Object-Oriented Programming
Lab 05
Inheritance
Version 1.0

Reusability is yet another feature of OOP. C++ strongly supports the concept of reusability using inheritance. The mechanism of deriving a new class from an old one is called inheritance. With the help of inheritance it is possible to form new classes using already defined and more generalized ones. Inheritance helps to reuse existing code when defining several classes which have some common properties or methods.

There are various forms of inheritance. Such as-

1. Single

2. Multiple

3. Multilevel

4. Hierarchical

5. Hybrid etc.

Defining derive class:

class derive-class-name : visibility-mode **base-class-name**

{

// members of derive-class

};

The colon(:) indicates that the derive-class-name is derive from the base-class-name. The visibility-mode may be either private or public and by default it is private.

Example-

class X: public Y

{

members of X // public derivation

};

class X: private Y

{

members of X   // private derivation

};

Now let us see the visibility of inherited members in the following table:

Base class visibility	**Derive class visibility**		
	Public derivation	**Private derivation**	**protected derivation**
Private	Not inherited	Not inherited	Not inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance. The type of inheritance is specified by the access-specifier as explained above.

We hardly use protected or private inheritance, but public inheritance is commonly used. While using different type of inheritance, following rules are applied:

- **Public Inheritance:** When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.
- **Protected Inheritance:** When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.
- **Private Inheritance:** When deriving from a private base class, public and protected members of the base class become private members of the derived class

A question arise that what are the various functions that can have access to private and protected members of a class? They could be:

1. A function that is a friend of the class.
2. A member function of a class that is a friend of the class.
3. A member function of a derive class.

Single Inheritance:

Let us see a complete example of public derivation in case of single inheritance:

```
// Single inheritance with pubic derivation
#include<iostream>
using namespace std;
class B{
      int a;
public:
      int b;
      void getab();
      int geta();
      void showa();
};
class D: public B{
      int c;
public:
      void mul(void);
      void display(void);
};
void B:: getab(void)
{
      a=5;b=10;
}
int B:: geta()
{
      return a;
}
void B:: showa(){
      cout<<"a="<<a<<"\n";
}
void D:: mul()
{
      c=b*geta();
}
void D:: display()
{
      cout<<"a="<<geta()<<"\n";
      cout<<"b="<<b<<"\n";
      cout<<"c="<<c<<"\n\n";
```

```
//output
a=5
a=5
b=10
c=50


a=5
b=20
c=100
```

```
}
int main()
{
	D d;
	d.getab();
	d.mul();
	d.showa();
	d.display();

	d.b=20;
	d.mul();
	d.display();
	return 0;
}
```

The class D is a public derivation of the base class B. So, a public member of the base class B is also a public member of the derive class D.

A complete example of private derivation in case of single inheritance is shown below: ***(Analyze the following program yourself)***

```
// Single inheritance with private derivation

#include<iostream>
using namespace std;
class B{
	int a;
public:
	int b;
	void getab();
	int geta(void);
	void showa();
};
class D: private B{
	int c;
public:
	void mul(void);
	void display(void);
};
void B:: getab(void)
{
cout<<"Enter values for a and b:";
cin>>a>>b;

}
int B:: geta();
{
	return a;
}
void B:: showa(){
	cout<<"a="<<a<<"\n";
```

```
//output
Enter values
for a and b:5
10
a=5
b=10
c=50

Enter values
for a and b:12
20
a=12
b=20
c=240
```

```
}
void D:: mul()
{
	getab();
	c=b*geta();
}
void D:: display()
{
	showa();
	cout<<"b="<<b<<"\n";
	cout<<"c="<<c<<"\n\n";
}
int main()
{
	D d;
	//d.getab();won't work
	d.mul();
	//d.showa();won't work
	d.display();

	d.b=20; won't work, b has become private
	d.mul();
	d.display();
	return 0;
}
```

Multilevel Inheritance:

```
// Multilevel inheritance
#include<iostream>
using namespace std;

class student{
protected:
int roll_number;
public:
void getnumber(int);
void putnumber(void);
};

void student :: getnumber(int a)
{
roll_number=a;
}
void student :: putnumber()
{
cout<<"Roll number:"<<roll_number<<"\n";
}


class test : public student
{
protected:
float sub1;
float sub2;
public:
void getmark(float,float);
void putmark(void);
};

void test :: getmark(float x, float y)
{
sub1=x;
sub2=y;
}
void test :: putmark()
{
cout<<"Mark in sub1="<<sub1<<"\n";
cout<<"Mark in sub2="<<sub2<<"\n";
}

class result: public test
{
float total;
Pulic:
void display(void);
};

void result :: display()
{
total=sub1+sub2;
putnumber();
putmark();
cout<<" Total="<<total<<"\n";
```

```
//output
Roll number:123
Mark in sub1=75
Mark in
sub2=59.5
Total=134.5
```

```
}

int main()
{
result st1;
st1.getnumber(123);
st1.getmark(75.0,59.5);
st1.display();
return 0;

}
```

When class A serves as a base class for the derive class B, which in turn serves as a base class for the derive class for C then such type of inheritance is known as multilevel inheritance. The chain ABC is known as inheritance path.

Let us consider a simple example shown below:

In the above example, class student stores the roll-number, class test stores the marks obtained in two subjects and class result contains the total marks obtained in the test.

Multiple Inheritance:

A class can inherit the attributes of two or more classes is known as multiple inheritance. It is like a child inheriting the features of one parent and the intelligence of another.

The syntax of multiple inheritance as follow:

class D: visibility B1,visibility B2,…….

{

// body of D

};

The following program illustrating how three classes are implemented in multiple inheritance modes.

```
// Multiple inheritance with three classes

#include<iostream>
using namespace std;
class M{
protected:
      int m;
public:

      void getm(int);

};
class N{
protected:
      int n;
public:

      void getn(int);

};

class P :public M, public N
{
public:
void display(void);
};
void M :: getm(int x)
{
m=x;
}
void N :: getn(int y)
{
n=y;
}
void P :: display(void)
{
cout<<"m="<<m<<"\n";
cout<<"n="<<n<<"\n";
cout<<"m*n="<<m*n<<"\n";
}


int main()
{
      P d;
      d.getm(10);
      d.getn(20);
      d.display();
      return 0;
}
```

```
//output
m=10
n=20
m*n=120
```

Hybrid Inheritance:

There could be situations where we need to apply two or more types of inheritance to design a program. Such types of inheritances is known as hybrid inheritance. For example, hybridization of multiple,multilevel inheritance.

Let us consider a simple example shown below:

```
// Hybrid inheritance
#include<iostream>
using namespace std;

class student{
protected:
int roll_number;
public:
void getnumber(int a)
{
roll_number=a;
}

void putnumber()
{
cout<<”Roll number:”<<roll_number<<”\n”;
}
};


class test : public student
{
protected:
float sub1;
float sub2;
public:
void getmark(float x, float y)
{
sub1=x;
sub2=y;
}
void putmark()
{
cout<<”Mark in sub1=”<<sub1<<”\n”;
cout<<”Mark in sub2=”<<sub2<<”\n”;
}
};
class sports
{
protected:
float score;
public:
void getscore(float s)
{
score=s;
}
void putscore(void)
{
cout<<”Sports wt:”<<score<<”\n\n”;
```

```
//output
Roll number:123
Mark in sub1=75
Mark in
sub2=59.5
Sports wt:6.0
Total=140.5
```

```
}
};

class result: public test, public sports
{
float total;
public:
void display(void);
};

void result :: display()
{
total=sub1+sub2+score;
putnumber();
putmark();
putscore();
cout<<" Total="<<total<<"\n";
}

int main()
{
result st1;
st1.getnumber(123);
st1.getmark(75.0,59.5);
st1.getscore(6.0);
st1.display();
return 0;

}
```

In the above example, class student stores the roll-number, class test stores the marks obtained in two subjects, class sports contains the sports weight and class result contains the total marks obtained.

Home Task

1. Write a detailed note on Single Inheritance and Multilevel Inheritance, Multiple Inheritance, Hybrid Inheritance.
2. Create a generic base class **building** that stores the number of floors, number of rooms, and its total square footage. Create a derive class called **house** that inherits building and also stores number of bedrooms and bathrooms. Also, create a derive class called **office** that inherits house and also stores number of fire extinguishers and telephones. Now, write a program to display the contents stored in the office class.

3. The class **master** derives information from both **account** and **admin** classes which in turn derive information from the class **person**. Define all four classes and write a program to create, update and display the information contained in master objects.