
                dim[0] = x;
                dim[1] = y;
                dim[2] = z;
        }

        vector(const vector& obj)
        {
                dim = new int[3];
                for (int i = 0; i < 3; i++)
                        dim[i] = obj.dim[i];
        }

        vector operator=(const vector& obj)
        {
                for (int i = 0; i < 3; i++)
                        dim[i] = obj.dim[i];
                return *this;
        }

        ~vector()
        {
                delete[]dim;
```

1 1 2 2 2

```
        dim = NULL;
}

vector operator+(vector& o)
{
        vector res;
        for (int i = 0; i < 3; i++)
                res.dim[i] = dim[i] + o.dim[i];
        return res;
}

vector operator*(vector& o)
{
        vector res;
        for (int i = 0; i < 3; i++)
                res.dim[i] = dim[i] * o.dim[i];
        return res;
}

vector operator-(vector& o)
{
        vector res;
        for (int i = 0; i < 3; i++)
                res.dim[i] = dim[i] - o.dim[i];
        return res;
}

vector operator++(int)
{
        vector res(*this);
        for (int i = 0; i < 3; i++)
                dim[i]++;
        return res;
}

bool operator==(vector& o)
{
        for (int i = 0; i < 3; i++)
                if (dim[i] != o.dim[i])
                        return false;
```

```
        return true;
    }

    double operator()()
    {
        double x = pow(dim[0], 2);
        double y = pow(dim[1], 2);
        double z = pow(dim[2], 2);
        return pow(x + y + z, 0.5);
    }

    double operator^(vector &o)
    {
        double mag1, mag2;
        mag1 = (*this)();
        mag2 = o();
        vector r=((*this) * o);
        double dot = r.dim[0] + r.dim[1] + r.dim[2];
        double angle = acos(dot/(mag1 * mag2));
        return angle;
    }

    vector operator%(vector &o)
    {
        vector res;
        res.dim[0] = (dim[1] * o.dim[2]) - (dim[2] * o.dim[1]);
        res.dim[1] = (dim[0] * o.dim[2]) - (dim[2] * o.dim[0]);
        res.dim[2] = (dim[0] * o.dim[1]) - (dim[1] * o.dim[0]);
        return res;
    }

    friend vector operator*(int x, vector& o);
    friend ostream& operator<<(ostream &oo, vector& o);
    friend istream& operator>>(istream &ii, vector& o);
};

vector operator*(int x,vector &o)
{
    vector res;
    for (int i = 0; i < 3; i++)
```

2 2 2 2

```
            res.dim[i] = x * o.dim[i];
        return res;
}

ostream& operator<<(ostream& out, vector& o)
{
        out << o.dim[0] << "x+ " << o.dim[1] << "y+ " << o.dim[2] << "z\n";
        return out;
}

istream& operator>>(istream& in, vector& o)
{
        in >> o.dim[0] >>o.dim[1] >> o.dim[2];
        return in;
}

int main()
{
        vector V1, V2(3, 4, 5); // initialize V1 with value=1 for each dimension and V2 with
given set of values
        vector V3(V2); //initialize V3 using V2
        vector V4 = V2 + V3; //Add two vectors
        cin >> V1; //update magnitude of V1
        vector V5;
        V5 = V3++;
        V5 = V4 - V3;//Difference of 2 vectors
        cout << (V3 ^ V2); //angle between 2 vectors. Assume math.h is included for this
function only to call
        //pow() and cos-1 [ term acos() is used ] see help for this function below
        vector V6 = 5 * V5;//scaler product of vector
        vector V7 = V3 * V4; // dot product of vectors
        vector V8 = V3 % V4; //Cross product of vectors
        if (V4 == V7)
        {
                cout << V4;
        }
        else
                cout << V4 << V7;
}
```

**[CLO 2: Apply OOP concepts (Encapsulation, Inheritance, Polymorphism, Abstraction) to computing problems for the related program]**

**Q2:** Draw the UML Class Diagram (identify only classes and the relationship between them) of the following C++ code. **[10 Marks]**

```
#include <iostream>
#include <string>
#include <chrono>
#include <iomanip> // For std::put_time
using namespace std;
// Forward declarations
class Shipment;
class ShippingCompany;
// Constants for array sizes
const int max_inventory_size = 10; // Maximum number of CoffeePod items in inventory
const int max_shipments_size = 5;  // Maximum number of Shipments per Order
class Customer {
private:
    string name;
    string address;
    string creditCardInfo;
public:
    Customer(){}
    Customer(string n, string addr, string ccInfo)
        : name(n), address(addr), creditCardInfo(ccInfo) {}
    void placeOrder() {
        cout << "Order placed by " << name << endl;
    }
};
class CoffeePod {
private:
    string podType;
    float pricePerPack;
    int packsInStock;
public:
    CoffeePod(){}
    CoffeePod(string type, float price, int stock)
        : podType(type), pricePerPack(price), packsInStock(stock) {}

    void updateStock(int quantity) {
        packsInStock += quantity;
        cout << "Updated stock for " << podType << ": " << packsInStock << " packs" <<
endl;
    }
    float getPrice() const { return pricePerPack; }
    int getStock() const { return packsInStock; }
};
class Shipment {
private:
    int shipmentID;
    bool isFirstShipment;
    Customer customer;
    CoffeePod coffeePod;
    ShippingCompany* shippingCompany;

public:
    Shipment(){}
    Shipment(int id, bool first, Customer cust, CoffeePod pod)
        : shipmentID(id), isFirstShipment(first), customer(cust), coffeePod(pod) {}

    void setShippingCompany(ShippingCompany* company) {
        shippingCompany = company;
```

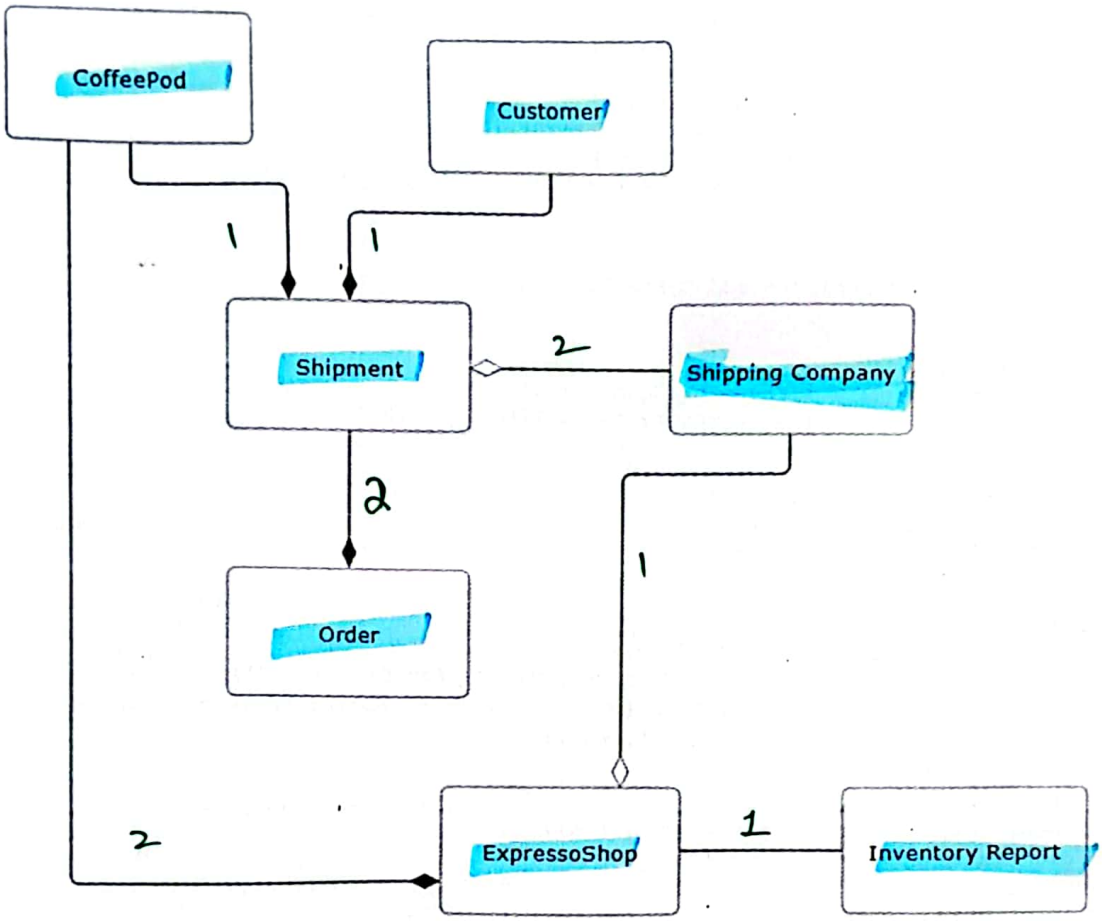
```
    }
    void calculateShippingCost() {
        if (isFirstShipment) {
            cout << "Shipping cost applied for first shipment of order." << endl;
        }
    }
    void chargeCustomer() {
        cout << "Customer charged for shipment ID: " << shipmentID << endl;
    }
};
class ShippingCompany {
private:
    string companyName;
    float costPerShipment;
public:
    ShippingCompany(string name, float cost) : companyName(name), costPerShipment(cost) {}
    void deliverShipment(Shipment& shipment) {
        cout << "Shipment delivered by " << companyName << endl;
    }
};
class Order {
private:
    int orderID;
    float totalPrice;
    float salesTax;
    float shippingFee;
    string status;
    Shipment shipments[max_shipments_size]; // Fixed-size array for Shipments
    int shipmentCount;
public:
    Order(int id) : orderID(id), totalPrice(0), salesTax(0), shippingFee(0),
status("Pending"), shipmentCount(0) {}

    void calculateTotal() {
        totalPrice = totalPrice + salesTax + shippingFee;
        cout << "Total price calculated for Order ID: " << orderID << endl;
    }
    void addShipment(const Shipment& shipment) {
        if (shipmentCount < max_shipments_size) {
            shipments[shipmentCount++] = shipment;
            cout << "Shipment added to Order ID: " << orderID << endl;
        }
        else {
            cout << "Unable to add shipment. Maximum number of shipments reached." << endl;
        }
    }
};
class InventoryReport {
private:
    string reportDate;
public:
    InventoryReport() {
        auto now = chrono::system_clock::now();
        auto in_time_t = chrono::system_clock::to_time_t(now);
        reportDate = put_time(localtime(&in_time_t), "%Y-%m-%d %X");
    }

    void generateReport() {
        cout << "Inventory report generated on " << reportDate << endl;
    }
};
class ExpressoShop {
private:
    string shopName;
    CoffeePod inventory[max_inventory_size]; // Fixed-size array for CoffeePod inventory
    int inventoryCount;
    ShippingCompany* shippingCompany;
```

```
public:
    ExpressoShop(string name, ShippingCompany* company) : shopName(name),
inventoryCount(0), shippingCompany(company) {}

    void addCoffeePod(const CoffeePod& pod) {
        if (inventoryCount < max_inventory_size) {
            inventory[inventoryCount++] = pod;
            cout << "CoffeePod added to inventory: " << pod.getStock() << " packs
available." << endl;
        }
        else {
            cout << "Unable to add CoffeePod. Inventory is full." << endl;
        }
    }
    void manageInventory() {
        for (int i = 0; i < inventoryCount; ++i) {
            cout << "Managing inventory for coffee pod type: " << i + 1 << endl;
        }
    }
    void generateMonthlyReport() {
        InventoryReport report;
        report.generateReport();
    }
};
int main() {
    ShippingCompany sc("FastShip", 25.0);
    Customer customer("Alice", "123 Main St", "1234-5678-9101-1121");
    CoffeePod pod("Espresso", 12.99, 50);
    ExpressoShop shop("Expresso Heaven", &sc);
    shop.addCoffeePod(pod);
    Order order(101);
    Shipment shipment(1, true, customer, pod);
    shipment.setShippingCompany(&sc);
    customer.placeOrder();
    shipment.calculateShippingCost();
    shipment.chargeCustomer();
    sc.deliverShipment(shipment);
    order.addShipment(shipment);
    order.calculateTotal();
    shop.generateMonthlyReport();
    return 0;
}
```




**[CLO 2: Apply OOP concepts (Encapsulation, Inheritance, Polymorphism, Abstraction) to computing problems for the related program]**

**Q3:** Examine the given code and illustrate the resulting output **[10 marks]**

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

class Resource {
public:
    Resource() {
        cout << "Resource Constructor called." << endl;
    }
    ~Resource() {
        cout << "Resource Destructor called." << endl;
    }
    void useResource() const {
        cout << "Using shared resource." << endl;
    }
};
class Manager {
```

```
private:
    string name;
    Resource* sharedResource;
public:
    Manager(const string& n, Resource* res) : name(n), sharedResource(res) {
        cout << "Manager Constructor: " << name << endl;
    }
    ~Manager() {
        cout << "Manager Destructor: " << name << endl;
    }
    void manage()  {
        cout << "Manager " << name << " is managing using shared resource." << endl;
        if (sharedResource) {
            sharedResource->useResource();
        }
    }
};
class Department {
private:
    string deptName;
    Manager* deptManager;
public:
    Department(const string& name, Manager* manager)
        : deptName(name), deptManager(manager) {
        cout << "Department Constructor: " << deptName << endl;
    }
    ~Department() {
        cout << "Department Destructor: " << deptName << endl;
    }
    void operate()  {
        cout << "Department " << deptName << " is operating." << endl;
        if (deptManager) {
            deptManager->manage();
        }
    }
};
class Company  {
private:
    Department dept1;
    Department dept2;
public:
    Company(const std::string& name, Manager* mgr1, Manager* mgr2)
        :  dept1("HR", mgr1), dept2("IT", mgr2) {
        cout << "Company Constructor: " << endl;
    }
    ~Company() {
        cout << "Company Destructor: " << endl;
    }

    void runCompany()  {
        cout << "Running Company: " << endl;
        dept1.operate();
        dept2.operate();
    }
};
int main() {
    cout << "=== Program Start ===\n" << endl;
    Resource sharedResource;
    Manager manager1("Alice", &sharedResource);
```

```
    Manager manager2("Bob", &sharedResource);
    Company myCompany("TechCorp", &manager1, &manager2);
    cout << "\n--- Running the Company ---\n";
    myCompany.runCompany();
    cout << "\n--- Program End, Destructors Called ---\n" << endl;
    return 0;
}
```

**Output:**

```
=== Program Start ===
Resource Constructor called.
Manager Constructor: Alice
Manager Constructor: Bob
Department Constructor: HR
Department Constructor: IT
Company Constructor:
--- Running the Company ---
Running Company:
Department HR is operating.
Manager Alice is managing using shared resource.
Using shared resource.
Department IT is operating.
Manager Bob is managing using shared resource.
Using shared resource.
--- Program End, Destructors Called ---
Company Destructor:
Department Destructor: IT
Department Destructor: HR
Manager Destructor: Bob
Manager Destructor: Alice
Resource Destructor called.
```

3

4

3

**[CLO 4: Apply good programming practices]**

**Q4:** You are provided with the codes in C++, you are required to apply principles of OOP and identify the errors if any in the code. Explain the error. If there are multiple errors explain each with the line number. Write the output only if there is no error in the code [3+3+4=10 Marks]

| S. no. | Code |
|---|---|
| 1) | 1. #include <iostream><br>2. using namespace std;<br>3. class Rectangle {<br>4. private:<br>5. int length;<br>6. int width; |

```
7.  public:
8.      Rectangle(int length = 0, int width = 0) {
9.              this->length = length;
10.             this->width = width;
11.     }
12.     void setLength(int length) {
13.             this->length = length;
14.     }
15.     void setWidth(int width) {
16.             this->width = width;
17.     }
18.     double getLength() const {
19.             return this->length;
20.     }
21.     double getWidth() const {
22.             return this->width;
23.     }
24.     Rectangle operator+(const int& num)
25.     {
26.             return Rectangle(20, 30);
27.     }
28.     void display() {
29.             cout << "Length = " << this->length << endl;
30.             cout << "Width = " << this->width << endl;
31.     }
32. };
33. int main(){
34.     Rectangle r1(4, 5), r2;
35.     r2 = 5 + r1; // this calls a global function. Only available
                     is member operator+() function
36.     r1.display();
37.     r2.display();
38. }
```

3

**2)**

```
1.  #include <iostream>
2.  using namespace std;
3.  class Vehicle {
4.      int wheels;
5.      int speed;
6.      Vehicle(int w, int s) {  //Constructor cannot be private
7.          wheels = w;
8.          speed = s;
9.      }
10. public:
11.     void showInfo() {
12.         cout << "Wheels: " << wheels << ", Speed: " << speed <<
    endl;
13.     }
14.     void changeSpeed(int newSpeed) {
15.         speed = newSpeed;
16.     }
17. };
18. int main() {
19.     Vehicle car(4, 120);
20.     car.showInfo();
21.     car.changeSpeed(100);
22.     car.showInfo();
23.     return 0;
24. }
```

3

**3)**

```
1.  #include<iostream>
2.  using namespace std;
3.  class Sample {
```

```
4.  private:
5.      int data;
6.  public:
7.      Sample(int d) : data(d) {}
8.      void setData(int value) { data = value; }
9.      int getData() const { return data; }
10. };
11. int main() {
12.     const Sample obj(10);
13.     obj.setData(20); //Constant Object cannot make changes
                         after constructor
14.     cout << obj.getData();
15.     return 0;
16. }
```

4

**Bonus Question: Find the Missing Letter [5 Marks]**

| | | | |
|---|---|---|---|
| R | E | T | T |
| E | L | G | N |
| I | S | S | I |
| M | E | H | T |
| D | N | I | F |

**Letter: F** 5

**(any cutting/over writing=ZERO marks)**