CS1004 Object Oriented Programming

Saturday April 08, 2023

Course Instructor

Mr. Rizwan Ul Haq, Mr. Ch. Usman Ghous, Dr. Bilal Khan, Dr. Imran Babar, Dr. Khalid Hussain, Mr. Hafiz Saud

Serial No:

2nd Mid Term Exam

Total Time:1 Hour

Total Marks: 45

____

_____

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 two (03) questions printed on eight (08) pages including this title page.
2. Attempt all questions in a space given against each question in the given order clearly. *Cutting and overwriting will not be graded*.
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 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	Total
Total Marks	15	15	15	**45**
Marks Obtained				

Vetted By: _____ **Vetter Signature:** _____

University Answer Sheet Required: **No** ☐ **Yes** ☐

Q1. CLO-2	**5+2+2+3+3 = 15 Marks**

Consider a student class with id and gpa as int and float data members respectively. The class has default constructor (default value of id and gpa are 123 and 2.5 respectively) and other three methods, **setID**, **setGPA**, and **show** functions to set id gpa and display them. In the main, call all class three methods **using single statement.** The methods **prototype must** be in such a way that all methods can be called in a chain. The class must **restrict its data members** from outside access.

(a) Write the class definition with proper access specifiers and prototypes to claim any mark [data members and class body 1.5 marks, default constructor 1 marks, methods prototype and definition 2.5 marks, total 5 marks].

```
class student{
        private:
                int id;
                float gpa;
        public:
        student(int id=123, float gpa=2.5){
                this->id=id;
                this->gpa=gpa;
        }
        student& setID(int id){
                this->id=id;
                return *this;
        }
        student& setGPA(float gpa){
                this->gpa=gpa;
                return *this;
        }
        void display()
        {
                cout<<"ID is :"<<id<<"\t";
                cout<<"GPA is :"<<gpa<<endl;
        }
};
```

(b) Driver program, create an object of student and then call all setID and setGPA and then display them by calling display method. Calling **must be in a chain** and in **single statement**, otherwise zero marks [2 Marks].

```
int main(){

        student std;
        std.setID(957).setGPA(2.5).display();

return 0;
}
```

(c) Now we want to create 100 students objects and want to keep track that how many students objects are created immediately after creating objects using another class variable **count,** the count should be of such type that it can be called even when no object is created. What

else code need to be inserted in the existing class definition that guarantee the correct number of student objects created at that time, without any explicit function call [1 + 1 = 2 marks]

```
// code to add data member count

private: static int count
//code to update count immediately after every object creation
 student(int id=123, float gpa=2.5){
                this->id=id;
                this->gpa=gpa;
          count++; // count++ will be inserted into constructor body
          }
// if someone writes: insertion of count++ static variable into default constructor of
student class it's also okay
```

(d) Now suppose, there are two instances of student class obj1 and obj2, obj2 needs to replica of obj1 i.e. obj2(obj1), **which constructor** will be called for this replication, write the constructor **definition** with proper argument [1 + 2 = 3 marks]

```
copy constructor will be called

        student(student& obj){
                this->id=obj.id;
                this->gpa=obj.gpa;
        }
```

(e) Suppose we have a class defined below, write the output of this program [3 marks]

```
#include <iostream>
using namespace std;
    class student{
    private:
            int id;
            float *gpa;

public:

student(int id=222, float  gpa=2.5){
            this->id=id;
        this->gpa=new float;
         *(this->gpa)=gpa;
}

    void updateGPA(float GPA)
{         *gpa= GPA;    }
```

```
    student(student& s){
        this->id=s.id;
         this->gpa=s.gpa;
    }
};

    int main()
    {
        student s1;
        student s2=s1;
        s2.updateGPA(3.5);
        cout<<s2. getGPA ()<<endl;
        cout<<s1. getGPA ()<<endl;
        return 0;
}
```

What will be the output:?
3.5
3.5

Q2. CLO-2	**2 + 2 + 2 + 5 + 4 = 15 Marks**

Assume a Teacher and a Supervisor class is directly derived from the Employee class. An Employee class has an employee_id of type int and an employee_name of type string as private members. Similarly, a Teacher class has subject_name of type string as a private member. Also Supervisor has total_student of type int as private members. All the three classes provide getter functions to return the values of their members if the getter function is invoked on its corresponding objects.
You are required to write a program for the below:

(a) A complete Employee class with required constructor(s) and the getter function. [2 marks]

```
class employee  {
public:
employee(string _name="", int
_id=0): name(_name),id(_id)
{}
void getEmployee() {
     cout<< id<<"   "<<name; }
private:

int id;
string name;

};
```

(b) A complete Teacher class with required constructor(s) and the getter function. [2 marks]

```
class Teacher: public virtual
employee
{
public:
    Teacher(string       sub):
subject(sub) {}

    void        getTeacher()
{cout<<subject;}

private:
string subject;

};
```

(c) A complete Supervisor class with the required constructor(s) and the getter function [2 marks]

```
class    Supervisor:    public
virtual employee
{
public:
   Supervisor(int
total_st):students(total_st){
     }
     void      getSupervisor()
{cout<<students; }

private:
int students;

};
```

(d) A class named Derived, which is derived directly from classes Teacher and Supervisor. It provides constructor(s) and a single getter function, which will display all the information of the base classes from which it is directly and indirectly derived. [5 marks].

```
class Derived:Teacher,Supervisor {
public:
        Derived(int total_st, string _sub, int
_id, string _name): employee(_name,_id),
Supervisor(total_st), Teacher(_sub) { }
void getDerived() {

        cout<<"\nEmployee ID and name: ";
        getEmployee();
        cout<<"\n    Teacher Subjects: ";
        getTeacher();
        cout<<"\n   Supervisor Students: ";
        getSupervisor();
        }

private:


};
```

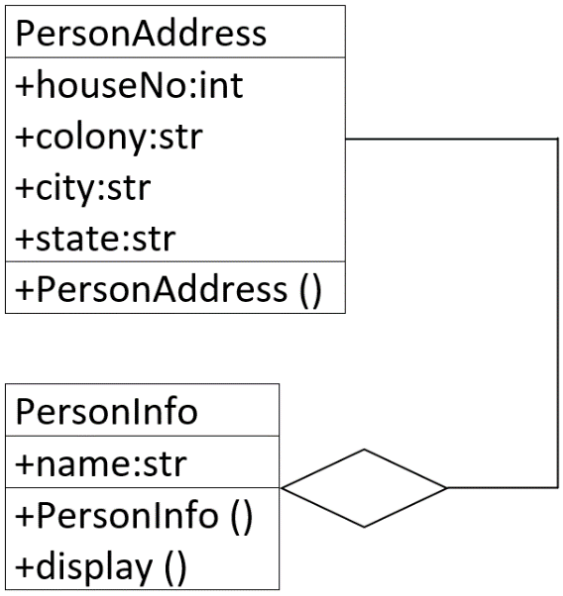
(e) In main() function, (i) create a static array of 5 Derived class objects. Furthermore, you will have to provide all the necessary information to the objects at the time of their creation. There is no need to provide information via console. In fact, there is no setter function in anyone of the classes. (ii) Use the getter function provided by the Derived class to display all the information in a descent way. [2+2 marks]

```
int main() {
  Derived d1[5] = {{Derived(10,
"MATHS", 1, "Hanif")},{Derived(11,
"ENGLISH", 2, "Rizwan")},{Derived(12,
"OOP", 3, "Asad")},{Derived(13,
"CALCULUS", 4, "Saba")},{Derived(14,
"Algebra", 5, "Sarah")}};


        for (int i=0;i<5;i++) {
        d1[i].getDerived();
        }
}
```

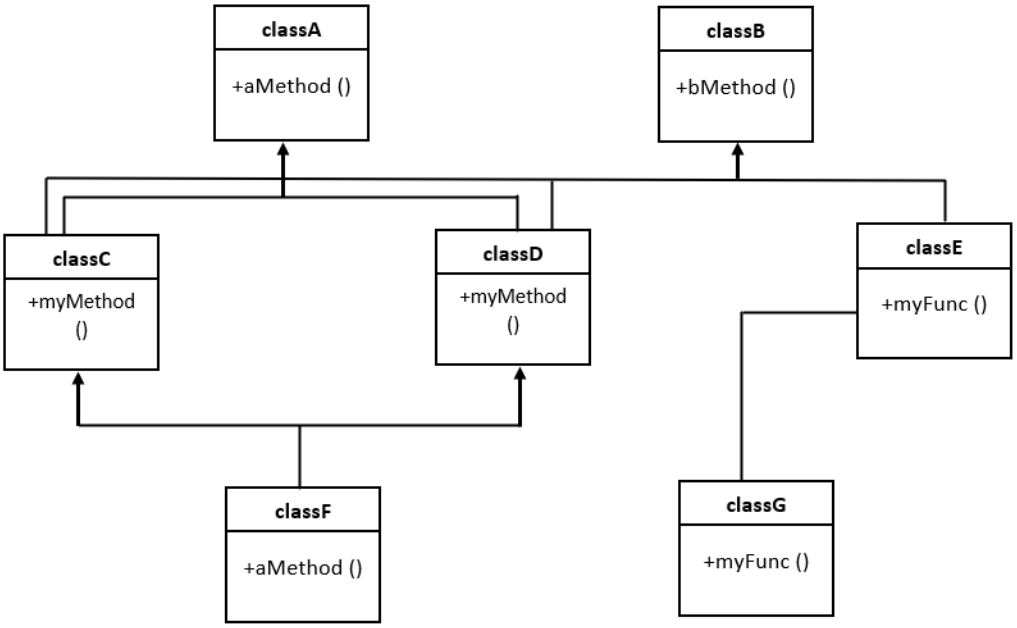
Q3. CLO-3	**8 + 7 = 15 Marks**

a) Implement the given aggregation scenario for the class diagram as shown below. The code must map with the class diagram. **Marks: [data member (2) + member functions (3) + aggregation (3)]**



```
class PersonAddress{
public:
int houseNo;
string colony, city, state;        //MARKS[1]
PersonAddress(int hno, string colony, string
city, string state){
this->houseNO=hno;
this->colony=colony;
this->city=city;
this->state=state;}                //MARKS[1]
};
class PersonInfo{
private:
PersonAddress *address;            //MARKS[1]
public:
string name;
PersonInfo(string name, PersonAddress
*address){
this->name=name;
this->address=address;}            //MARKS[1]
void
display(){cout<<name<<address-
>houseNo<<address->colony<<address-
>city<<address->state;}
          //MARKS[1]
};
int main(){                        //MARKS[3]
PersonAddress
objAdd=PersonAddress("parameters");
PersonInfo p1=Person("Imran's Address:",
&objAdd);
PersonInfo p2=Person("Ali's Address:",
&objAdd);
p1.display();
p2.display();
retrun 0;
}
```


b) Draw the code structure for the following class diagram. Call methods of base classes in child classes by keeping in view the concepts of inheritance, association and method overriding. **[Marks: 7]**




```
class classA{
public:
aMethod();
};
class classB{
public:
bMethod();
};
class classC: public classA, public classB{
public:
myMethod();
};
class classC: public classA, public classB{
public:
myMethod();
};
class classF: public classC, public classD{
public:
aMethod();
};
class classE: public classB{
public:
myFunc();
};
```

```
class classG{
classE objclassE;                    //Association
public:
myFunc();
};                                   //MARKS [2]
int main(){
//method calling through class objects        //MARKS[1]
classF objclassF;
objclassF.aMethod();                 //method of classF will be called
classA objclassA;
objclassA.aMethod();                 //method of classA will be called

//Method Overriding                  //MARKS [2]
classA *objclassA;
objclassA = &objclassF;
objclassA->aMethod();                //method of classA will be called as the pointer is of
the
//type classA
objclassA->aMethod();                //method of classF will be called if base class
aMethod is
//virtual
//Diamond Problem Solution           //MARKS[2]
objclassF.classC::aMethod();         //Diamond problem solution through path selection,
method
//of classA will be called
objclassF.classD::aMethod();         //Diamond problem solution through path selection,
method
//of classA will be called
classG objclassG;
objclassG.myFunc();                  //Method of classG will be called
}
```

If a student has used disinheritance method or path selection method for diamond problem the marks will be 2 in both of the cases.

```
//Concept of Disinheritance to resolve diamond problem keeping in view the methods are
still overridden
class classF: virtual public classC, virtual public classD{
public:
aMethod();
                                     };
```