AMBITION GURU

Object-Oriented programming in C++ (CSIT 202)

II semester, BScCSIT

Compiled by Ankit Bhattarai
Ambition Guru College

Syllabus

AMBITION GURU

Unit	Contents	Hours	Remarks
1.	Introduction to C++ and OOP	3	
2.	Classes and objects	7	
3.	Operator Overloading & Type Conversion	7	
4.	**Inheritance**	**7**	
5.	Polymorphism and Virtual Functions	7	
6.	Templates and Exception Handling	7	
7.	I/O Stream	8	

Practical Works

Credit hours : 3

Unit 4
Inheritance
(7 hrs.)

Base class and Derived class, Concept and types of inheritance, Constructors and destructors in inheritance, Ambiguity and virtual base classes, Object slicing.

Introduction to inheritance

C++ supports the concept of reusability.

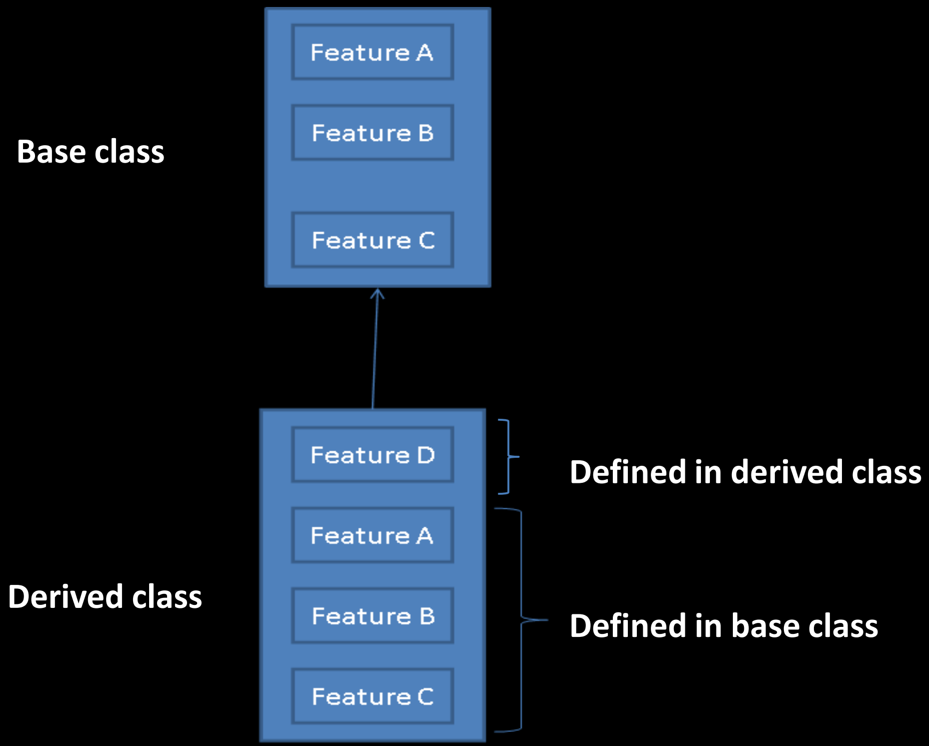
C++ classes can be reused repeatedly by the programmers to suit their requirements.

This can be achieved by creating new classes from the existing one. ***The process of creating new class from the existing one is called inheritance.***

Base and Derived class

The existing class is called an ***ancestor class*** or ***base class*** or ***parent class*** or ***superclass****.*

And the new class created from the existing one is called a ***descendant class*** or ***derived class*** or ***child class*** or ***sub class***.


Base class
Feature A
Feature B
Feature C
Derived class
Feature D
Feature A
Feature B
Feature C
Defined in derived class
Defined in base class

AMBITION GURU

C++ example showing base and derived class concepts of Vehicle → Car relationship

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

// Base class
class Vehicle {
protected:
    string brand;
    int year;

public:
    Vehicle(string b, int y) {
        brand = b;
        year = y;
    }

    void showInfo() {
        cout << "Brand: " << brand << ", Year: " << year << endl;
    }
};
```

```
// Derived class
class Car : public Vehicle {
private:
    string model;

public:
    Car(string b, int y, string m) : Vehicle(b, y) {
        model = m;
    }

    void showCarInfo() {
        showInfo();
        cout << "Model: " << model << endl;
    }
};
```

```
int main() {
    Car myCar("Toyota", 2022, "Corolla");
    myCar.showCarInfo();

    return 0;
}
```

Output:

```
Brand: Toyota, Year: 2022
Model: Corolla
```

Access Specifiers

AMBITION GURU

- **C++ access specifiers** are used for determining or setting the boundary for the availability of class members (data members and member functions) beyond that class.
- Access modifiers (or access specifiers) are keywords in object-oriented languages that set the accessibility of classes, methods, and other members.
- Access modifiers are a specific part of programming language syntax used to facilitate the encapsulation of components.
- Following are the access specifiers available in C++
  - ➢ private access specifier
  - ➢ protected access specifier
  - ➢ public access specifier

- By default, functions and data declared within a class are private to that class and may be accessed only by other members of the class.
- The **public** access specifier allows functions or data to be accessible to other parts of our program.
- The **protected** access specifier is needed only when inheritance is involved

Protected access specifier and its role in inheritance

The private and protected access specifiers has same effect in a class however their difference is seen in case of inheritance only.

The ***protected*** members are ***visible*** to ***derived class*** however ***private*** members ***are not***.

The purpose of making data member (s) as protected in a class is to make such members accessible to derived class functions.

Syntax of derived class declaration:

```
class Derived_class_name : visibility mode Base_class_name
{
        //Body of derived class
}
```

- The colon(:) after the Derived_class_name indicates that the new class is created from the existing one.
- Here visibility mode specifies either public or private or protected.

Visibility mode and its effect in inheritance

- Only the public and protected members of a class gets inherited in a derived class while private members of a class are not inherited.

AMBITION GURU

Visibility Mode	Inheritable public members of a baseclass in a derived class becomes	Inheritable protected membersof a base class in a derived class becomes	Private members ofa base class are not inherited in a derived class
public	public	protected	
protected	protected	protected	
private	**private**	**private**	

public, protected and private inheritance

public, protected, and private inheritance have the following features:

- *public inheritance* makes public members of the base class *public* in the *derived class*, and the *protected members* of the base class remain protected in the *derived class*.
- *protected inheritance* makes the *public* and *protected members* of the *base class protected in the derived class*.
- *private inheritance* makes the *public and protected* members of the *base class private in the derived class*.

```cpp
class Base {
  public:
    int x;
  protected:
    int y;
  private:
    int z;
};

class PublicDerived: public Base {
  // x is public
  // y is protected
  // z is not accessible from
PublicDerived
};
```

```cpp
class ProtectedDerived: protected Base {
  // x is protected
  // y is protected
// z is not accessible from ProtectedDerived
};

class PrivateDerived: private Base {
  // x is private
  // y is private
  // z is not accessible from PrivateDerived
};
```

// C++ program to demonstrate the working of public inheritance

AMBITION GURU

```
#include <iostream>
using namespace std;
class Base {
  private:
    int pvt = 1;
  protected:
    int prot = 2;
  public:
    int pub = 3;
    // function to access private member
    int getPVT() {
      return pvt;
    }
};
```

```
class PublicDerived : public Base {
  public:
    // function to access protected member from Base
    int getProt() {
      return prot;
    }
};


int main() {
  PublicDerived object1;
  cout << "Private = " << object1.getPVT() << endl;
  cout << "Protected = " << object1.getProt() << endl;
  cout << "Public = " << object1.pub << endl;
  return 0;
}
```

Output:

```
Private = 1
Protected = 2
Public = 3
```

AMBITION GURU

// C++ program to demonstrate the working of protected inheritance

```
#include <iostream>
using namespace std;
class Base {
  private:
    int pvt = 1;
  protected:
    int prot = 2;
   public:
    int pub = 3;
    // function to access private member
    int getPVT() {
      return pvt;
    }
};
```

```
class ProtectedDerived : protected Base {
  public:
    // function to access protected member from Base
    int getProt() {
      return prot;
    }

    // function to access public member from Base
    int getPub() {
      return pub;
    }
};
```

AMBITION GURU

```
int main() {
  ProtectedDerived object1;
  cout << "Private cannot be accessed." << endl;
  cout << "Protected = " << object1.getProt() << endl;
  cout << "Public = " << object1.getPub() << endl;
  return 0;
}
```

Output:

```
Private cannot be accessed.
Protected = 2
Public = 3
```

// C++ program to demonstrate the working of private inheritance

```
#include <iostream>
using namespace std;
class Base {
  private:
    int pvt = 1;
  protected:
    int prot = 2;
  public:
    int pub = 3;
    // function to access private member
    int getPVT() {
      return pvt;
    }
};
```

```
class PrivateDerived : private Base {
  public:
    // function to access protected member from Base
    int getProt() {
      return prot;
    }
    // function to access private member
    int getPub() {
      return pub;
    }
};
```

```
int main() {
  PrivateDerived object1;
  cout << "Private cannot be accessed." << endl;
  cout << "Protected = " << object1.getProt() << endl;
  cout << "Public = " << object1.getPub() << endl;
  return 0;
}
```

Output:

```
Private cannot be accessed.
Protected = 2
Public = 3
```

AMBITION GURU

Data Member Overriding

Data member overriding

- If the base class and derived class have the same name of the *data member(s)* then it is called ***data member overriding***
- The data member(s) of the base class are hidden in the derived class i.e., data member of derived class overrides (hides or displaces) the data member(s) of the base class.

Member function overriding

- If the base class and derived class have the same name of the member function, then it is called ***member function overriding***
- The member function of the base class is hidden in the derived class i.e. member function of derived class overrides(hides or displaces) the member function of the base class.

```
#include<iostream>
using namespace std;
class Base{
  protected:
    int n;
  public:
    void input(){
     cout<<"Enter the value to the n of a base class:"<<endl;
     cin>>n;
}
void display(){
     cout<<"The value to the base of n="<<n<<endl;
}
};
```

```
class Derived:public Base{
   protected:
     int n;
   public:
      void input( ){
     cout<<"Enter the value to the n of a derived class:"<<endl;
     cin>>n;
}
void display( )
{
     cout<<"The value to the n of a derived class is:"<<n<<endl;
     cout<<"The value of both the n is :"<<n+n<<endl;
}
};
```

```
int main( )
{
     Derived d;
     d.input(); //with the intention of calling Base input()
     d.display( ); //with the intention of calling Base display()
     d.input( ); //with the intention of calling Derived input()
     d.display( ); //with the intention of calling Derived display()
     return 0;
}
```

Output:

```
Enter the value to the n of a derived class:
3
The value to the n of a derived class is: 3
The value of both the n is :6
Enter the value to the n of a derived class:
2
The value to the n of a derived class is: 2
The value of both the n is :4
```

Overridden members of a base class can be invoked by two ways:

Overridden members of a base class can be invoked by twoways:

1. By the help of a member function of a derived class.
2. By the help of a derived class object.

In both the cases class_name scope resolution(::) and the name ofthe member is written.

By the help of a derived class function:

```
#include<iostream>
using namespace std;
class Base
{
protected:
int n;
public:
```

```
void input( )
{
     cout<<"Enter the value to the n of a base class:"<<endl;
     cin>>n;
}

void display( )
{
     cout<<"The value to the n of a base class="<<n<<endl;
}
};

class Derived:public Base
{
protected:
int n;
public:
```

```
void input( )
{
 Base::input( );
 cout<<"Enter the value to the n of a derived class:"<<endl;
 cin>>n;
}

void display( )
{
 Base::display( );
 cout<<"The value to the n of derived class is:"<<n<<endl;
 cout<<"The value of both the n is:"<<n+Base::n;
}};
```

```
int main()
{
    Derived d;
    d.input();
    d.display();
    return 0;
}
```

Output:

```
Enter the value to the n of a base class:
3
Enter the value to the n of a derived class:
5
The value to the n of a base class=3
The value to the n of derived class is:5
The value of both the n is:8
```

By the help of a derived class object:

```
#include<iostream>
using namespace std;
class Base{
protected:
int n;
public:
void input( ){
cout<<"Enter the value to the n of a base class:"<<endl;
cin>>n;
}
void display( ){
cout<<"The value to the n of a base class="<<n<<endl;
}
};
```

```
class Derived:public Base{
protected:
int n;
public:
void input( )
{
    cout<<"Enter the value to the n of a derived class:"<<endl;
    cin>>n;
}

void display( )
{
    cout<<"The value of n of a derived class is:"<<n<<endl;
    cout<<"The value of both the n is:"<<n+Base::n;
}};
```

```
int main( )
{
    Derived d;
    d.Base::input( );
    d.Base::display( );
    d.input( );  //invokes derived class input( );
    d.display( ); //invokes derived class display( );
    return 0;
}
```

Output:

```
Enter the value to the n of a base class:
3
The value to the n of a base class=3
Enter the value to the n of a derived class:
2
The value of n of a derived class is:2
The value of both the n is:5
```

Types of inheritance

Types of inheritance

1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance

AMBITION GURU

1. Single Inheritance

The type of inheritance in which there is a ***single base class, and a single derived*** class is called single inheritance.

The general form(syntax):

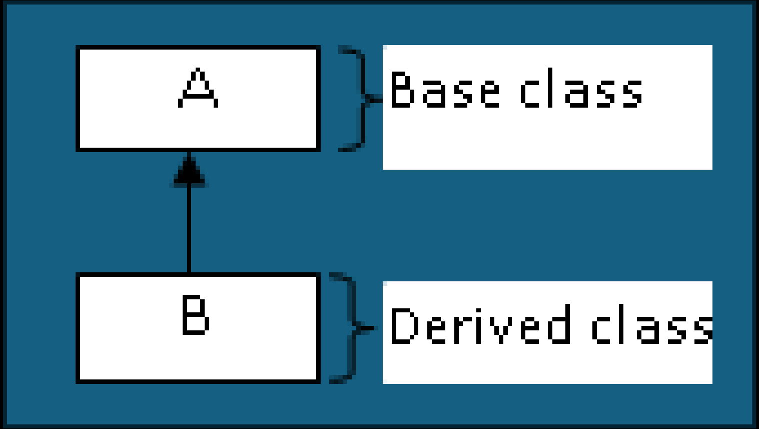
```
class base
{……….. . };

class derived: visibility_mode base
{……….. . };
```

For Example:

```
class A
{…};

class B:public A
{…};
```



//Example Program:

```
#include<iostream>
using namespace std;
class employee{
protected:
    char name[20];
    float salary;
public:
void input( )
{
    cout<<"Enter the name of an employee:"<<endl;
    cin>>name;
    cout<<"Enter the salary of an employee:"<<endl;
    cin>>salary;
}
```

```
void display( ){
     cout<<"Name of an employee="<<name<<endl;
     cout<<"Salary of an employee="<<salary<<endl;
}};

class manager:public employee
{
protected:
     char title[20];
public:
void input( )
{
     employee::input( );
     cout<<"Enter the title of an manager:"<<endl;
     cin>>title;
}
```

```
void display( ){
    employee::display( );
    cout<<"The title of a manager="<<title<<endl;
}};

int main( )
{
    manager m1,m2;
    m1.input( );
    m2.input( );
    m1.display( );
    m2.display( );
    return 0;
}
```

Output:

```
Enter the name of an employee:
Mark
Enter the salary of an employee:
560
Enter the title of an manager:
Engineer
Enter the name of an employee:
Samson
Enter the salary of an employee:
450
Enter the title of an manager:
Supervisor
```

```
Name of an employee = Mark
Salary of an employee = 560
The title of a manager = Engineer
Name of an employee = Samson
Salary of an employee = 450
The title of a manager = Supervisor
```

2. Multiple Inheritance:

The type of inheritance in which there are multiple(more than one) base classes, and a single derived class is called multiple inheritance.

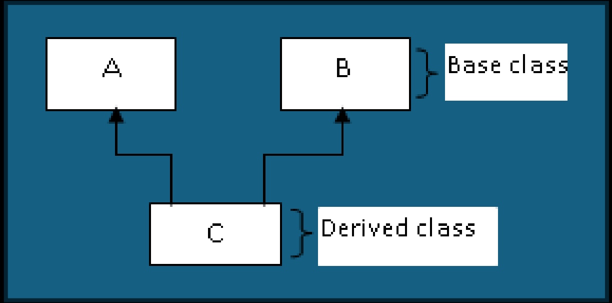
- The syntax of multiple inheritance is :

```
class base1
{………..};
class base2
{………..};

.

.
class basen
{…………};

class derived:visibility_mode base1,visibility_mode base2,..
{………….};
```



AMBITION GURU

AMBITION GURU

For Example:

```
class A
{…};

class B
{…};

class C:public A, public B
{…};
```

//Example program:

AMBITION GURU

```
#include<iostream>
using namespace std;
class Base1
{
protected:
int x;
public:

void get_x( )
{
    cout<<"Enter the value to x of Base1:"<<endl;
    cin>>x;
}

void display_x( )
{
    cout<<"The value to x of Base1="<<x<<endl;
}
};
```

```
class Base2
{
    protected:
    int y;
    public:

void get_y( )
{
    cout<<"Enter the value to y of a Base2:"<<endl;
    cin>>y;
}

void display_y( )
{
    cout<<"The value to the y of a Base2="<<y<<endl;
}
};
```

AMBITION GURU

```
class derived: public Base1,public Base2
{
protected:
int z;

public:
void get_z( )
{
     cout<<"Enter the value to z of a derived class:"<<endl;
     cin>>z;
}

void display_z( )
{
     cout<<"The value to the z of a derived class="<<z<<endl;
}

void display( )
{
     cout<<"The sum is:"<<x+y+z<<endl;
}
};
```

AMBITION GURU

```
int main( )
{
    derived d;
    d.get_x( );
    d.get_y( );
    d.get_z( );
    d.display_x( );
    d.display_y( );
    d.display_z( );
    d.display( );
    return 0;
}
```

Output:

```
Enter the value to x of Base1:
4
Enter the value to y of a Base2:
5
Enter the value to z of a derived class:
6
The value to x of Base1=4
The value to the y of a Base2=5
The value to the z of a derived class=6
The sum is:15
```

Ambiguity and its resolution in multiple inheritance:

In multiple inheritance there are multiple base classes and a single derived class.

If multiple base classes have same members name which are being inherited to derived class and when we access those members by using the object of a derived class, then the compiler becomes confused to use which version of member and of which base class and this is called the ambiguity in multiple inheritance.

In order to resolve this problem while invoking the same members name which are in more than one base classes, we use class name and scope resolution operator while invoking those members.

```
#include<iostream>
using namespace std;
class Base1
{
    protected:
    int x;
    public:
    void get_x( )
    {
    cout<<"Enter the value to x of a Base1:"<<endl;
    cin>>x;
    }

    void display_x( )
    {
    cout<<"The value to x of  Base1="<<x<<endl;
    }
};
```

```
class Base2
{
    protected:
    int x;
    public:
    void get_x( )
    {
        cout<<"Enter the value to x of a Base2:"<<endl;
        cin>>x;
    }

    void display_x( )
    {
        cout<<"The value to x of a Base2="<<x<<endl;
    }
};
```

```
class derived:public Base1,public Base2
{
protected:
int z;
public:
void get_z( )
{
     cout<<"Enter the value to z of a derived class:"<<endl;
     cin>>z;
}

void display_z( )
{
     cout<<"The value to the z of a derived class="<<z<<endl;
}

void display( )
{
     cout<<"The sum is:"<<Base1::x+Base2::x+z<<endl;
}
};
```

```
int main( )
{
derived d;
//d.get_x( );error
d.Base1::get_x( );
//d.display_x( );error
d.Base1::display_x( );

d.Base2::get_x( );
d.Base2::display_x( );
d.get_z( );
d.display_z( );
d.display( );
return 0;
}
```

Output:

```
Enter the value to x of Base1:
4
Enter the value to x of a Base2:
5
Enter the value to z of a derived class:
6
The value to x of Base1=4
The value to the x of a Base2=5
The value to the z of a derived class=6
The sum is:15
```

3. Multilevel Inheritance:

The type of inheritance in which derived class is formed from already derived class is called multilevel inheritance. The transitive nature of inheritance is shown by multilevel inheritance.

3. Multilevel Inheritance:

The type of inheritance in which derived class is formed from already derived class is called multilevel inheritance. The transitive nature of inheritance is shown by multilevel inheritance.

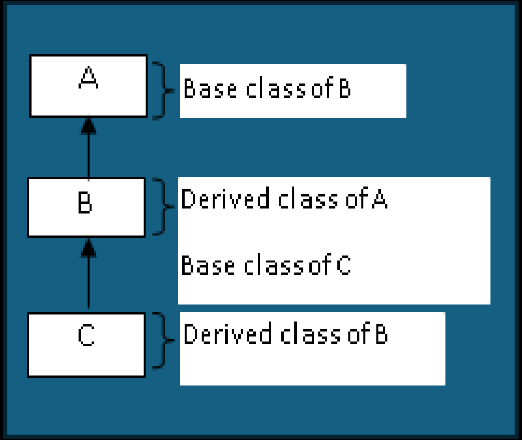
AMBITION GURU

General form(syntax):

```
class base
{……..};

class derived1:visibility_mode base
{………..};

class derived2:visibility_mode derived1
{………….};
```



AMBITION GURU

For Example:

```
class A
{…};

class B:public A
{…};

class C:public B
{…};
```

//Example Program

AMBITION GURU

```
#include<iostream>
using namespace std;
class student
{
protected:
     char name[20];
     int roll;
public:
void input( )
{
     cout<<"Enter the name of an student:"<<endl;
     cin>>name;
     cout<<"Enter the roll number of an student:"<<endl;
     cin>>roll;
}
```

AMBITION GURU

```
void display( )
{
     cout<<"The name of an student="<<name<<endl;
     cout<<"The roll number of an student="<<roll<<endl;
}
};

class test:public student
{
protected:
     float sub1,sub2;
public:
void input( )
{
     student::input( );
     cout<<"Enter the marks in subject1 and subject2:"<<endl;
     cin>>sub1>>sub2;
}
```

```
void display( )
{
     student::display( );
     cout<<"The marks of subject1="<<sub1<<endl;
     cout<<"The marks of subject2="<<sub2<<endl;
}
};

class result: public test
{
protected:
     float total; public:
void input( )
{
     test::input( );
     total=sub1+sub2;
}
```

```
void display( )
{
     test::display( );
     cout<<"Total="<<total<<endl;
}
};

int main( )
{
     result r;
     r.input( );
     r.display( );
     return 0;
}
```

Output:

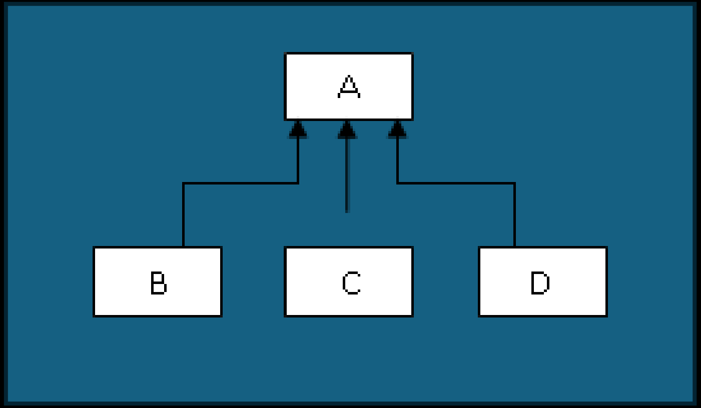
```
Enter the name of an student:
Ramesh
Enter the roll number of an student:
34
Enter the marks in subject1 and subject2:
3
4
The name of an student=Ramesh
The roll number of an student=34
The marks of subject1=3
The marks of subject2=4
Total=7
```

4. Hierarchical Inheritance

The type of inheritance in which there is a single base class, and multiple derived classes is called hierarchical inheritance.

The general form is:

```
class base
{…………};
class derived1 : visibility_mode base
{      …. .     };
class derived2 : visibility_mode base
{      ….       };
class derived3 : visibility_mode base
{      ….       };
.
.
class derivedn:visibility_mode base
{      ….       };
```



AMBITION GURU

For Example:

```
class A
{…};

class B : public A
{…};

class C : public A
{…};

class D : public A
{…};
```

//Example Program:

AMBITION GURU

```
#include<iostream>
using namespace std;
class employee{
protected:
    char name[20];
    float salary;
public:
void input( ){
    cout<<"Enter the name of an employee:"<<endl;
    cin>>name;
    cout<<"Enter the salary of an employee:"<<endl;
    cin>>salary; }

void display( ){
    cout<<"Name of an employee="<<name<<endl;
    cout<<"Salary of an employee="<<salary<<endl;
}
};
```

```
class manager: public employee
{
protected:
    char title[20];
public:
void input( )
{
    employee::input( );
    cout<<"Enter the title of a manager:"<<endl;
    cin>>title;
}
void display( )
{
    employee::display( );
    cout<<"The title of a manager="<<title<<endl;
}
};
```

AMBITION GURU

```
class teacher:public employee
{
protected:
    char faculty[20];
public:
void input( )
{
    employee::input( );
    cout<<"Enter the faculty of the teacher:"<<endl;
cin>>faculty;
}
void display( )
{
    employee::display( );
    cout<<"The faculty of the teacher="<<faculty;
}
};
```

```
int main( )
{
     cout<<"Manager:"<<endl;
     manager m;
     m.input( );
     cout<<"Teacher:"<<endl;
     teacher t;
     t.input( );
     cout<<"Manager details:"<<endl;
     m.display( );
     cout<<"Teacher details:"<<endl;
     t.display( );
     return 0;
}
```

Output:

```
Manager:
Enter the name of an employee:
Vicky
Enter the salary of an employee:
450
Enter the title of a manager:
Supervisor
Teacher:
Enter the name of an employee:
Kaushal
Enter the salary of an employee:
560
Enter the faculty of the teacher:
CSIT
```

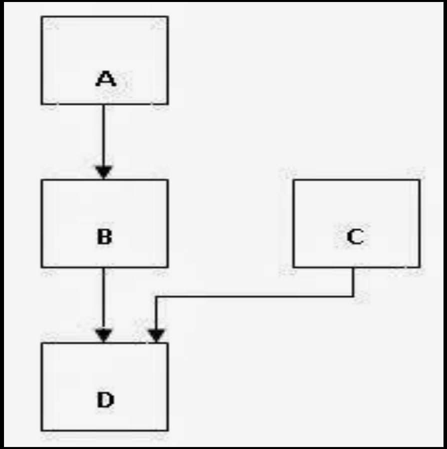
```
Manager details:
Name of an employee=Vicky
Salary of an employee=450
The title of a manager=Supervisor
Teacher details:
Name of an employee=Kaushal
Salary of an employee=560
The faculty of the teacher=CSIT
```

Hybrid Inheritance:

The type of inheritance in which there are more than one inheritances mixed together.

General form (syntax):

```
class base1
{…………..};
class derived1:visibility_mode base1
{…………..};
class base2
{……………};
class derived2:visibility_mode- derived1,visibility_mode base2
{……………};
```



AMBITION GURU

Hybrid Inheritance

Example:

```
class A{…};

class B:public A

{…};

class C{…};

class D: public B, public C

{…};
```

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

class student
{
protected:
     char name[20];
     int roll;
public:
void input( )
{
     cout<<"Enter the name of the student:"<<endl;
     cin>>name;
     cout<<"Enter the roll number of the student:"<<endl;
     cin>>roll;
}
```

```
void display( )
{
     cout<<"Name of the student="<<name<<endl;
     cout<<"Roll number of the student="<<roll<<endl;
}
};

class test: public student
{
protected:
     float sub1,sub2;
public:
void input( )
{
     student::input( );
     cout<<"Enter the marks in subject1 and subject2:"<<endl;
     cin>>sub1>>sub2;
}
```

```
void display( )
{
     student::display( );
     cout<<"The marks of subject1="<<sub1<<endl;
     cout<<"The marks of subject2="<<sub2<<endl;
}
};

class sports
{
protected:
     float sm;
public:
void input( )
{
     cout<<"Enter the marks of sports:"<<endl;
     cin>>sm;
}
```

```
void display( )
{
     cout<<"The marks of the sports="<<sm<<endl;
}
};

class result: public test, public sports
{
protected:
     float total;
public:
void input( )
{
     test::input( );
     sports::input( );
     total=sub1+sub2+sm;
}
```

```
void display( )
{
     test::display( );
     sports::display( );
     cout<<"Total marks="<<total<<endl;
}
};

int main( )
{
     result r;
     r.input( );
     r.display( );
      return 0;
}
```

Output:

```
Enter the name of the student:
Arjun
Enter the roll number of the student:
34
Enter the marks in subject1 and subject2:
23
45
Enter the marks of sports:
50
Name of the student=Arjun
Roll number of the student=34
The marks of subject1=23
The marks of subject2=45
The marks of the sports=50
Total marks=118
```

Constructors and destructors in inheritance

Constructor in derived class:

- If there is no constructor at all or there is a default constructor in a base class then it is not necessary to have a constructor in a derived class.
- However if there is a parameterized constructor in a base class then it is mandatory to have constructor in a derived class too because it is the job of derived class constructor to pass value(s) to the base class constructor.

```
//Example program:
#include<iostream>
using namespace std;
class alpha
{
protected:
    int x;
public:
alpha(int i)
{
 x=i;
 cout<<"alpha initialized"<<endl;
}
void display_x( )
{
 cout<<"x="<<x<<endl;
}
};
```

```
class beta
{
protected:
    int y;
public:
beta( int j)
{
    y=j;
    cout<<"beta initialized"<<endl;
}
void display_y( )
{
     cout<<"y="<<y<<endl;
}};
class gamma: public alpha, public beta
{
protected:
     int m,n;
public:
gamma(int a,int b,int c,int d):alpha(a),beta(b)
{
     m=c;
     n=d;
     cout<<"gamma initialized"<<endl;
}};
```

```
void display_mn( )
{
    cout<<"m="<<m<<endl;
    cout<<"n="<<n<<endl;
}
};
int main( )
{
gamma g(11,12,13,14);
g.display_x( );
g.display_y( );
g.display_mn( );
return 0;
}
```

Output:

```
alpha initialized
beta initialized
gamma initialized
x=11
y=12
m=13
n=14
```

AMBITION GURU

Constructor and destructor invocation order in case of single inheritance

Constructor and destructor invocation order in case of single inheritance

AMBITION GURU

- If there are constructors in base class and derived class in single inheritance, then first the constructor of base class is invoked and after that the constructor of derived class is invoked.
- The destructors invocation order is the reverse order of invocation of constructor.

```
#include<iostream>
using namespace std;
class Base
{
public:
Base( )
{
cout<<"Base class constructor"<<endl;
}
~Base( )
{
    cout<<"Base class destructor"<<endl;
}
};

class Derived:public Base
{
public:
Derived( )
{
    cout<<"Derived class constructor"<<endl;
}
~Derived( )
{
    cout<<"Derived class destructor"<<endl;
}};
```

```
int main( )
{
{
Derived d;
}
    cout<<"End of main"<<endl;
return 0;
}
```

Output:

```
Base class constructor
Derived class constructor
Derived class destructor
Base class destructor
End of main
```

AMBITION GURU

Constructor and destructor invocation order in case of multiple inheritance

Constructor and destructor invocation order in case of multiple inheritance

- The constructor's invocation in multiple inheritance take places according to the order of the derived class declaration.
- The destructors invocation order is reverse to the order of invocation of the constructor.

```
#include<iostream>
using namespace std;
class Base1
{
public:
Base1( )
{
    cout<<"Constructor from Base1:"<<endl;
}
~Base1( )
{
    cout<<"Destructor from Base1:"<<endl;
}};

class Base2
{
public:
Base2( )
{
    cout<<"Constructor from Base2:"<<endl;
}
~Base2( )
{
    cout<<"Destructor from Base2:"<<endl;
}};
```

```
class Derived:public Base1,public Base2
{
public:
Derived( )
{
    cout<<"Constructor from Derived:"<<endl;
}
~Derived( )
{
    cout<<"Destructor from Derived:"<<endl;
}
};
```

```
int main( )
{
{
Derived d;
}
    cout<<"End of main:"<<endl;
return 0;
}
```

Output:

```
Constructor from Base1:
Constructor from Base2:
Constructor from Derived:
Destructor from Derived:
Destructor from Base2:
Destructor from Base1:
End of main:
```

AMBITION GURU

Constructor and destructor invocation order in case of multilevel inheritance

Constructor and destructor invocation order in case of multilevel inheritance

- The order of constructor's invocation takes place in case of multilevel inheritance according to the order of inheritance.
- Destructors invocation takes place according to the reverse order of the invocation of the constructor.

```
//Example program
#include<iostream>
using namespace std;
class Base
{
public:
Base( ){
    cout<<"Base class constructor:"<<endl;
}
```

```
~Base( )
{
    cout<<"Base class destructor:"<<endl;
}
};

class derived1:public Base
{
public:
derived1( )
{
    cout<<"derived1 class constructor:"<<endl;
}
```

```
~derived1( )
{
    cout<<"derived1 class destructor:"<<endl;
}
};

class derived2:public derived1
{
public:
derived2( )
{
    cout<<"derived2 class constructor:"<<endl;
}
```

```
~derived2( )
{
    cout<<"derived2 class destructor:"<<endl;
}
};

int main( )
{
{
derived2 d;
}
    cout<<"End of main:"<<endl;
return 0;
}
```

Output:

```
Base class constructor:
derived1 class constructor:
derived2 class constructor:
derived2 class destructor:
derived1 class destructor:
Base class destructor:
End of main:
```

AMBITION GURU

Aggregation

Aggregation

AMBITION GURU

- Aggregation is called a “has a” relationship. We say a library has a book or an invoice has an item line.
- Aggregation is also called a “part-whole” relationship: the book is part of the library.
- In object-oriented programming, aggregation may occur when one object is an attribute of another. Here’s a case where an object of class A is an attribute of class B:

Example:

```
class A{
};

class B{
A objA; // define objA as an object of class B
};
```

```
#include <iostream>
#include <string>
using namespace std;
class Book {
    string title, author;
    public:
    Book()
    {
        title="";
        author="";
    }
Book(string t, string a)
    {
        title=t;
        author=a;
    }
    void display()
    {
        cout << "Title: " << title ;
        cout<< ", Author: " << author << "\n";
    }
};
```

```
// Class representing a Library
class Library
{
public:
    string name;
    Book book1;
    Book book2;
    Library(string n, Book b1, Book b2)
    {
        name=n;
        book1=b1;
        book2=b2;
    }

    void display()
    {
        cout << "Library: " << name << endl;
        book1.display();
        book2.display();
    }
};
```

```
int main()
{
    // Creating Book objects
    Book book1("The Great Gatsby", "F. Scott Fitzgerald");
    Book book2("1984", "George Orwell");
    // Creating a Library object and associating it with the Book objects
    Library library("City Library", book1, book2);
    // Displaying information about the library and its books
    library.display();
    return 0;
}
```

Output:

```
Library: City Library
Title: The Great Gatsby, Author: F. Scott Fitzgerald
Title: 1984, Author: George Orwell
```

AMBITION GURU

Virtual base classes

Virtual base classes

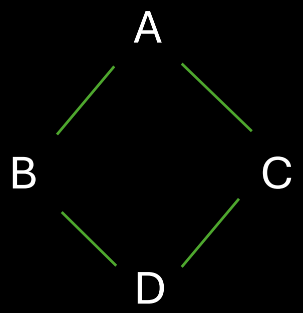
Purpose:
Solves the **Diamond Problem** in multiple inheritance, where the same base class is inherited more than once.

Diagram — Diamond Problem:

$$\begin{tikzcd}
 & A \arrow[dl, dash] \arrow[dr, dash] & \\
B \arrow[dr, dash] & & C \arrow[dl, dash] \\
 & D &
\end{tikzcd}$$

Without virtual inheritance, D gets two copies of A’s members → ambiguity.

Virtual base classes



Example:

```
class A {
public:
    int x;
};

class B : public A {};
class C : public A {};
class D : public B, public C {};

int main() {
    D obj;
    // obj.x = 5; // Ambiguous: which A::x?
}
```

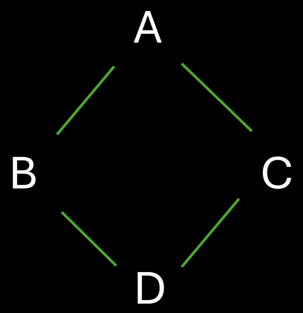
Virtual Base Class

- A **virtual base class** in C++ is a special kind of base class in multiple inheritance that ensures *only one shared copy* of the base class is inherited, even if it appears multiple times in the inheritance hierarchy.
- In other words, A virtual base class is a base class declared with the virtual keyword during inheritance to prevent multiple "copies" of the base class from being created in the derived class, thereby solving the diamond problem and avoiding ambiguity.

What does it do ?

- Prevents duplication of the base class in complex hierarchies.
- Declared like this: class B : virtual public A {};
- All derived classes that inherit from the same virtual base share the same instance of it.
- Used mainly to avoid ambiguity in multiple inheritance.

Virtual base classes



Solution:

```
class A {
public:
    int x;
};

class B : virtual public A {};
class C : virtual public A {};
class D : public B, public C {};

int main() {
    D obj;
    obj.x = 5; // Only one A exists
}
```

AMBITION GURU

Object Slicing

Object Slicing

When a derived class object is assigned to a base class object, the **extra data** (members of the derived part) gets “sliced off” — only the base class part remains.

Effects of Object Slicing

- Data loss: Derived members are lost.
- Loss of polymorphism: Overridden functions in derived class won’t be called.

Object Slicing Example:

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

class Base {
public:
    int a;
    void show()
      {
      cout << "Base: " << a << endl;
      }
};

class Derived : public Base {
public:
    int b;
    void show()
      {
      cout << "Derived: " << a << ", " << b << endl;
      }
};
```

Object Slicing

AMBITION GURU

```
int main() {
    Derived d;
    d.a = 1;
    d.b = 2;

    Base b = d;   // Slicing: b has only 'a'
    b.show();    // Prints: Base: 1
}
```

Questions

AMBITION GURU

1. Explain the practical implication of protected specifier in inheritance. list advantages and disadvantages of inheritance.
2. Describe the chain of constructors and destructors in inheritance.
3. What is aggregation? Write a program for implementing following: Create a class author with attributes name and qualification. Also create a class publication with pname. From these classes derive a classes derive a class book having attributes title and price. Each of the three classes should have getdata() method to get their data from user. The classes should have putdata() method to display the data. Create instance of the class book in main.
4. What are the various class access specifiers? How public inheritance differs from private inheritance?
5. How ambiguity occurs in multiple inheritances? Explain with an example how ambiguity can be resolved?

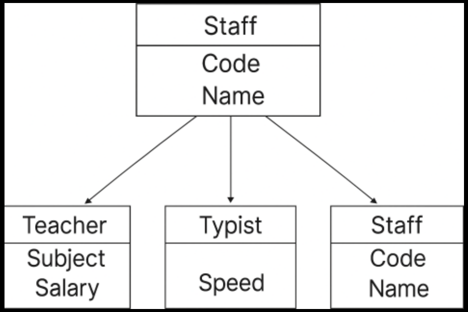
Questions

6. What do you mean by virtual base class ? At which condition it has to be implemented ? Explain it with suitable example.

7. Define derived class and base class. Explain any two types of inheritance in brief with example.

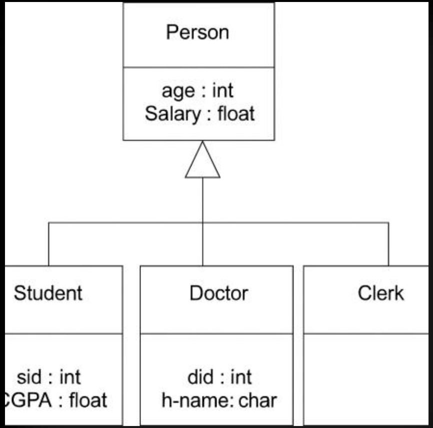
8. Depict the difference between private and public derivation. Explain derived class constructor with suitable program.

9. State the use of new operator. An educational institute wishes to maintain a data of its employee. The hierarchical relationships of related classes are as follows. Define all the classes to represent below hierarchy and define functions to retrieve individual information as and when required.



Questions

10. Write a program to realize the above hierarchy. Create necessary member functions (constructor) to initialize and display necessary information.



AMBITION GURU

THANK YOU

Any Queries ?