

Types of Inheritance

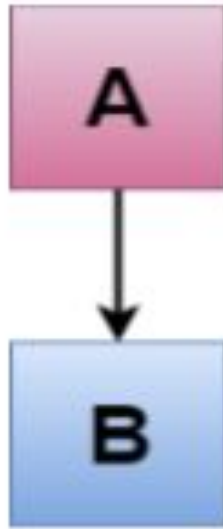
Week 7 Lec 1 & 2

Types of Inheritance

- 1. Single Inheritance.**
- 2. Multilevel Inheritance.**
- 3. Multiple Inheritance.**
- 4. Hierarchical Inheritance.**
- 5. Hybrid Inheritance.**

Single Inheritance

- Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class. i.e. one sub class is inherited by one base class only.



Syntax:

```
class subclass_name : access_mode base_class
{
    //body of subclass
};
```

Single Level Inheritance Example: Inheriting Fields

```
#include <iostream>
using namespace std;
class Account {
public:
    float salary = 60000;    };
class Programmer: public Account {
public:
    float bonus = 5000;    };
int main(void) {
    Programmer p1;
    cout<<"Salary: "<<p1.salary<<endl;
    cout<<"Bonus : "<<p1.bonus<<endl;
    return 0; }
```

Output

```
Salary: 60000
```

```
Bonus : 5000
```

```
-----
```

```
Process exited after 0.1252 seconds with return value 0
```

```
Press any key to continue . . .
```

OUTPUT?

```
#include <iostream>
using namespace std;
class Account {
public:
    float salary = 60000; };
class Programmer: public Account {
public:
    float bonus = 5000; };
int main(void) {
    Account a1;
    cout<<"Salary: "<<a1.salary<<endl;
    cout<<"Bonus : "<<a1.bonus<<endl;
    return 0; }
```

ERROR

```
#include <iostream>
using namespace std;
class Account {
    public:
    float salary = 60000; };
class Programmer: public Account {
    public:
    float bonus = 5000; };
int main(void) {
    Account a1;
    cout<<"Salary: "<<a1.salary<<endl;
    cout<<"Bonus : "<<a1.bonus<<endl;  //[Error] 'class Account' has no member named 'bonus'
    return 0; }
```


C++ Single Level Inheritance Example: Inheriting Methods

```
#include <iostream>
using namespace std;
class Animal {
public:
void eat() {
cout<<"Eating..."<<endl;
}
};
```

C++ Single Level Inheritance Example: Inheriting Methods

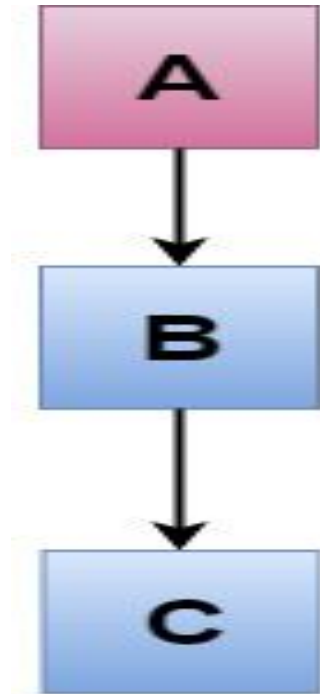
```
class Dog: public Animal {  
public:  
void bark(){  
cout<<"Barking..."; } };  
int main(void) {  
Dog d1;  
d1.eat();  
d1.bark();  
return 0;  
}
```

Output

```
Eating...  
Barking...  
-----  
Process exited after 0.129 seconds with return value 0  
Press any key to continue . . .
```

Multilevel Inheritance

- **Multilevel inheritance** is a process of deriving a class from another derived class.



Multilevel Inheritance

- When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C++. Inheritance is transitive so the last derived class acquires all the members of all its base classes.

MULTI LEVEL Inheritance Example

```
class electronicDevice
{
    public:
        // constructor of the base class 1
        electronicDevice()
        {
            cout << "I am an electronic device.\n\n";
        }
};
```

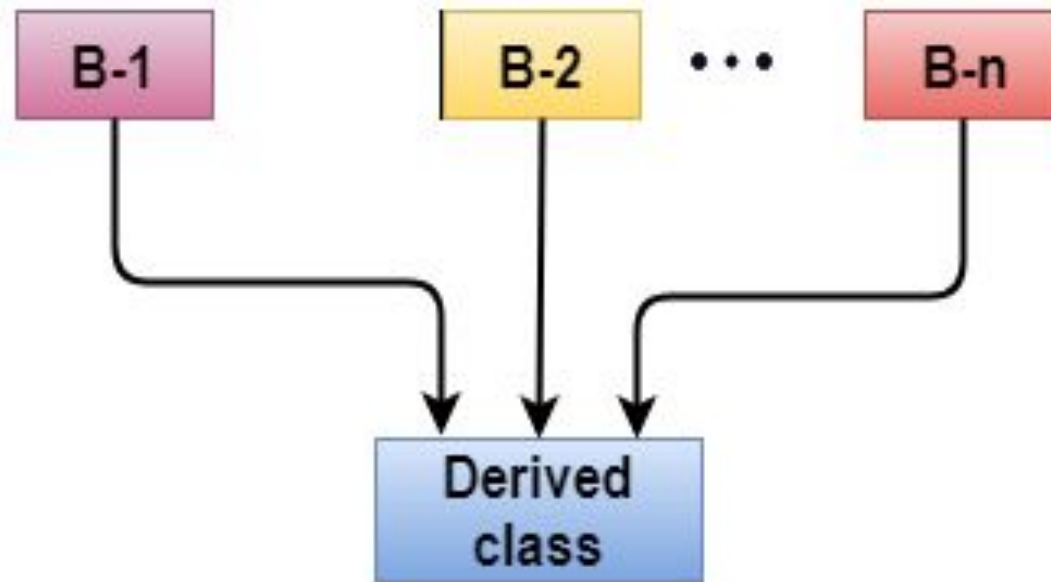
```
// class_B inheriting class_A
class Computer: public electronicDevice
{
    public:
        // constructor of the base class 2
        Computer()
        {
            cout << "I am a computer.\n\n";
        }
};
```

```
// class_C inheriting class_B
class Linux_based : public Computer
{
    public:
        // constructor of the derived class
        Linux_based()
        {
            cout << "I run on Linux.\n\n";
        }
};
```

```
int main()
{
    // create object of the derived class
    Linux_based obj; // constructor of base class 1,
                    // base class 2, derived class will be
    called
    return 0;
}
```

Multiple Inheritance

- **Multiple inheritance** is the process of deriving a new class that inherits the attributes from two or more classes.



Syntax of the Derived class:

```
class D : visibility B-1, visibility B-2
```

```
{
```

```
    // Body of the class;
```

```
}
```

Here, the number of base classes will be separated by a comma (‘, ‘) and access mode for every base class must be specified.

Multiple Inheritance Example

```
class A {  
    protected:  
    int a;  
    public:  
    void get_a(int n) {  
        a = n;} };  
class B {  
    protected:  
    int b;  
    public:  
    void get_b(int n) {  
        b = n; } };
```

Multiple Inheritance Example

```
class C : public A,public B {  
    public:  
    void display() {  
        cout << "The value of a is : " <<a<< endl;  
        cout << "The value of b is : " <<b<< endl;  
        cout<<"Addition of a and b is : "<<a+b; } };  
int main() {  
    C c;  
    c.get_a(10);  
    c.get_b(20);  
    c.display();  
    return 0;  
}
```

Ambiguity Resolution in Inheritance

- Ambiguity can be occurred in using the multiple inheritance when a function with the same name occurs in more than one base class.

Ambiguity Resolution in Inheritance

```
#include <iostream>
using namespace std;
class A {
    public:
    void display() {
        cout << "Class A" << std::endl; } };
class B {
    public:
    void display() {
        cout << "Class B" << std::endl; } };
```

Ambiguity Resolution in Inheritance

```
class C : public A, public B {  
    public:  
        void view() {  
            display(); } };  
  
int main() {  
    C c;  
    c.view();  
    return 0; }
```

OUTPUT

[Error] reference to 'display' is ambiguous

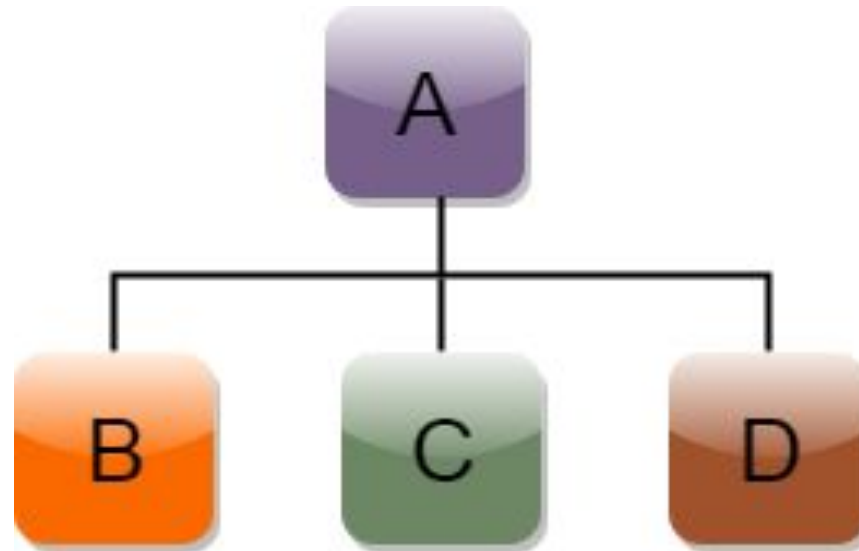
The above issue can be resolved by using the class resolution operator with the function. In the above example, the derived class code can be rewritten as:

```
class C : public A, public B
{
    public: void view()
    {
        A :: display();    // Calling the display() function of class A.
        B :: display();    // Calling the display() function of class B.

    }
};
```


Hierarchical Inheritance

- Hierarchical inheritance is defined as the process of deriving more than one class from a base class.



Syntax of Hierarchical inheritance:

```
class A
{
    // body of the class A.
}
class B : public A
{
    // body of class B.
}
class C : public A
{
    // body of class C.
}
class D : public A
{
    // body of class D.
}
```

Example

```
#include <iostream>
using namespace std;
class Shape          // Declaration of base class.
{   public:
    int a;
    int b;
    void get_data(int n,int m)  {
        a= n;
        b = m;  }
};
```

Example

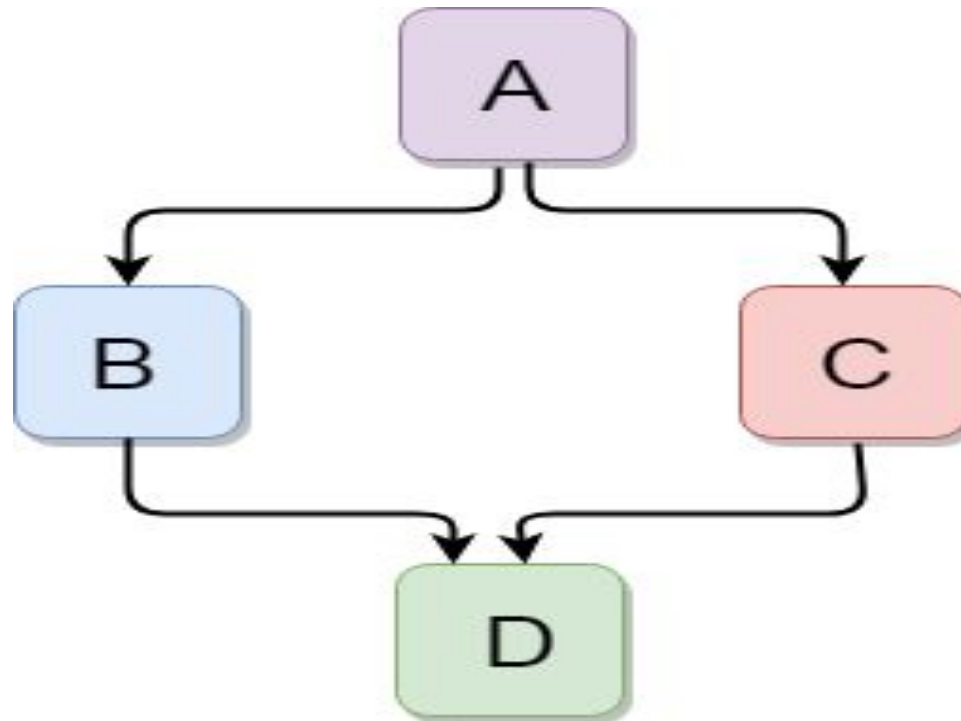
```
class Rectangle : public Shape{ // inheriting Shape class
public:
    int rect_area() {
        int result = a*b;
        return result; } };
class Triangle : public Shape { // inheriting Shape class
public:
    int triangle_area() {
        float result = 0.5*a*b;
        return result; } };
```

Example

```
int main() {  
    Rectangle r;  
    Triangle t;  
    int length,breadth,base,height;  
    cin>>length>>breadth;  
    r.get_data(length,breadth);  
    int m = r.rect_area();  
    cout << "Area of the rectangle is : " <<m<< std::endl;  
    cin>>base>>height;  
    t.get_data(base,height);  
    float n = t.triangle_area();  
    cout <<"Area of the triangle is : " << n<<std::endl;  
    return 0; }
```

C++ Hybrid Inheritance

- Hybrid inheritance is a combination of more than one type of inheritance.



Example

```
#include <iostream>
using namespace std;
class A {
    protected:
    int a;
    public:
    void get_a() {
        cout << "Enter the value of 'a' : " << std::endl;
        cin>>a; } };
•
```

Example

```
class B : public A {  
    protected:  
    int b;  
    public:  
    void get_b() {  
        cout << "Enter the value of 'b' : " << endl;  
        cin>>b; } };
```


Example

```
class C : public A {  
    protected:  
    int c;  
    public:  
    void get_c() {  
        cout << "Enter the value of c is : " << endl;  
        cin>>c; } };
```

Example

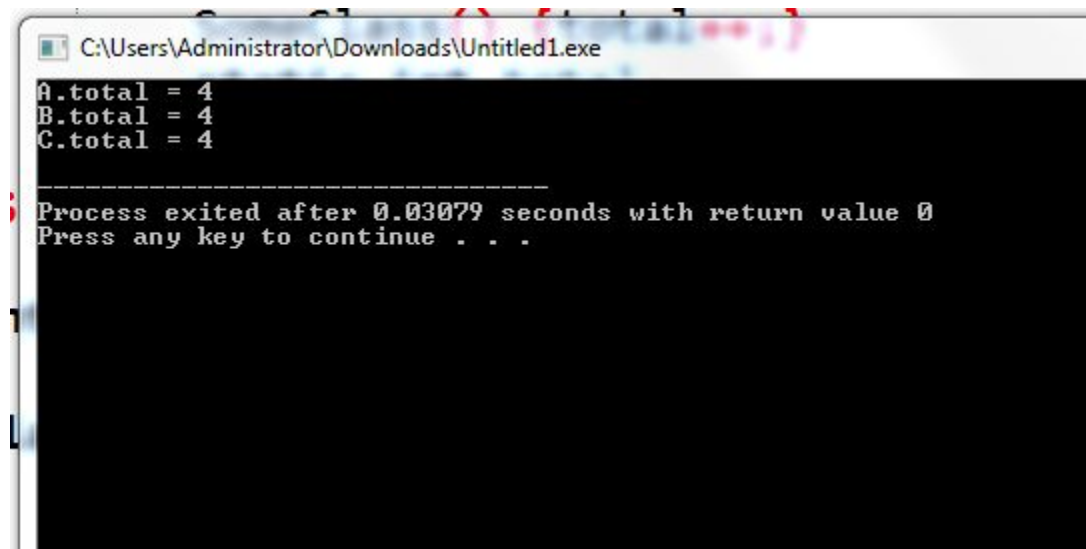
```
class D : public B, public C {  
    protected:  
    int d;  
    public:  
    void mul() {  
        get_a();  
        get_b();  
        get_c();  
        cout << "Multiplication of a,b,c is : " <<a*b*c<< endl; } };
```

Example

```
int main() {  
    D d;  
    d.mul();  
    return 0; }
```

When static members are inherited, are they static for the entire hierarchy, or just that class?

```
class SomeClass{
    public:
        SomeClass() {total++;}
        static int total;
        void Print(string n) { cout << n << ".total = " << total << endl; };
}
int SomeClass::total = 0;
class SomeDerivedClass: public SomeClass{
    public:
        SomeDerivedClass() {total++;};
}
int main(){
    SomeClass A;
    SomeClass B;
    SomeDerivedClass C;
    A.Print("A");
    B.Print("B");
    C.Print("C");
    return 0;}
```



A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Users\Administrator\Downloads\Untitled1.exe". The command prompt area has a black background with white text. The output of the program is as follows:

```
A.total = 4  
B.total = 4  
C.total = 4  
-----  
Process exited after 0.03079 seconds with return value 0  
Press any key to continue . . .
```

Task

- Create a class named **MusicalComposition** that contains fields for title, composer, and year written(attribute only accessible by its subclass). Include a setter function that requires all three values and an appropriate display function. The child class **NationalAnthem** contains an additional field that holds the name of the anthem's nation. The child class setter requires a value for this additional field. The child class also contains a display function. Write a main() function that instantiates object of subclass and call both display functions .

Solution

```
#include <iostream>
#include <String>
#include<ios> //used to get stream size
#include<limits> //used to get numeric limits
using namespace std;
class MusicComposition {
protected:
    string title,compose;
    int year;
public:
    void setA(){
        cout << "enter title"<< endl;
        getline(cin ,title);
        cout << "composer"<< endl;
        getline(cin ,compose);
        cout << "enter year"<< endl;
        cin >> year;
```

Solution

```
void display() {  
    cout << title << compose << year << endl; } };  
class NationalAnthem : public MusicComposition{  
public:  
    string name;  
    void setB(){  
        cout << "enter name"<< endl;  
        cin.ignore(); //clear buffer before taking new line  
        getline(cin ,name);  
  
    }  
    void display() {  
        cout << name << endl; } };
```


Solution

```
int main() {  
    NationalAnthem n;  
    n.setA() ;  
    n.setB();  
    n.MusicComposition::display();  
    n.display();  
    return 0; }
```

Constructors and Destructors in Derived Classes

- When we construct a derived class object, the base object must be created first. If we do not specify any base-constructor, it calls a default base-constructor. This is because the base-constructor does initialization of derived object's the inherited base-class member. The members of derived object are initialized by the derived-constructor.
- Invocation of constructors and destructors depends on the type of inheritance being implemented.

Constructors and Destructors in Derived Classes

- Instantiating a derived-class object begins a chain of constructor calls in which the derived-class constructor, before performing its own tasks, invokes its direct base class's constructor either explicitly (via a base-class member initializer) or implicitly (calling the base class's default constructor). Similarly, if the base class is derived from another class, the base-class constructor is required to invoke the constructor of the next class up in the hierarchy, and so on. The last constructor called in this chain is the constructor of the class at the base of the hierarchy, whose body actually finishes executing first. The original derived-class constructor's body finishes executing last. Each base-class constructor initializes the base-class data members that derived-class object inherits.

Constructor and destructor in single inheritance

```
#include <iostream>
using namespace std;
class A{
public:
    A() { cout << "A()" << endl; };
class B : public A{
public:
    B() { cout << "B()" << endl; }; // BY DEFAULT B() : A()
int main(){
    B b;
return 0;
}
```

Constructor and destructor in single inheritance

Remember, a derived-class constructor always calls a base-class constructor. The base-class object should be constructed before the code enters the body of the derived-class constructor.

Important points

- Whenever the derived class's default constructor is called, the base class's default constructor is called automatically.
- To call the parameterized constructor of base class inside the parameterized constructor of sub class, we have to mention it explicitly.
- The parameterized constructor of base class can be called in default constructor of sub class

Example

```
#include <iostream>
using namespace std;
class A{
protected:
    int ia;
public:
    A()
        { cout << "A()" << endl; };
class B : public A{
    int ib;
public:
    B(int n) : ib(n) { cout << "B()" << endl; };
int main(){
    B b(2);
return 0;}
```

Example

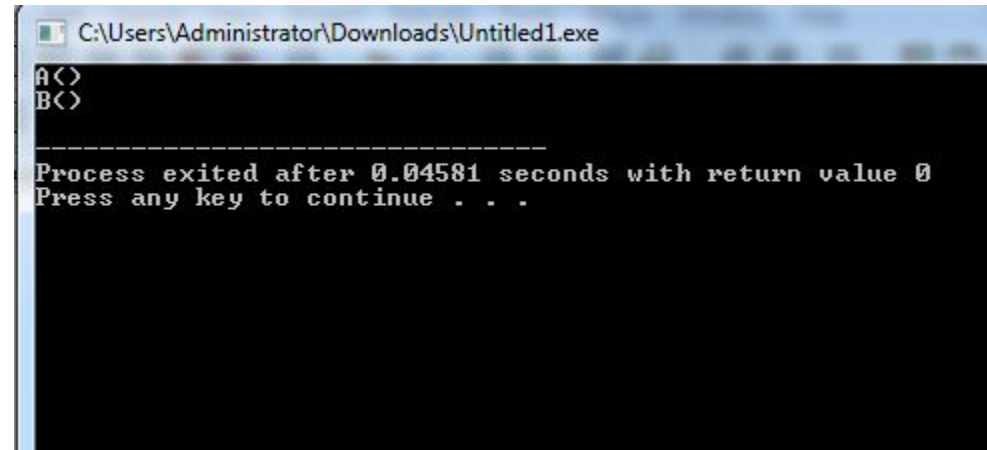
```
#include <iostream>
using namespace std;
class A{
protected:
    int ia;
public:
    A(int n) : ia(n) { cout << "A()" << endl; };
class B : public A{
    int ib;
public:
    B(int n) : ib(n) { cout << "B()" << endl; }; //[Error] no matching function for call to 'A::A()'
int main(){
    B b(2);
return 0;}
```


Example

```
#include <iostream>
using namespace std;
class A{
protected:
    int ia;
public:
    A(int n) : ia(n) { cout << "A()" << endl; };
class B : public A{
    int ib;
public:
    B(int n) : ib(n) { cout << "B()" << endl; }; // [Error] no matching function for call to 'A::A()'
int main(){
    A a (3);
    B b(2);
    return 0;}
```

The parameterized constructor of base class can be called in default constructor of sub class

```
#include <iostream>
using namespace std;
class A{
protected:
    int ia;
public:
    A(int n) : ia(n) { cout << "A()" << endl; };
class B : public A{
    int ib;
public:
    B() : A(3) { cout << "B()" << endl; };
int main(){
    B b;
return 0;}
```



```
C:\Users\Administrator\Downloads\Untitled1.exe
A<>
B<>
-----
Process exited after 0.04581 seconds with return value 0
Press any key to continue . . .
```

Parametrized Base Constructor

- If we need to initialize inherited the base-class member with different value from a default value, we can use base-class constructor in the initializer list of the derived-class constructor
- Derived class constructor(arg1,arg2) : base(arg1){
derived= arg2;} // order of arguments doesn't matter
- OR
- Derived class constructor(arg1,arg2) : base(arg1) , derived(arg2){}
- OR
- Derived class constructor(arg1) : base(any value) , derived(arg1){}

Parametrized Base Constructor

```
#include <iostream>
using namespace std;
class A{
public:
    A(int n = 1) : ia(n)
    { cout << "A() ia = " << ia << endl; }
protected:
    int ia;};
class B : public A{
    int ib;
public:
    B(int n , int a) : ib(n), A(a)
    { cout << "B()" << endl; } };
int main(){
    B b(2 ,4);
    return 0;}
```

Parametrized Base Constructor

```
A() ia = 4
```

```
B()
```

```
-----
```

```
Process exited after 0.1476 seconds with return value 0
```

```
Press any key to continue . . .
```

Case Study

Write a C++ program that has a class named “Course”.

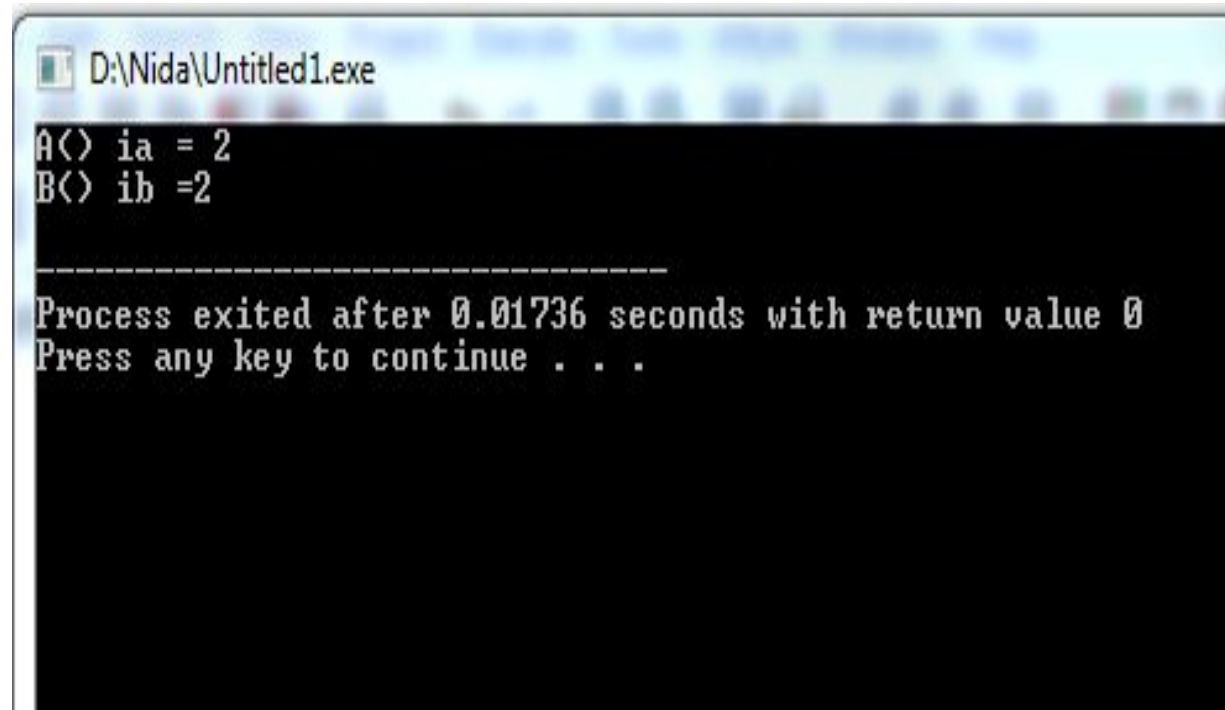
The class Course has the attributes course name, course code, class venue and credit hours, all are protected members.

1. Set all these attributes with a parameterized constructor.
2. Derive a class “OOP Course” that has an attribute teacher name.
3. Make a constructor and invoke the base class’s parameterized constructor.
4. Set the teacher name in the constructor.
5. The derived class has a function Display that displays all the details of the course and the derived class.
6. In the main, display all the details.

Parametrized Base Constructor

```
#include <iostream>
using namespace std;
class A{
public:
    A(int n) : ia(n)
    { cout << "A() ia = " << ia << endl; }
protected:
    int ia;};
class B : public A{
    int ib;
public:
    B(int n) : ib(n), A(n)
    { cout << "B() ib =" << ib << endl; } };
int main(){
    B b(2);
    return 0;}
```


Parametrized Base Constructor

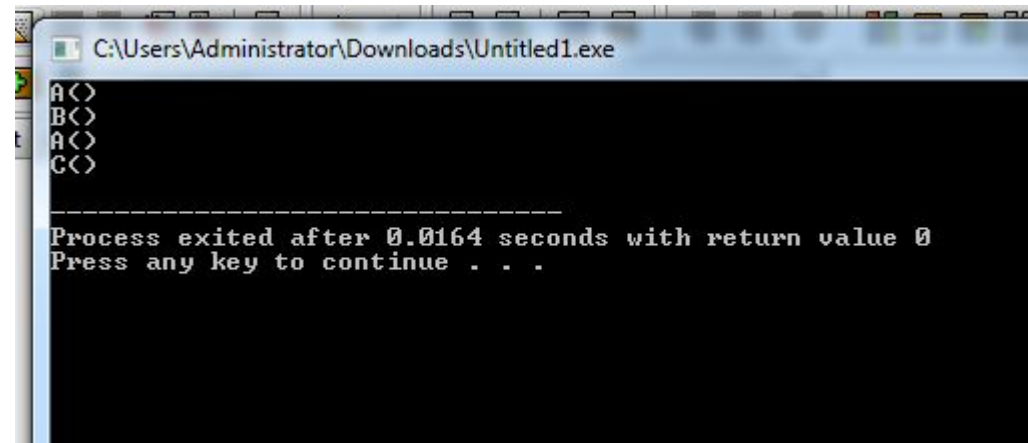


```
D:\Nida\Untitled1.exe
A() ia = 2
B() ib =2

-----
Process exited after 0.01736 seconds with return value 0
Press any key to continue . . .
```

Constructor and destructor in Hierarchical inheritance

```
#include <iostream>
using namespace std;
class A{
public:
    A() { cout << "A()" << endl; };
class B : public A{
public:
    B() { cout << "B()" << endl; } };
class C : public A{
public:
    C() { cout << "C()" << endl; } };
    int main(){
        B b;
        C c;
        return 0;}
```



```
C:\Users\Administrator\Downloads\Untitled1.exe
A(>)
B(>)
A(>)
C(>)
-----
Process exited after 0.0164 seconds with return value 0
Press any key to continue . . .
```

Constructor and destructor in multiple inheritance

- Constructors from all base class are invoked first and the derived class constructor is called.
- Order of constructor invocation depends on the order of how the base is inherited.
- For example:
- `class C : public A , public B`

Constructor and destructor in multiple inheritance

- Here, A is inherited first, so the constructor of class A is called first and then constructor of class B is called next.
- However, the destructor of derived class is called first and then destructor of the base class which is mentioned in the derived class declaration is called from last towards first in sequentially.

Syntax

- Derived class constructor(arg1,arg2,arg3) : base1(arg1) , base2(arg2){
derived= arg3;}

OR

- Derived class constructor(arg1,arg2,arg3) : base1(arg1) , base2(arg2),
derived(arg3){}

Constructor and destructor in multiple inheritance

```
class A{
public: A(int a) { cout << "A()" << a << endl; };
class B{
public: B(int b ){ cout << "B()" << b << endl; } };
class C :public A , public B{
public:
    int m;
    C(int x, int y, int z): A(x) , B(y) ,m(z){
        cout<< "C()" << m << endl;}};
    int main(){
        C c(2,4,6);
        return 0;}
```

Constructor and destructor in multiple inheritance

```
A()2  
B()4  
C()6
```

```
-----  
Process exited after 0.1935 seconds with return value 0  
Press any key to continue . . .
```


Constructor and destructor in multi level inheritance

```
class A{
public:
    A() { cout << "A()" << endl; };
class B : public A{
public:
    B() { cout << "B()" << endl; } };
class C : public B{
public:
    C() { cout << "C()" << endl; } };
int main(){
    C c;
    return 0;}
```

Constructor and destructor in multi level inheritance

```
A()  
B()  
C()
```

```
Process exited after 0.1995 seconds with return value 0  
Press any key to continue . . .
```

You can not directly call indirect parent(A) constructor in C

```
class A{
public:
    A(int a) { cout << "A()" << endl; };
class B : public A{
public:
    B(int b) :A(b){ cout << "B()" << endl; } };
class C : public B{
public:
    C():B(3) { cout << "C()" << endl; } };
int main(){
    C c;
    return 0;}
```

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

class A{
public:
    A(int a) { cout << "A()" << endl; };
class B : public A{
public:
    B(){ cout << "B()" << endl; } };
class C : public B{
public:
    C():A(3) { cout << "C()" << endl; } }; //error
int main(){
    C c;
    return 0;}
```

Task

- Create a class named A include a constructor that accepts one parameter(any type) , create another class named B that inherits from A also include a constructor that accepts one parameter(any type) , create a class named C that inherits from B having a constructor that accepts one parameter now initialize all base class constructors through the child classes
- **Remember** if the base class is derived from another class, the base-class constructor is required to invoke the constructor of the next class up in the hierarchy

Solution

```
#include <iostream>
using namespace std;
class A{
public:
    A(int a=1) { cout << "A()" << a << endl; };
class B : public A{
public:
    B(int b , int valA ):A(valA) { cout << "B()" << b << endl; } };
class C : public B{
public:
    C(int c ,int valB):B(valB ,6) { cout << "C()" <<c << endl; } };
    int main(){
        C c(2,4);
        return 0;}
```

Solution

```
#include <iostream>
using namespace std;
class A{
public:
    A(int a=1) { cout << "A()" << a << endl; };
class B : public A{
public:
    B(int b , int valA ):A(valA) { cout << "B()" << b << endl; } };
    class C : public B{
public:
        C(int c ,int valB, int ia):B(valB ,ia) { cout << "C()" <<c << endl; } };
        int main(){
            C c(2,4,6);
            return 0;}
```

Solution

```
A()6
```

```
B()4
```

```
C()2
```

```
-----
```

```
Process exited after 0.2411 seconds with return value 0
```

```
Press any key to continue . . .
```


Task

Create a class Number that contains the following attributes and methods.

1. -number
2. + getNumber()
3. + setNumber()

Derive two classes from class Number namely,

1. SquareOfNumber
2. CubeOfNumber

Write appropriate functions in the class Square and Cube to calculate the square and cube of any given number.

Solution

```
class Number {  
protected:  
    int number;  
  
public:  
    Number(int num) : number(num) {}  
  
    int getNumber() {  
        return number;  
    }  
  
    int returnNumber() {  
        return number;  
    }  
};
```

```
class SquareOfNumber : public Number {  
public:  
    SquareOfNumber(int num) : Number(num) {}  
  
    int calculateSquare() {  
        return number * number;  
    }  
};  
  
class CubeOfNumber : public Number {  
public:  
    CubeOfNumber(int num) : Number(num) {}  
  
    int calculateCube() {  
        return number * number * number;  
    }  
};
```

Case Study

Create a base class called shape. Use this class to store two double type values that could be used to compute the area of figures. Derive two specific classes called triangle and rectangle from the base shape. Add to the base class, a member function `get_data()` to initialize base class data members and another member function `display_area()` to compute and display the area of figures. Using these three classes, design a program that will accept dimensions of a triangle or a rectangle interactively and display the area. Remember the two values given as input will be treated as lengths of two sides in the case of rectangles and as base and height in the case of triangles and used as follows: Area of rectangle = $x * y$ Area of triangle = $\frac{1}{2} * x * y$

Case Study

Consider a class BankAccount that has:

Two attributes i.e. accountID and balance

A function named balanceInquiry() to get information about the current amount in the account.

Derive two classes from the BankAccount class i.e. CurrentAccount and SavingAccount. Both classes inherit all the attributes and behaviors from the BankAccount class. In addition, followings are required to be the part of both classes:

Appropriate constructors to initialize data fields of base class

A function named amountWithdraw(amount) to withdraw certain amount while taken into account the following conditions

While withdrawing from current account, the minimum balance should not decrease Rs. 5000

While withdrawing from saving account, the minimum balance should not decrease Rs. 10,000.

AmountDeposit(amount) to deposit amount in the account.

In main () function, create instance of derived class (i.e. CurrentAccount and SavingAccounts) and invoke their respective functions to test their working.

Case Study

Consider a base class named Employee and its derived classes HourlyEmployee and PermanentEmployee while taking into account the following criteria.

Employee class has two data fields i.e. name (type of string) and empID (of type integer).

Both classes (HourlyEmployee and PermanentEmployee) have an attribute named hourlyIncome.

Both classes (HourlyEmployee and PermanentEmployee) have three argument constructor to initialize the hourlyIncome as well as data fields of the base class.

Class HourlyEmployee has a function named calculate_hourly_income to calculate the income of an employee for the actual number of hours he or she worked. One-hour income is Rs.150.

Similarly, PermanentEmployee class has the function named calculate_income to calculate the income of an employee that gets paid the salary for exact 240 hours, no matter how many actual hours he or she worked. Again, one hour is Rs 150.

Implement all class definition with their respective constructors to initialize all data members

and functions to compute the total income of an employee. In the main() function, create an

instance of both classes (i.e. HourlyEmployee and PermanentEmployee) and test the working

of functions that calculate the total income of an employee.

Case Study

We want to calculate the total marks of each student of a class in Physics, Chemistry and Mathematics and the average marks of the class. The number of students in the class are entered by the user. Create a class named Marks with private data members 1. roll number 2. name 3.. marks Create three other classes inheriting the Marks class, namely 1. Physics 2. Chemistry 3. Mathematics which are used to define marks in individual subject of each student. Roll number of each student will be generated automatically.

Case study

- Create an Investment class that contains fields to hold the initial value of an investment, the current value, the profit (calculated as the difference between current value and initial value), and the percent profit (the profit divided by the initial value). Include a constructor that requires initial and current values and a display function. Create a House class that includes fields for street address and square feet, a constructor that requires values for both fields, and a display function. Create a HouseThatIsAnInvestment class that inherits from Investment and House. It includes a constructor and a display function that calls the display functions of the parents. Write a main() function that declares a HouseThatIsAnInvestment and displays its values.

Home Task

Write a C++ program that has a class named “Person”.

The class has a default constructor that displays “I am a person”.

The class has attributes name, age, nationality, address and CNIC.

The class has an input function that prompts the user to enter all the details. For CNIC, the total number of digits should be exactly 13. If it’s less than 13 or greater display an error message.

The class also has a display function that displays all the details.

Derive a class Employee from Person.

The class Employee has a default constructor that invokes the base class’s constructor and displays “I am an Employee”.

The class has the attributes name of company, company’s location (city), no of years worked.

The class has an input function that prompts the user to enter all the details. It also has a

display function that displays all the details.

Derive a class Manager from Employee.

The class Manager has a default constructor that invokes the base class’s constructor and

displays “I am a Manager”.

The class has an array that contains the names of employee’s who are working under the

manager’s supervision. Input at least five employee’s in the array from the user and display all

these employee’s too.

In the main program, call all the functions and display the details.