CL103
COMPUTER PROGRAMMING

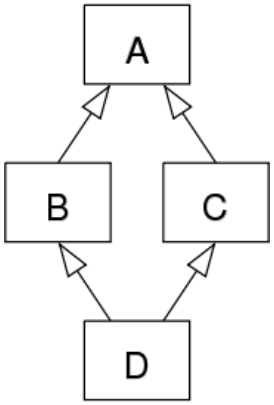
LAB 10

Diamond Problem, Casting & Run time Polymorphism

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

DIAMOND PROBLEM

In case of hybrid inheritance a Diamond problem may arise. The “dreaded diamond” refers to a class structure in which a particular class appears more than once in a class’s inheritance hierarchy.



EXAMPLE:

```
class Animal
{
public:
void walk()
{
cout<<”animal walks”;
}

};
```

```
class Donkey : public Animal
{

};
```

```
int main()
{
Mule obj;
obj.walk();
return 0;
};
```

```
class Horse : public Animal
{

};
```

```
class Mule : public Donkey, public Horse
{

};
```

Call is ambiguous; the compiler can't know which implementation of **walk()** to use, the one from the **Donkey** class or the one from the **Horse** class.

HOW TO SOLVE DIAMOND PROBLEM? – VIRTUAL BASE CLASS INHERITANCE

```
class Animal
{
public:
void walk()
{
cout<<"animal walks";
}

};
```

```
class Donkey : virtual public Animal
{

};
```

```
int main()
{
Mule obj;
obj.walk();
return 0;
};
```

```
class Horse : virtual public Animal
{

};
```

```
class Mule : public Donkey, public Horse
{

};
```

When we use virtual inheritance, we are guaranteed to get only a single instance of the common base class. In other words, the **Mule** class will have only a single instance of the **Animal** class, shared by both the **Donkey** and **Horse** classes. By having a single instance of **Animal**, we've resolved the compiler's immediate issue, the ambiguity, and the code will compile fine.

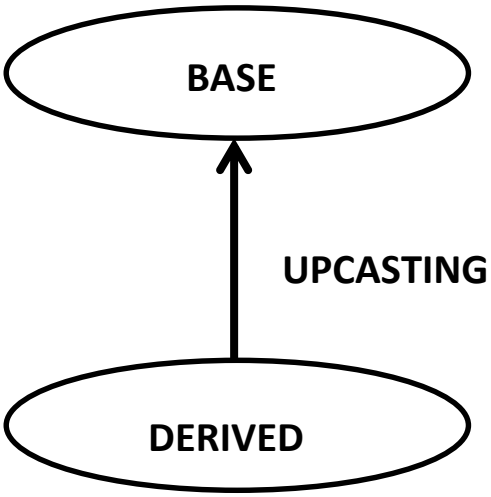
CASTING

A cast is a special operator that forces one type to be converted into another.

UPCASTING

Upcasting is a process of treating a pointer or a reference of derived class object as a base class pointer.

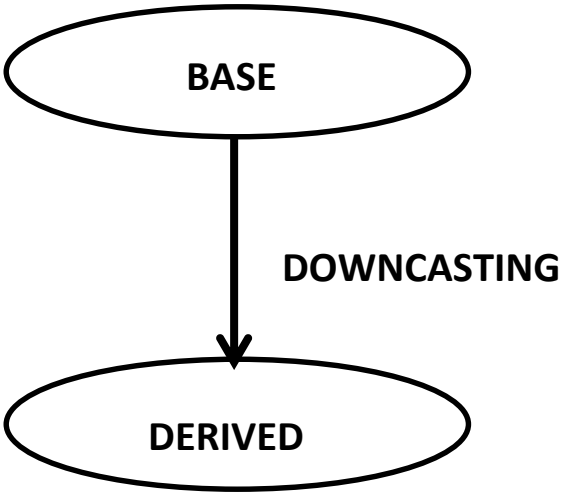
- A base class pointer can only access the public interface of the base class.
- The additional members defined in the derived class are therefore inaccessible.
- Upcasting is not needed manually. We just need to assign derived class pointer (or reference) to base class pointer.



DOWNCASTING

The opposite of Upcasting is Downcasting, It converts base class pointer to derived class pointer.

- Type conversions that involve walking down the tree, or *downcasts*, can only be performed explicitly by means of a cast construction. The cast operator **(type)**or the **static_cast< >** operator are available for this task, and are equivalent in this case.
- After downcasting a pointer or a reference, the entire public interface of the derived class is accessible.



STATIC CAST

Syntax: static_cast<type>(expression)

The operator static_cast< > converts the expression to the target type type.

EXAMPLE CODE (UPCASTING AND DOWNCASTING)

```
#include <iostream>
using namespace std;
class Employee
{
public:
	Employee(string fName, string lName, double sal)
	{
		FirstName = fName;
		LastName = lName;
		salary = sal;
	}
	string FirstName;
	string LastName;
	double salary;
	void show()
	{
		cout << "First Name: " << FirstName << " Last Name: " << LastName << " Salary: " << salary<< endl;
	}
	void addBonus(double bonus)
	{
		salary += bonus;
	}
};

class Manager :public Employee
{
public:
	Manager(string fName, string lName, double sal, double comm) :Employee(fName, lName, sal)
	{
		Commision = comm;
	}
	double Commision;
	double getComm()
	{
		return Commision;
	}
};
```

FOR UPCASTING

```
int main()
{
	Employee* emp;		//pointer to base class object

	Manager m1("Ali", "Khan", 5000, 0.2);  //object of derived class

	emp = &m1;  //implicit upcasting

	emp->show();  //okay because show() is a base class function
	return 0;
}
```

FOR DOWNCASTING USING (type)

```
int main()
{
	Employee e1("Ali", "Khan", 5000);  //object of base class

	//try to cast an employee to Manager
	Manager* m3 = (Manager*)(&e1);  //explicit downcasting

	cout << m3->getComm() << endl;
	return 0;
}
```

FOR DOWNCASTING USING (static_cast)

```
int main()
{
	Employee e1("Ali", "Khan", 5000);  //object of base class

	//try to cast an employee to Manager
	Manager* m3 = static_cast<Manager*>(&e1);  //explicit downcasting

	cout << m3->getComm() << endl;
	return 0;
}
```

DOWNCASTING IS UNSAFE

- **Since, e1 object is not an object of Manager class so, it does not contain any information about commission.**
- **That's why such an operation can produce unexpected results.**
- **Downcasting is only safe when the object referenced by the base class pointer really is a derived class type.**
- **To allow safe downcasting C++ introduces the concept of *dynamic casting.***

POLYMORPHISM

Polymorphism refers to the ability of a method to be used in different ways, that is, it can take different forms at different times (poly + morphos).

TYPES OF POLYMORPHISM

There are two types of polymorphism:

- Compile time polymorphism
- Run time polymorphism.

RUN TIME POLYMORPHISM

Run time (dynamic) polymorphism occurs when the methods itself are changed/overridden.

If we inherit a class into the derived class and provide a definition for one of the base class's function again inside the derived class, then that function is said to be overridden, and this mechanism is called Function Overriding. Calling a virtual method makes the compiler execute a version of the method suitable for the object in question, when the object is accessed by a pointer or a reference to the base class.

REQUIREMENTS FOR OVERRIDING

- Inheritance should be there. Function overriding cannot be done within a class. For this we require a derived class and a base class.
- Function that is redefined must have exactly the same declaration in both base and derived class, that means same name, same return type and same parameter list.

EXAMPLE CODE (VIRTUAL FUNCTIONS)

Virtual Function is a function in base class, which is overrided in the derived class, and which tells the compiler to perform Late Binding on this function.

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

class Shape
{
	public:
	 virtual void Draw()
	{
		cout<<"Shape drawn!"<<endl;
	}
};

class Square : public Shape
{
	public:
	void Draw()
	{
		cout<<"Square drawn!"<<endl;
	}
};

int main()
```

```
{
        Square obj;       //Derived Class Object

        Shape* shape = &obj;  /* derived class object being referenced by a pointer to the
                                                base class */

        shape->Draw();      /* outputs "Square drawn!" if Draw() is virtual in base class else outputs
                                "Shape drawn!" */
        return 0;
}
```

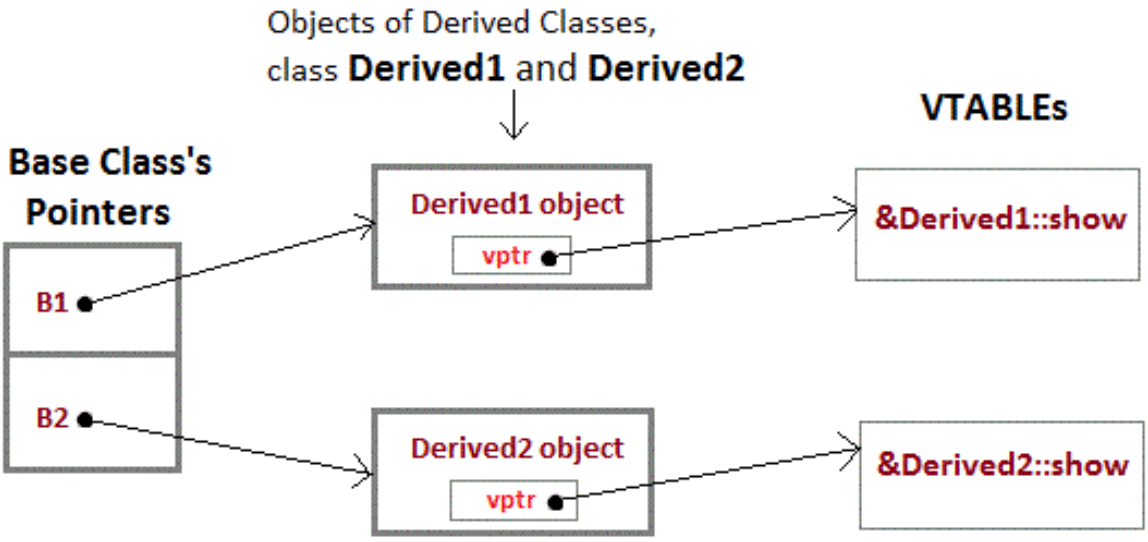
INTERESTING FACTS

We can call private function of derived class from the base class pointer with the help of virtual keyword. Compiler checks for access specifier only at compile time. So at run time when late binding occurs it does not check whether we are calling the private function or public function.

MECHANISM OF LATE BINDING

To accomplish late binding, Compiler creates VTABLEs, for each class with virtual function. The address of virtual functions is inserted into these tables. Whenever an object of such class is created the compiler secretly inserts a pointer called vpointer, pointing to VTABLE for that object. Hence when function is called, compiler is able to resolve the call by binding the correct function using the vpointer.

- Only the Base class Method's declaration needs the Virtual Keyword, not the definition.
- If a function is declared as virtual in the base class, it will be virtual in all its derived classes.
- The address of the virtual Function is placed in the VTABLE and the compiler uses VPTR(vpointer) to point to the Virtual Function.



vptr, is the vpointer, which points to the Virtual Function for that object.

VTABLE, is the table containing address of Virtual Functions of each class.

LAB 10 EXERCISES

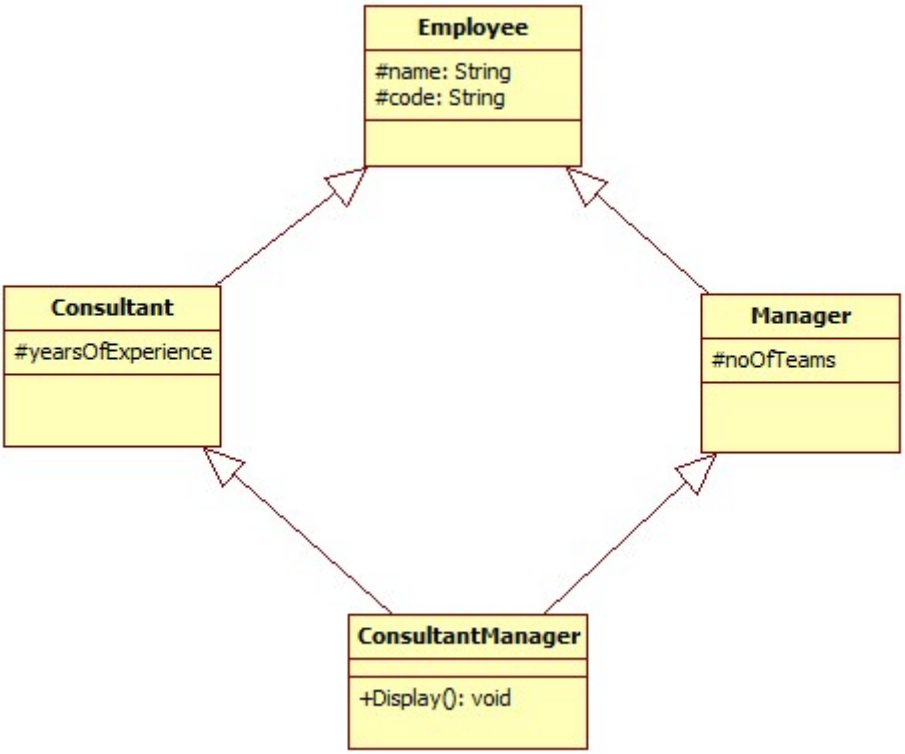
INSTRUCTIONS:

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

Implement the following scenario in C++:



1) No accessors and mutators are allowed to be used.

2) The Display() function in "ConsultantManager" should be capable of displaying the values of all the data members declared in the scenario (name,code,yearsOfExperience,noOfTeams) without being able to alter the values.

3) The "int main()" function should contain only three program statements which are as follows:

a) In the first statement, create object of "ConsultantManager" and pass the values for all the data members:

ConsultantManager obj("Ali",”S-123”,17,5);

b) In the second statement, call the Display() function.
c) In the third statement, return 0.

4) All the values are required to be set through constructors parameters.

QUESTION#2

Design and implement a program that shows the relationship between person, student and professor. Your person class must contain two pure virtual functions named getData() of type void and isOutstanding() of type bool and as well getName() and putName() that will read and print the person name. Class student must consist of function name getData (), which reads the GPA of specific person and isOutstanding() function which returns true if the person GPA is greater than 3.5 else should return false. Class professor should take the respective persons publications in getData() and will return true in Outstanding() if publications are greater than 100 else will return false . Your main function should ask the user either you want to insert the data in professor or student until and unless user so no to add more data.

QUESTION#3

A company pays its employees weekly. The employees are of four types: **Salaried employees** are paid a fixed weekly salary regardless of the number of hours worked, **hourly employees** are paid by the hour and receive overtime pay for all hours worked in excess of 40 hours, **commission employees** are paid a percentage of their sales and **base-salaryplus-commission employees** receive a base salary plus a percentage of their sales. For the current pay period, the company has decided to reward base-salary-plus-commission employees by adding 10 percent to their base salaries. The company wants you to draw UML diagram of the given scenrio and do implemention on C++ that performs its payroll calculations polymorphically .

Your Employee class must have

- first name, last name and social security number as a private data members. Use accesor and mutators to set and get these values.
- Constructor with fisrt name, last name and ssn number as parameter.
- Pure Virtual function named earning with return type double;
- Virtual Function named print with return type void, which employee first name,last name and ssn number.

Your Salaried Emplyee must have

- Earning method which returns the salary.
- Print method that print the employee detail and employee salary.

Your hourly Emplyee must have

- You must take the wage and hours as a parameter using base class initializer.
- Earning method which returns the salary the employed has worked.
- Print method that print the employee detail and employee salary.

Your commission employees must have

- You must take the commission rate and gross sale rate as parameter using base class initializer.
- Earning method which returns the commission of the employee.
- Print method that print the employee detail and employee commision.

Your base-salaryplus-commission employees must have

- You must take the base salary as a parameter using base class initialize.
- Earning method which returns base commission of the employee.
- Print method that print the employee detail and employeebase salary.

You must have to perform upcasting and downcasting, and print the details of each employee type.