CL103 COMPUTER PROGRAMMING

LAB 09

FRIEND FUNCTION, COMPOSITION & AGGREGATION

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

FRIEND FUNCTION

A friend function of a class is defined outside that class's scope, yet has the right to access the non-public (and public)members of the class. Standalone functions, entire classes or member functions of other classes may be declared to be friends of another class.

EXAMPLE

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

class Box
{
        double width;
        public:
        friend void printWidth( Box box );
        void setWidth( double wid );
};

// Member function definition
void Box::setWidth( double wid )
{
        width = wid;
}

// Note: printWidth() is not a member function of any class.
void printWidth( Box box )
{
        /* Because printWidth() is a friend of Box, it can   directly access any member of this class */
        cout << "Width of box : " << box.width <<endl;
}

// Main function for the program
int main( )
{
        Box box;
        box.setWidth(10.0);  // set box width with member function
        printWidth( box );    // Use friend function to print the wdith.
        return 0;
}
```

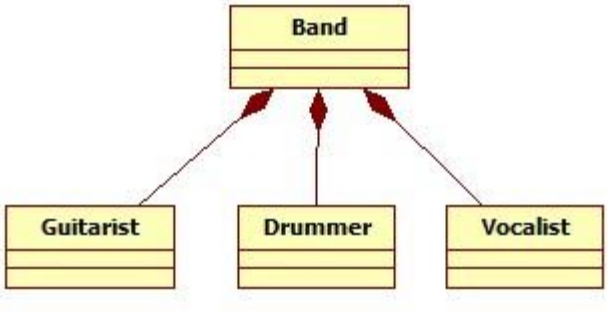
COMPOSITION – HAS A RELATIONSHIP

- Composition is a relationship among classes by which a class can be made up of any combination of objects of other classes. It allows objects to be placed directly within the body of other classes.
- It is the whole/part relationship in which the lifetime of the 'part' is controlled by the 'whole'.
- Composition is referred as "has–a" relationship, with the ability to navigate from the whole to its parts.

EXAMPLE

We have a class called Band. The data members of Band consist of objects from the Guitarist, Drummer, and Vocalist classes. These objects are data members of the Band class, but not descendants or parent classes. They are related by composition, not by inheritance.

UML REPRESENTATION – FILLED DIAMOND



CODE IMPLEMENTATION

```
class Band
{
        Guitarist gtr;
        Drummer drm;
        Vocalist voc;
};
```

```
class Guitarist
{
};
```

```
class Drummer
{
};
```

```
class Vocalist
{
};
```

"Is A" vs. "Has A" Relationships

Inherited classes are commonly described as having an "is a" relationship with each other while composition involves a "has a" relationship between classes. We cannot say that Guitarist *is a* Band. Instead, we say that Band *has a* Guitarist.

AGGREGATION

Aggregation is a closely related concept to composition. Aggregation models a whole part relationship between an aggregate (the whole) and its parts. When an aggregate is destroyed, the sub objects are not destroyed.

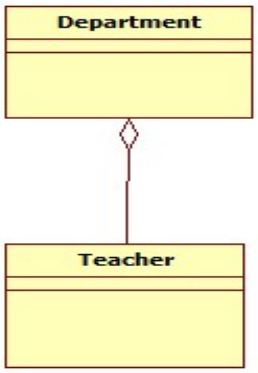
In an aggregation, we also add other subclasses to our complex aggregate class as member variables. However, these member variables are typically either references or pointers that are used to point at objects that have been created outside the scope of the class. Consequently, an aggregate class usually either takes the objects it is going to point to as constructor parameters, or it begins empty and the sub objects are added later via access functions or operators.

Because these subclass objects live outside of the scope of the class, when the class is destroyed, the pointer or reference member variable will be destroyed, but the subclass objects themselves will still exist.

EXAMPLE

Let's take an example of relationship between Department and Teacher. A Teacher may belong to multiple departments. Hence Teacher is a part of multiple departments. But if we delete a Department, Teacher Object will not destroy.

UML REPRESENTATION – HOLLOW DIAMOND



CODE IMPLEMENTATION

```
class Department
{
   private:
   Teacher *teacher
   public:
   Department(Teacher *cs_teacher=NULL)
      : teacher(cs_teacher)
   {
   }
 };
```

```
class Teacher
{
   private:
   string name;
   public:
   Teacher(string strName): name(strName)
   {
   }
   string GetName() { cout<< name; }
};
```

```
//TestAggregation.cpp
int main()
{
   // Create a teacher outside the scope of the Department
   Teacher *ptrTeacher = new Teacher("Dummy"); // create a teacher
   {
      // Create a department and use the constructor parameter to pass
      // the teacher to it.
      Department csDept(ptrTeacher);

   } // cDept goes out of scope here and is destroyed

   // pTeacher still exists here because cDept did not destroy it
   delete ptrTeacher;
}
```

LAB 09 EXERCISES

INSTRUCTIONS:

NOTE: Violation of any of the following instructions may lead to the cancellation of your submission.

1) Create a folder and name it by your student id (k15-1234).
2) Paste the .cpp file for each question with the names such as Q1.cpp, Q2.cpp and so on into that folder.
3) Submit the zipped folder on slate.

QUESTION#1

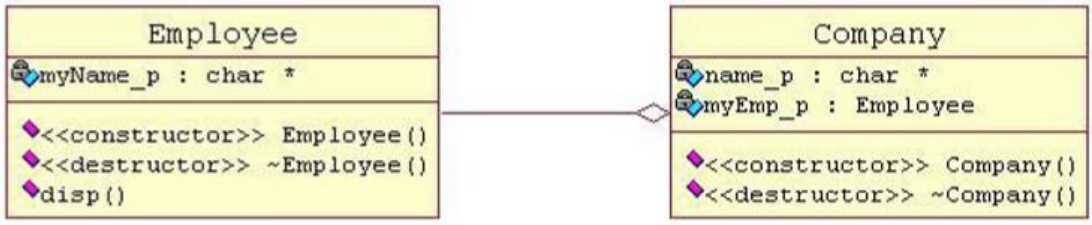
Human body is made up of limbs, heart and brain. Each of them plays a vital role such as,

- Heart pumps the blood.
- Brain receives the messages.
- Limbs make the required movements.

Can limbs, heart or brain exist without a human body? Identify the relationship between these classes and provide the implementation in C++.

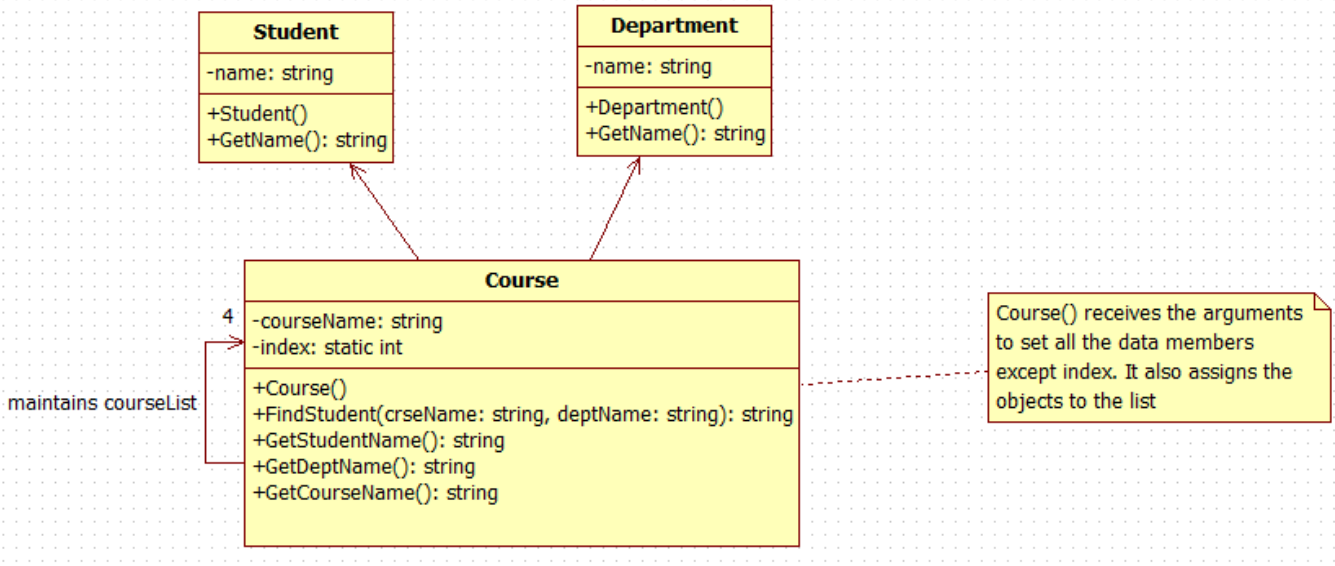
QUESTION#2

A single Employee can not belong to multiple Companies (legally!!), but if we delete the Company, Employee object will not destroy. Identify the relationship between the classes and Implement the given scenario using C++.



QUESTION#3

Implement the following scenario in C++



Provide proper code that works for the following main():

```
int main()
{
   int i;
   std::cout<<"We have got 4 students.\n";
   Student *studentNames[4] = {new Student("Sakeena"), new Student("Jeena"), new Student("Teena"), new Student("Meena")} ;
   std::cout<<"\n";
   std::cout<<"We have got 2 Departments...\n";
   Department *departNames[2] = {new Department("Mathemetics"), new Department("ComputerSceince")} ;
   std::cout<<"\n";
   std::cout<<"Here class Course Associates Student and Department, with a Course name ...\n";
   Course course1("DataStructure",studentNames[0], departNames[1]);
   Course course2("Maths",studentNames[3], departNames[0]);
   Course course3("Geometry",studentNames[2], departNames[0]);
   Course course4("CA",studentNames[1], departNames[1]);
   std::cout<<"\n";
   std::cout<<"Finding a Student using Course and Department...\n";
   std::cout<<"Student who has taken Maths Course in Mathemetics Department is:"<<Course::findStudent("Maths",
"Mathemetics")<<std::endl;
   return 0;
}
```

QUESTION#4

Design a "Bill" class that displays a list of hard coded products (normal and discounted) along with their prices through constructor. The list should be as follows:

Ordinary Products:	**Price**
Butter	250.00/-
Crackers	20.23/-
Milk	80.23/-
Biscuits	50/-

Discounted Products:	**Price**
Cheese	340/-
Baking Powder	53.25/-
Juice	70.236/-

"Bill" contains a function InputItems() that enables the user to enter the products he wants to buy along with the price and quantity. If the entered product doesn't exist in the hard coded list, the user must be prompted about the unavailability. All the normal products entered by the user are stored in the grid (shoppingCart) while the discounted ones are stored in another grid (discountedShoppingCart). Both the grids are protected. The storing should be in the following format.

Product	**Price**	**Quantity**
Butter	250.00/-	2

Two classes; "OrdinaryBill" and "DiscountedBill" extend the functionality of "Bill".

"OrdinaryBill" has a function "CalculateOrdinaryBill()" which uses the "shoppingCart**"** grid and calculates the bill by multiplying the quantity and the corresponding prices. The result must be stored in a protected data member "OrdinaryBill".

"DiscountedBill" has a function "CalculateDiscountedBill()" which uses the "discountedshoppingCart**"** grid and calculates the bill by multiplying the quantity and the corresponding prices. A discount of 20% is offered on the final discounted bill. The result must be stored in a protected data member "DiscountedBill".

Another class “BillReceipt” which enhances the functionality of both the classes “OrdinaryBill” and “DiscountedBill” simultaneously has a function “GenerateReceipt()” which calculates the total bill by adding “OrdinaryBill” and “DiscountedBill”. The function then displays both the grids and the total amount.

Display the result when the user enters 3 ordinary products and 2 discounted products.

InputItems() must be called from main().

All the data members must be private if not explicitly mentioned while all the member functions must be public if not explicitly mentioned.

QUESTION#5

All the data members must be private if not explicitly mentioned while all the member functions must be public if not explicitly mentioned.

Design a class “Product” that holds and displays a list of hard coded products along with their prices through constructor. The list should be as follows:

Dairy Products:	**Price**
Milk	80.23/-
Cheese	347.52/-
Butter	50.36/-
Yogurt	50/-

Beverages:	**Price**
Coffee	85.00/-
Tea	56.58/-
Juice	70.236/-

Personal Care Products:	**Price**
Shampoo	103.63/-
Soap	50.23/-
Hand wash	70.236/-

“Product” contains a container (shoppingCart) which is protected.

Design the classes; “DairyProducts”, “Beverages” and “PersonalCareProducts”.All of them **are** “Products” and contain one function; “AddDairyProduct()”, “AddBeverages()” and “AddPersonalCareProduct()” repectively. These functions take the product’s name as argument and add it to “shoppingCart”.

“Product” contains a function, “CalculateBill()” that calculates the bill by matching the products in “shoppingcart” and the hard coded list in order to obtain the prices. If the “shoppingCart” contains two or more similar products e.g. two similar dairy products then the customer is only charged for one such product as a discount. This bill is stored in “bill” which is private.

“Product” has another immutable function “DisplayProductsBill()” that displays the added products along with the final bill.

Another class “ShoppingCart” **contains** “Product” class along with all its sub types in such a manner that both cannot exist independently. This class should only contain public interface.

The main() function must be designed as follows:

1) Create an instance “obj” of “ShoppingCart”.
2) Display the hard coded list of products without calling any function.
3) Add the following products into the “shoppingCart” by calling the appropriate functions through “obj”.
    - Milk
    - Soap
    - Milk
    - Soap
    - Soap
    - Juice
4) Calculate the bill and display it by calling appropriate functions of “Product” class.

QUESTION#6

- Do not hard-code anything, if not explicitly mentioned .
- All the data members must be private while all the member functions must be public if not explicitly mentioned.
- You can add constructors if required.
- You can add extra data members if required but not extra member functions.

Design a class “Product” that holds and displays a list of hard coded products along with their prices through the function “DisplayProductsList()”. The list should be as follows:

Dairy Products:	**Price**
Milk	80.23/-
Cheese	347.52/-
Butter	50.36/-
Yogurt	50/-

Beverages:	**Price**
Coffee	85.00/-
Tea	56.58/-
Juice	70.236/-

Personal Care Products:	**Price**
Shampoo	103.63/-
Soap	50.23/-
Hand wash	70.236/-

“Product” contains a container (productsCart) which is protected.

Design the classes; “DairyProducts”, “Beverages” and “PersonalCareProducts”.All of them **are** “Products” and contain one function; “AddDairyProduct()”, “AddBeverages()” and “AddPersonalCareProduct()” repectively. These functions take the product’s name as argument and add it to “productsCart”.

“Product” contains a function, “CalculateBill()” that calculates the bill by matching the products in “productsCart” and the hard coded list in order to obtain the prices. If the “productsCart” contains two or more similar products e.g. two similar dairy products then the customer is only charged for one such product as a discount. This bill is stored in “bill” which is private.

“Product” has another immutable function “DisplayProductsBill()” that displays the added products(i.e the “productsCart”) along with the final bill.

Another class “ShoppingCart” **has** “Product” class along with all its sub types in such a manner that they can exist independently. This class should only contain public interface.

The following instructions must be followed while designing the main() function:

5) It must contain only one instance i.e. “obj” of “ShoppingCart”.
6) It must display the hard coded list of products by calling the appropriate function.
7) It must add the following products into the “productsCart” by calling the appropriate functions.
    - Milk
    - Soap
    - Milk
    - Soap
    - Soap
    - Juice
8) Calculate the bill and display it along with the products by calling appropriate functions of “Product” class.