# Object Oriented Programming (CT-260)
# Lab 05
Introduction to inheritance

**Objectives**

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context. Students will be representing solutions to different problems as flowcharts and pseudo code

The purpose of this lab is to enable students to understand the basics of problem solving in

programmatic context. Students will be representing solutions to different problems as flowcharts and pseudo code The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context. Students will be representing solutions to different problems as flowcharts and pseudo code

The objective of this lab is to familiarize students with the concept of inheritance in object oriented programming. By the end of this lab, students will be able to understand and implement various types of inheritances in C++ language.

**Tools Required**

DevC++ IDE / Visual Studio / Visual Code

Course Coordinator –
Course Instructor –
Lab Instructor –
Prepared By Department of Computer Science and Information Technology
NED University of Engineering and Technology

# INTRODUCTION TO INHERITANCE

Inheritance is one of the key features of Object-oriented programming in C++. It allows us to create a new class (derived class) from an existing class (base class).

**Base Class:**
A base class is the class from which features are to be inherited into another class.

**Derived Class:**
A derived class is the one which inherits features from the base class. It can have additional properties and methods that are not present in the parent class that distinguishes it and provides additional functionality.

**Is – A Relationship:**
Inheritance is represented by an is-a relationship which means that an object of a derived class also can be treated as an object of its base class for example, a Car is a Vehicle, so any attributes and behaviors of a Vehicle are also attributes and behaviors of a Car.

**Advantages of Inheritance:**
The main advantage of inheritance is code reusability. You can reuse the members of your base class inside the derived class, without having to rewrite the code.

**Real World Example:**
A real world example of inheritance constitutes the concept that children inherit certain features and traits from their parents. In addition, children also have their unique features and traits that distinguishes them from their parents.

**Basic syntax for Inheritance:**

```
class derived-class-name : access base-class-name {
// body of class
};
```

# MODES OF INHERITANCE BASED ON BASE CLASS ACCESS CONTROL

There are three types of inheritance with respect to base class access control:

- Public
- Private
- Protected

**Public Mode:**
If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.

**Protected Mode:**
If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.

**Private Mode:**
If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

**Note:** The private members in the base class cannot be directly accessed in the derived class, while protected members can be directly accessed.

**Syntax for public Inheritance:**

```
Class (name of the derived class) : public (name of the base class)
Class Car : public Vehicle
```

**Base Class Access Control for Public, Private and Protected:**

| Access Modifier of Base Class Member | Mode of Inheritance | | |
|---|---|---|---|
| | Public Inheritance | Private Inheritance | Protected Inheritance |
| **Public** | Public in derived class | Private in derived class | Protected in derived class |
| **Private** | Hidden in derived class | Hidden in derived class | Hidden in derived class |
| **Protected** | Protected in derived class | Hidden in derived class | Protected in derived class |

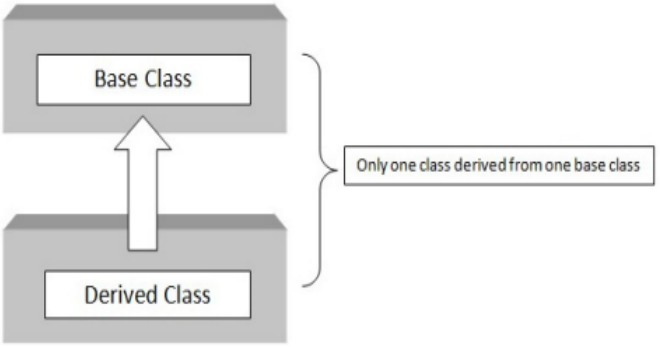
## TYPES OF INHERITANCE BASED ON DERIVED CLASSES

Inheritance based on derived classes can be categorized as follows:

- Single Inheritance
- Multiple Inheritance (will be covered after Mid Exam)
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

**Single Inheritance:**

In this type of inheritance there is one base class and one derived class. As shown in the figure below, in single inheritance only one class can be derived from the base class. Based on the visibility mode used or access specifier used while deriving, the properties of the base class are derived




**Syntax for single Inheritance:**

```
class A   // base class
{
  // body of the class
};
class B : acess_specifier A   // derived class
{
   // body of the class
};
```

**Example code for single Inheritance:**

```
#include <iostream>
using namespace std;
class base { //single base class
public:
        int x;
        void getdata( ) {
              cout << "Enter the value of x = ";
              cin >> x;
```

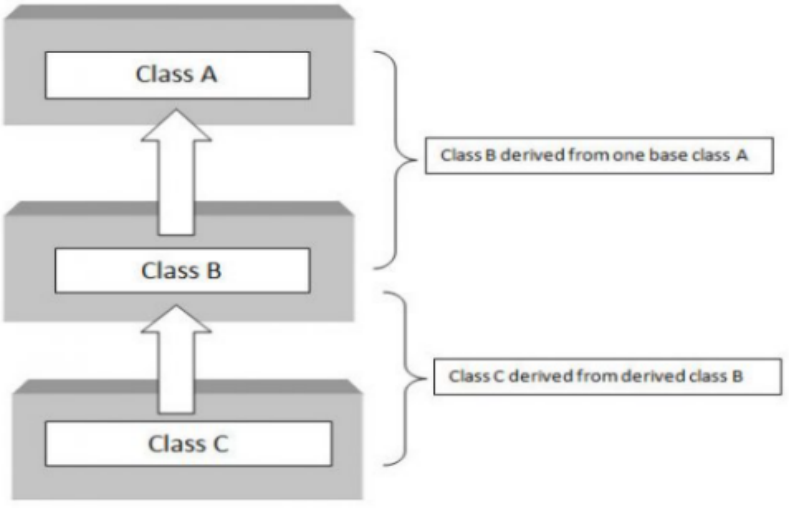
```
        }
};
class derive : public base { //single derived class
private:
        int y;
public:
        void readdata( ) {
            cout << "Enter the value of y = ";
            cin >> y;
        }
        void product( ) {
            cout << "Product = " << x * y;
        }
};
int main( ) {
        derive a; //object of derived class
        a.getdata( );
        a.readdata( );
        a.product( );
        return 0;
} //end of program
```

**Sample Run**

Enter the value of x = 3
Enter the value of y = 4
Product = 12

**Multilevel Inheritance:**

- If a class is derived from another derived class then it is called multilevel inheritance, so in multilevel inheritance, a class has more than one parent class.
- As shown in the figure below, class C has class B and class A as parent classes.
- As in other inheritance, based on the visibility mode used or access specifier used. while deriving, the properties of the base class are derived. Access specifier can be private, protected or public



**Syntax for multilevel Inheritance:**

```
class A   // base class
{
  // body of the class
};
class B : acess_specifier A // derived class
{
  // body of the class
};
class C : acess_specifier B     // derived from class B
{
    // body of the class
};
```

**Example code for multilevel Inheritance:**

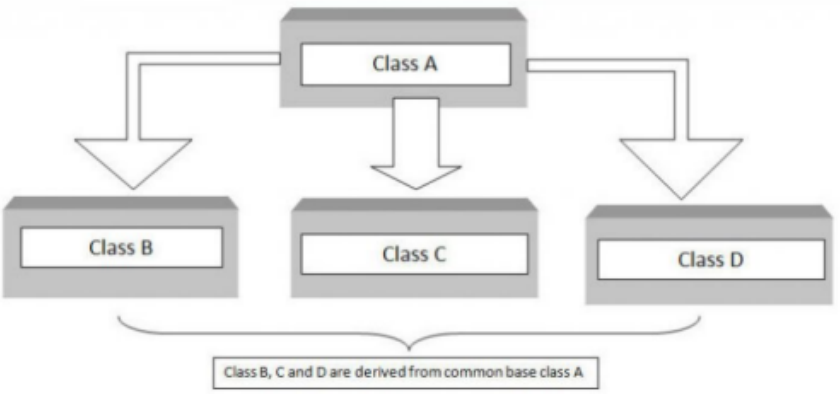
```
#include<iostream>
using namespace std;
class base {//single base class
public:
        int x;
        void getdata( ) {
              cout << "Enter value of x= ";
              cin >> x;
        }
};
class derive1 : public base { // derived class from base class
 public:
        int y;
        void readdata( ) {
              cout << "\nEnter value of y= ";
              cin >> y;
        }
 };
class derive2 : public derive1{ // derived from class derive1
private:
        int z;
public:
        void indata( ) {
             cout << "\nEnter value of z= ";
             cin >> z;
        }
        void product( ) {
              cout << "\nProduct= " << x * y * z;
        }
};
int main( ) {
        derive2 a; //object of derived class
        a.getdata( );
        a.readdata( );
        a.indata( );
        a.product( );
        return 0;
```

```
} //end of program
```

**Sample Run**
Enter value of x= 2
Enter value of y= 3
Enter value of z= 3 Product= 18

**Hierarchical Inheritance:**

When several classes are derived from a common base class it is called as hierarchical inheritance. In C++ hierarchical inheritance, the feature of the base class is inherited onto more than one sub-class. For example, a car is a common class from which Audi, Ferrari, Mustang etc can be derived. As shown in the figure below, in C++ hierarchical inheritance all the derived classes have a common base class. The base class includes all the features that are common to derived classes.




**Syntax for hierarchical Inheritance:**

```
class A   // base class
{
  // body of the class
};
class B : acess_specifier A // derived class from A
{
  // body of the class
};
class C : acess_specifier A    // derived class from A
{
   // body of the class
};
class D : acess_specifier A    // derived class from A
{
   // body of the class
};
```

**Example code for hierarchical Inheritance:**

```
#include<iostream>
using namespace std;
class A { //single base class
public:
        int x, y;
        void getdata( ) {
             cout << "\n Enter value of x and y:\n";
             cin >> x >> y;
        }
};
```

```
class B : public A { //B is derived from class base
public:
        void product( ) {
            cout << "\n Product= " << x * y;
        }
};
class C : public A { //C is also derived from class base
public:
        void sum() {
            cout << "\n Sum= " << x + y;
        }
};
int main( ) {
        B obj1; //object of derived class B
        C obj2; //object of derived class C
        obj1.getdata( );
        obj1.product( );
        obj2.getdata( );
        obj2.sum( );
        return 0;
} //end of program
```

**Sample Run**

```
Enter value of x and y: 2 3
Product= 6
Enter value of x and y: 2 3
Sum= 5
```

## Exercise

1. Create a base class with the following members:
   - Private integer privateInt
   - Protected integer protectedInt
   - Public integer publicInt

   Create getters and setters for each of these variables. Derive 3 classes from the base class with the three types of inheritance based on **visibility**(public, protected, private). You can name these classes as **publicChild**, **privateChild** and **protectedChild**. After doing this, try and figure out which member you can access publicly or through getters and setters. Then print out the way that you were able to access them. **For example**, if you did private inheritance, you could not be able to access the members in the child classes, and would need to use their getters or setters.

2. Create a class Teacher with the following private attributes: Name, Age, and Institute. Derive three classes from it that has the following names: **HumanitiesTeacher**, **ScienceTeacher**, **MathsTeacher.** These classes should have the following members:
   - Department (this should have the value "science", "maths" or "humanities")
   - Course Name
   - Designation (for example, lecturer, professor, etc)

   Create proper accessors and mutators for the attributes. Create objects for each of the classes and display the values. You can ask the user to input the values.

3. A defense organization is making a hierarchy of different types of weapons. They have classified the nuclear bomb as follows: **Weapons → Hot Weapons → Bombs → Nuclear Bombs.** Create classes and apply inheritance as necessary for the above hierarchy. Each class should have a method called: "xxxxxDescription", where xxxx would be class name. The method should print out what that weapon does. Eg. Hot weapons uses gunpowder, or explode. Bombs blow up. Nuclear bombs blow up, and use nuclear fission and fusion.
4. A bakery is making a program to calculate the bills for each of their customers. The items in the bakery are categorized as follows:

   **Item:**
   - Name
   - Quantity

   **Baked Goods: (derived from item)**
   - Discount = 10%

   **Cakes: (derived from baked goods)**
   - Price = 600

   **Bread: (derived from baked goods)**
   - Price = 200

   **Drinks: (derived from item)**
   - Discount = 5%
   - Price = 100

   In your main function, as a bakery personnel, input the items and their quantities and calculate the bill accordingly.