Object-Oriented Programming
Lab 05
Type Conversions and Inheritance
Version 1.0

Before discussing the topic inheritance, we will first discuss about type conversions.

Type conversions

The type of data to the right side of an assignment operator is automatically converted to the type of the variable on the left. For example, the statements

```
int m;
float x=3.14159;
m=x;
```

Convert x to an integer before its value is assigned to m;

Three types of situations might arise in the data conversion between incompatible types:

1. Basic type to class type
2. Class type to basic type
3. Class type to another class type

1. Basic type to class type:

Let us considering an example of converting an int type to a class type:

```
class time
{
    int h,m;
public:
    time( int t)
    {
        h=t/60;
        m=t%60;
        cout<<h<<m;
    }
};

int main()
{
    int d=85;
    time t1=d;
    return 0;
}
```

Note- The constructors used for the type conversion take a single argument whose type is to be converted.

2. Class type to basic type:

The general format of the conversion function is:

```
operator typename()
{
        ……
        …..
        (Function statements)
        …..
}
```

This function converts a class object to typename.

Consider the following conversion function:

```
 vector :: operator double()
 {
     double magnitude,sum=0;
     for(int i=0; i<size; i++)
     {
         cin>>v[i];
         sum=sum+v[i]*v[i];
     }
     magnitude = sqrt(sum);
     return magnitude;
 }
```

The operator double() can be used as follows:

```
double magnitude=double(v1); // double magnitude=v1;
```

Where v1 is an object of type vector.

3. Class type to another class type (Self-study)

Hint: overload **operator = (assignment operator)**

Inheritance

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	X	X	X
Protected	Protected	Private	Protected
Public	Public	Private	Protected

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. **(Friend function)**
2. A member function of a class that is a friend of the class. **(Friend class)**
3. A member function of a **derive** class can access protected members of base 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 setab(void)
    {
        a=5;
        b=10;
    }
    int geta()
    {
        return a;
    }
    void showa()
    {
        cout<<"a="<<a<<"\n";
    }
};
class D: public B
{
    int c;
public:
    void mul()
    {
        c=b*geta();
    }
    void display()
    {
        cout<<"a="<<geta()<<"\n";
        cout<<"b="<<b<<"\n";
        cout<<"c="<<c<<"\n\n";
    }
};

int main()
{
    D d;
    d.setab();  // a=5, b=10
    d.mul();    // c=5*10
    d.showa();  // a=5
    d.display();

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

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


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

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";
}
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;
}
```

```
//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
```

Multilevel Inheritance:

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:

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

class student
{
protected:
    int roll_number;
public:
    void getroll(int);
    void showroll(void);
};

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


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

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

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

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

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

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

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.

Home Task

1. Write a detailed note on Single Inheritance and Multilevel 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.

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";
```

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

```
cout<<”m*n=”<<m*n<<”\n”;
}


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

Hybrid Inheritance:

There could be situations where we need to apply two or more types of inheritance to design a program. Such type of inheritance is known as hybrid inheritance. For example, hybridization of single and 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 getroll(int a)
    {
        roll_number=a;
    }

    void showroll()
    {
        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 showmark()
    {
        cout<<"Mark in sub1="<<sub1<<"\n";
        cout<<"Mark in sub2="<<sub2<<"\n";
```

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

```
    }
};
class sports
{
protected:
    float score;
public:
    void getscore(float s)
    {
        score=s;
    }
    void showscore(void)
    {
        cout<<"Sports wt: "<<score<<"\n\n";
    }
};

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

void result :: display()
{
    total=sub1+sub2+score;
    showroll();
    showmark();
    showscore();
    cout<<" Total="<<total<<"\n";
}

int main()
{
    result st1;
    st1.getroll(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 contain the sports weight and class result contains the total marks obtained.

Virtual Base Classes:

Consider a situation where all the three kinds of inheritance multilevel, multiple and hierarchical are involved.

Let us see the following code segment:

```
class A {
……      // grandparent
……
};
class B1: public A {
…….
……..   // parent1
};
class B2: public A {
…….
……..   // parent2
};
class C: public B1, public B2    // Child
{
……. //  ambiguity, which copy of A will be inherited ??
……..
};
```

Here the duplication of data members occurs due to these multiple paths. These duplication can be avoided by making common base class as *virtual base class*. The key word virtual is used in this purpose.

```
class A {
……      // grandparent
……
};
class B1: virtual public A{
…….
……..   // parent1
};
class B2: public virtual A{
…….
……..   // parent2
};
class C: public B1, public B2        // Child
{
……. // Only one copy of A will be inherited
……..
};
```

Note - The keywords virtual and public may be used as either order.

Home Task

1. Write a detailed description on Multiple Inheritance, Hybrid Inheritance and Virtual Base Classes.
2. 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.