CS1004 Object Oriented Programming

Wednesday, May 31, 2023

Course Instructor

Dr. Bilal Khan, Dr. Imran Babar, Dr. Khalid Hussain, Rizwan ul Haq, Usman Ghous, Saud Arshad

Serial No:

Final Exam-Subjective

Total Time: 2 hr 20 mins

Total Marks: 80 Marks

_____

Signature of Invigilator

Roll No	Section	Signature

DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.

Instructions:

1. Verify at the start of the exam that you have a total of five (5) questions printed on fifteen (15) pages including this title page.
2. Attempt all questions on the question-book and in the given order.
3. The exam is closed books, closed notes. Please see that the area in your threshold is free of any material classified as 'useful in the paper' or else there may a charge of cheating.
4. Read the questions carefully for clarity of context and understanding of meaning and make assumptions wherever required, for neither the invigilator will address your queries, nor the teacher/examiner will come to the examination hall for any assistance.
5. Fit in all your answers in the provided space. You may use extra space on the last page if required. If you do so, clearly mark question/part number on that page to avoid confusion.
6. Use only your own stationery and calculator. If you do not have your own calculator, use manual calculations.
7. Use only permanent ink-pens. Only the questions attempted with permanent ink-pens will be considered. Any part of paper done in lead pencil cannot be claimed for checking/rechecking.

	Q-1	Q-2	Q-3	Q-4	Q-5	Total
Total Marks	15	25	10	20	10	**80**
Marks Obtained						

Vetted By: _____ Vetter Signature: _____

University Answer Sheet Required: No ☐ Yes ☐

Question 1:	CLO 2	Marks: 5

a. Rectify the given code. Only correct the erroneous lines. (5 marks)

```
class abc{
    private:
        int a;
    public:
        int abc (int b) { //constructor
            a = b;
            cout<<a<<"\n";
        }
        int sum (int x, int y){
            int mysum=x+y;
            cout<<"Sum: "<<mysum;
        }};
int main() {
    abc s1(5), *s1;
    s1->abc(5);
    s1->sum(5,10);
    return 0;
}
```

abc (int b) { }	//Constructor should be without return type.	[Marks: 2]
abc s1(5), *s;	//s1 and *s1 will result in error due to naming issue.	[Marks: 1]
s->abc(5);	// Constructor call using pointer is error	[Marks: 1]
s->sum(5,10);	// Function call using pointer	[Marks: 1]

b. What is the output of the given code snippet? (2 Marks)

```
class Count {
private:
int value;
public:
Count() : value(7){}
void operator ++() { value-=1;}
void display(){ cout<< “Count: ”<<value<<endl;}};
int main(){
Count objcount;
++objcount;
```

```
objcount.display();
return 0;
}
```

Count: 6

c. What is the output of the given code snippet? (3 Marks)

```
class base{
public:
~base() {
cout << "destructing base\n";
} };
class derived : public base {
public:
~derived() {
cout << "destructing derived\n";
}};
int main()
{
base *p = new derived;
delete p;
return 0;
}
```

```
Destructing base
```

d. Write the main function for the given code. Access all three show() member functions with one class pointer variable. (5 Marks)

```
class A {
public:
void show()
{
cout << "Parent class A ";
}};
class B : public A{
public:
void show()
{
cout << "Parent class B ";
}};
class C : public A{
public:
void show()
```

```
{
cout << "Parent class C ";
}};
```

```
class A {
public:
virtual void show()                                    [Marks: 1]
{
cout << "Parent class A ";
}};

int main()
{
A obj1;
B obj2;
C obj3;

A *Ptr;                                                [Marks: 1]

Ptr = &obj1;                                           [Marks: 1]
Ptr->show();

Ptr = &obj2;                                           [Marks: 1]
Ptr->show();

Ptr = &obj3;                                           [Marks: 1]
Ptr->show();

return 0;
}
```

Question 2:	Marks: 25

Implement a C++ program using object-oriented principles to count the occurrences of a specific element in an array. Design a template function that works for arrays of int, float, double, and char types. Your program should include the following:

1. Define a template function **CountOccurrences** that takes an array and the size of the array as arguments, along with a target element to count**. [CLO 4]** (5 Marks)

```
template<typename T, size_t N>
```

2. Implement the **CountOccurrences** function using a loop to iterate through the array elements, count the occurrences of the target element and return the count. **[CLO 1]** (5 Marks)

```
size_t CountOccurrences( const T (&arr)[N], const T& target )
{
    size_t count = 0;
    for (size_t i = 0; i < N; ++i) {
        if (arr[i] == target) {
            count++;
        }
    }
    return count;
}
```

3. Write a **main** function that demonstrates the usage of the **CountOccurrences** function by creating arrays of different types (**int**, **float**, **double**, and **char**) with varying sizes. Specify a target element for each array and call the **CountOccurrences** function to count the occurrences. **[CLO 1]** (5 Marks)

```
int main() {
    int intArr[] = {1, 2, 3, 4, 5, 2};

    int intTarget = 2;

    size_t intCount = CountOccurrences(intArr, intTarget);


}
```

4. Display the number of occurrences of the target element for each array type. **[CLO 1]** (5 Marks)

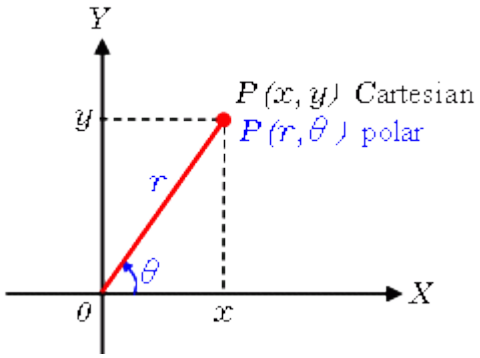
```
int main() {
    .

    .

    .

    size_t intCount = CountOccurrences(intArr, intTarget);

    cout << "Occurrences of " << intTarget << " in int array: " << intCount << endl;
     }
```

5. Test your program with different arrays and target elements to verify its correctness and robustness. **[CLO 1]** (5 Marks)

```
int main() {
    int intArr[] = {1, 2, 3, 4, 5, 2};
    float floatArr[] = {1.1f, 2.2f, 3.3f, 2.2f, 4.4f};
    double doubleArr[] = {1.1, 2.2, 3.3, 4.4, 5.5, 2.2};
    char charArr[] = "Hello, World!";

    int intTarget = 2;
    float floatTarget = 2.2f;
    double doubleTarget = 2.2;
    char charTarget = 'o';

    size_t intCount = CountOccurrences(intArr, intTarget);
    size_t floatCount = CountOccurrences(floatArr, floatTarget);
    size_t doubleCount = CountOccurrences(doubleArr, doubleTarget);
    size_t charCount = CountOccurrences(charArr, charTarget);

    cout << "Occurrences of " << intTarget << " in int array: " << intCount << endl;
    cout << "Occurrences of " << floatTarget << " in float array: " << floatCount <<
endl;
    cout << "Occurrences of " << doubleTarget << " in double array: " <<
doubleCount << endl;
    cout << "Occurrences of '" << charTarget << "' in char array: " << charCount
<< endl;

    return 0;
}
```

Question 3	CLO 3	Marks: 10

There are two ways to locate a point on the plane, Polar coordinates system and Cartesian coordinate system as shown in Figure below.



A point P in Polar system is defined as radius (r) and angle (theta), whereas the same point in Cartesian system can be represented by two numbers ***x*** and ***y***. The following formula can be used to convert a polar point P(radius, angle) to the Cartesian Coordinate C(x,y).
x = radius. cos(angle);
y = radius. sin(angle);

You are required to create two classes, Polar and Cartesian. Both of the classes have private data members according to the explanation given above and there ***Does Not*** exist getter functions in both classes.
[3 Points for the Cartesian Class + 7 Points for the Polar Class]
Furthermore, your program shall provide the necessary functionality to successfully execute the following instructions given in main() routine.

```
int main (){
Polar P(10.5,60)
Cartesian c = p;
}
```

There may exist two ways for conversion between the two coordinate systems, hence you are required to provide both solutions.

I asked for both ways to solve the conversion.
1. Conversion constructor to be implemented in Cartesian class
2. Using cast operator to be implemented in Polar class

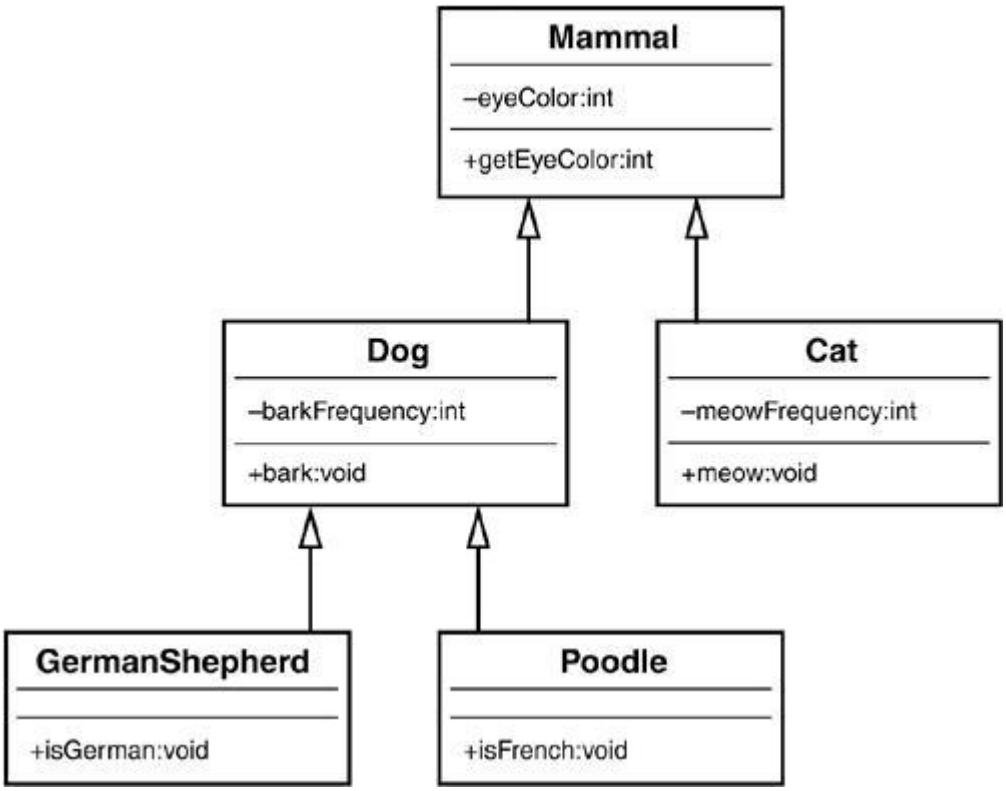
```
class Cartesian {
private:          //1 point
 double x;   double y;

public:

Cartesian(double _x=0, double _y=0) {  x = _x;   y = _y;  }

//conversion constructor

Cartesian(Polar p) {    //IMPORTANT: 2 Points. Conversion constructor shouldn't use any getter function for accessing polar's private members.

x = p.radius*cos(p.angle);      y = p.radius*sin(p.angle);      }
 };


class Polar {
private:          //1 Point
double radius;   double angle;

public:
Polar (double r=0, double a=0) {       //2 points
radius = r;  angle = a;
}

friend class Cartesian;          //2 points. Important for solution#1

 operator Cartesian() {          //2 Points, Solution using cast operator

 double x = radius*cos(angle);    double y = radius*sin(angle);
return Cartesian(x,y);
 }
};
int main() {
Polar P(13, 60);
Cartesian c = P;
}
```

Question 4	Marks: 20

Part A: CLO 2

Implement the below UML diagram using the concept of polymorphism. The Mammal class is an abstract class, and it has getEyeColor method and destructor, both are pure virtual. In all derived classes there is default argument constructor and destructor. Call the base class methods in child classes and do overriding where necessary. [10 Marks]

In driver program, create an array of pointers of mammal class pointing to objects of derived classes. Use loop to call methods of derived classes with base class pointer. [4 Marks]



Driver Function (4 marks)

```
#include<iostream>
using namespace std;
// *************** CODE ******************
// Step 1
class Mammal
{
int eyeColor;
public:
virtual int getEyeColor() = 0;
virtual ~Mammal() = 0;
// constructor
```

```
Mammal(){}
};

Mammal::~Mammal()
{
//cout << "Here is Pure Virtual Destructor of Mammal Class\n";
}

// Step 2
class Cat : public Mammal
{
int meowFrequency;
public:

// Default Argument Constructor
Cat(int meowFrequency = 6)
{
this->meowFrequency = meowFrequency;
}
void meow()
{
cout << "Bark Frequency : " << meowFrequency << '\n';
}

// function overriding
int getEyeColor() override
{
```

```
return 6;
}
//Destructor
~Cat(){}
};
// Step 3
class Dog : public Mammal
{
protected:
int barkFrequency;
public:
// Default Argument Constructor
Dog(int barkFrequency = 7)
{
this->barkFrequency = barkFrequency;
}
void Bark()
{
cout << "Bark Frequency : " << barkFrequency << '\n';
}
// Additional Function to Support Operator Overloading
int getBarkFrequency()
{
return barkFrequency;
}
//Destructor
~Dog() {}
};
// Step 4
class GermanShephard : public Dog
{
public:
void isGerman()
{
cout << "Yes! It's German\n";
}
// overriding
int getEyeColor() override
{
return 7;
}
//Destructor
~GermanShephard() {}
};
```

```
// Step 5
class Poodle : public Dog
{
public:
void isPoodle()
{
cout << "Yes! It's Poodle\n";
}
// overriding
int getEyeColor() override
{
return 8;
}
// Friend function to overload the stream insertion operator
friend ostream& operator<<(std::ostream& out, Poodle& p);
// Overloaded + operator as a member function
```

In driver program, create an array of pointers of mammal class pointing to objects of derived classes. Use loop to call methods of derived classes with base class pointer. [4 Marks]

```
int main()
{
Mammal* mammal[3];
mammal[0] = new Cat();
mammal[1] = new GermanShephard();
mammal[2] = new Poodle();
for (int i = 0; i < 3; i++)
{
// Base Class Methods
cout << mammal[i]->getEyeColor() << endl;
}
// Poodle p1,p2;
// cout << p1;
// cout << p1 + p2 << endl;
```

```
// Deallocate Memory
for (int i = 0; i < 3; i++)
{
delete mammal[i];
}
return NULL;
}
```

Part B: CLO 4 **6 Marks**

1. Overload the stream insertion operators as friend function for the Poodle class to output the object of poodle class. [3 Marks]

```
// Definition of the friend function
ostream& operator<<(ostream& out,Poodle& p) {
out << "EyeColor: " << p.getEyeColor() << ", BarkFreq: " <<
p.getBarkFrequency() << " ";
p.isPoodle();
out << endl;
return out;
}
```

2. Overload **+ operator** with return type integer and it should add barkFrequency of two Poodle objects. [3 Marks]

```
int operator+(const Poodle& other) const {
return barkFrequency + other.barkFrequency;
}
//Destructor
~Poodle() {}
};
```

Question 5:	CLO 2	Marks: 10

We want to store the information of different vehicles.
Create a class named Vehicle with two data member named mileage and price.

Create its two subclasses *Car with data members to store ownership cost, warranty (by years), seating capacity and fuel type (diesel or petrol). *Bike with data members to store the number of cylinders, number of gears, cooling type(air, liquid or oil), wheel type(alloys or spokes) and fuel tank size(in inches).

Make another two subclasses Audi and Ford of Car, each having a data member to store the model type.
Next, make two subclasses Honda and Yamaha, each having a data member to store the make-type.

Now, store and print the information of an Audi and a Ford car (i.e. model type, ownership cost, warranty, seating capacity, fuel type, mileage and price.)

Do the same for a Honda and a Yamaha bike.

Note: The solution below is very comprehensive and hence if portions of code in student's answer are true or partially right, marks can be awarded to the student.

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

class Vehicle{
private:
	float mileage;
	float price;
public:
	Vehicle(){}

	Vehicle(float mileage,float price){
		this->mileage=mileage;
		this->price=price;
	}

	float getMileage(){
		return this->mileage;
	}
	float getPrice(){
		return this->price;
	}

	virtual void display(){
		cout<<"Mileage: "<<mileage<<"\n";
		cout<<"Price: "<<price<<"\n";
	}
};

class Car:public Vehicle{
private:
	float ownershipCost;
	int warranty;
	int seatingCapacity;
	string fuelType;
public:
	Car(){}
	Car(float mileage,float price,float ownershipCost,int warranty,int
seatingCapacity,string fuelType):Vehicle(mileage,price){
		this->ownershipCost=ownershipCost;
		this->warranty=warranty;
		this->seatingCapacity=seatingCapacity;
		this->fuelType=fuelType;
	}
	float getOwnershipCost(){
		return this->ownershipCost;
	}
	int getWarranty(){
		return this->warranty;
	}
```

```
        int getSeatingCapacity(){
                return this->seatingCapacity;
        }
        string getFuelType(){
                return this->fuelType;
        }
        void display(){
                cout<<"Car\n";
                Vehicle::display();
                cout<<"Ownership Cost: "<<ownershipCost<<"\n";
                cout<<"Warranty: "<<warranty<<"\n";
                cout<<"Seating capacity: "<<seatingCapacity<<"\n";
                cout<<"Fuel type: "<<fuelType<<"\n";
        }
};
class Bike:public Vehicle{
private:
        int numberCylinders;
        int numberGears;
        string coolingType;
public:
        Bike(){}
        Bike(float mileage,float price,int numberCylinders,int numberGears,string
coolingType):Vehicle(mileage,price){
                this->numberCylinders=numberCylinders;
                this->numberGears=numberGears;
                this->coolingType=coolingType;
        }
        int getNumberCylinders(){
                return this->numberCylinders;
        }
        int getNumberGears(){
                return this->numberGears;
        }
        string getCoolingType(){
                return this->coolingType;
        }
        void display(){
                cout<<"Bike\n";
                Vehicle::display();
                cout<<"Number cylinders: "<<numberCylinders<<"\n";
                cout<<"Number gears: "<<numberGears<<"\n";
                cout<<"Cooling type: "<<coolingType<<"\n";
        }
};

class Audi:public Car{
private:
        string modelType;
public:
```

```
        Audi(float mileage,float price,float ownershipCost,int warranty,int
seatingCapacity,string fuelType):
        Car(mileage,price,ownershipCost,warranty,seatingCapacity,fuelType){
                this->modelType="Audi";
        }
        void display(){
                Car::display();
                cout<<"Model type: "<<modelType<<"\n";
        }
};
class Ford:public Car{
private:
        string modelType;
public:
        Ford(float mileage,float price,float ownershipCost,int warranty,int
seatingCapacity,string fuelType):
        Car(mileage,price,ownershipCost,warranty,seatingCapacity,fuelType){
                this->modelType="Ford";
        }
        void display(){
                Car::display();
                cout<<"Model type: "<<modelType<<"\n";
        }
};
class honda:public Bike{
private:
        string makeType;
public:
        honda(float mileage,float price,int numberCylinders,int numberGears,string
coolingType):
        Bike(mileage,price,numberCylinders,numberGears,coolingType){
                this->makeType="Honda";
        }
        void display(){
                Bike::display();
                cout<<"The make-type: "<<makeType<<"\n";
        }
};


class yamaha:public Bike{
private:
        string makeType;
public:
        yamaha(float mileage,float price,int numberCylinders,int numberGears,string
coolingType):
        Bike(mileage,price,numberCylinders,numberGears,coolingType){
                this->makeType="Yamaha";
        }
        void display(){
```

```
                Bike::display();
                cout<<"The make-type: "<<makeType<<"\n";
        }
};

int main()
{
        Audi carAudi(10000,36000,29000,10,3,"diesel");
        Audi carFord(20000,26000,19000,5,1,"petrol");
        honda bikeHonda(16000,12000,1,1,"liquid");
        yamaha bikeyamaha(56000,52000,6,6,"liquid");
        carAudi.display();
        cout<<"\n";
        carFord.display();
        cout<<"\n";
        bikeHonda.display();
        cout<<"\n";
        bikeyamaha.display();
        return 0;
}
```