**Experiment No: 2 Date:- 08-10-2020**

**AIM: To study the fundamentals of inheritance**

**THEORY:**

The capability of a class to derive properties and characteristics from another class is called **Inheritance**.

Inheritance is one of the most important features of Object Oriented Programming. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class.

● *Sub Class*: The class that inherits properties from another class is called Subclass or Derived Class.

● *Super Class*: The class whose properties are inherited by subclass is called Base Class or Super class.

**Implementing inheritance in C++:**

Syntax:

class subclass\_name : access\_mode base\_class\_name

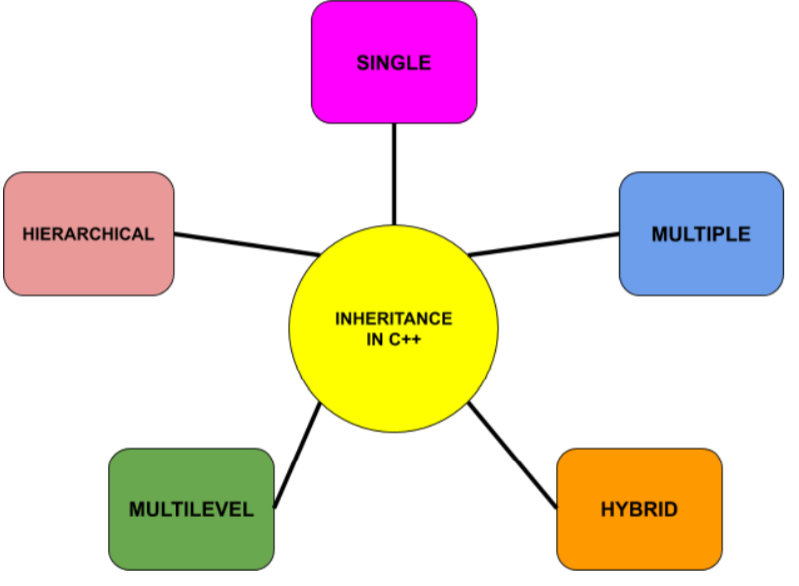
{

//body of subclass

};

In above syntax subclass\_name is the name of the sub class, access\_mode is the mode in which you want to inherit this sub class for example: public, private etc. and base\_class\_name is the name of the base class from which you want to inherit the sub class

**Types of Inheritance:**

****

1. **Single inheritance** is defined as the inheritance in which a derived class is inherited from the only one base class.

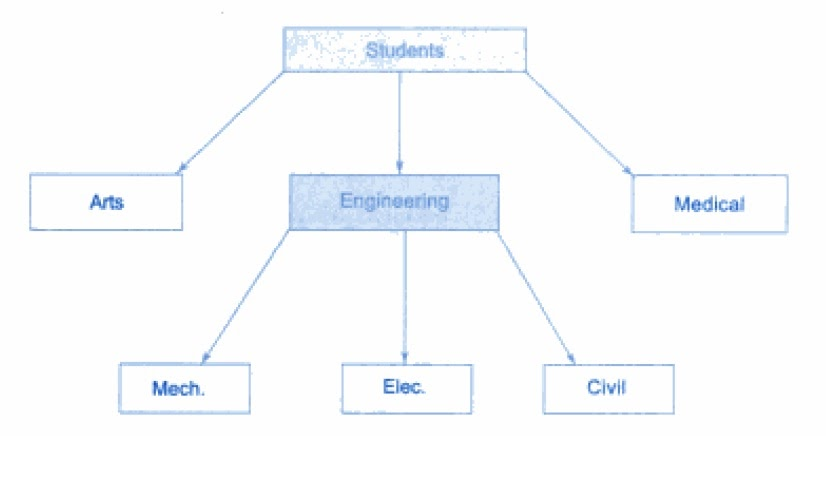
2. **Multilevel inheritance** is a process of deriving a class from another derived class.

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

4. **Hybrid inheritance** is a combination of more than one type of inheritance.

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

**E.g. of Hierarchical inheritance**



**Modes of Inheritance**

1. **Public mode:** If we derive a sub class 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.

2. **Protected mode:** If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in the derived class.

3. **Private mode:** If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in the derived class.

**Constructors in Derived Classes**

* As long as the base class constructor takes no arguments, the derived class constructor need not accept any arguments. However, if the base class has a parametrised constructor, then it is mandatory for the derived class to take arguments and pass them on to the base class.
* When both the derived class and the base class contain constructors, the base class constructor is executed first, followed by the execution of the derived class constructor.
* In case of multiple Inheritance, the base class constructors are called in the order in which they appear in the declaration of the derived class.
* In case of multi-level Inheritance, the constructors will be executed in the order of Inheritance.
* The header line of the derived class constructor function contains two parts separated by a colon. The first part provides the declaration of arguments that are passed to the derived constructor and the second part lists the function calls to the base constructors.

*General Form:*

Constructor(arg1, arg2, arg3…argN, argD):B1(arg1),B2(arg2)...BN(argn)

{

//constructor body

}

arg1, arg2 and so on upto argN are the arguments that the derived class constructor passes to the base constructors b1, b2 and so on upto bn. ArgD is the argument used by the derived class contructor.

**Virtual Base Class**

In the case of multiple Inheritance paths due to hybrid inheritance, the derived class may inherit the same members more than once, leading to duplication. This must be avoided by declaring the common base class as **virtual.**

When a class is made virtual, only one copy of that class is inherited regardless of the multiple inheritance paths between virtual base class and derived class.

The constructors for virtual base classes are called invoked before any non-virtual base class. If there are multiple virtual base classes, they are invoked in the order in which they are declared.

[A] Write a C++ program to study implementation of Hierarchical form of Inheritance

#include<iostream>

using namespace std;

class school

{

int n;

public:school(int x)

{

n=x;

}

protected:void shown()

{

cout<<n;

}

};

class student:public school

{

char name[30];

int age;

public:student(int n,char \*na,int a):school(n)

{

strcpy(name,na);

age=a;

show();

}

protected:void show()

{

cout<<"Your unique school id is:"<<endl;

int i=0;

while(i<=2)

cout<<name[i++];

cout<<age;

shown();

cout<<endl<<endl;

}

};

class teacher:public school

{

char name[30];

int age;

public:teacher(int n,char \*na,int a):school(n)

{

strcpy(name,na);

age=a;

show();

}

protected:void show()

{

cout<<"Your unique school id is:"<<endl;

int i=0;

while(i<=2)

cout<<name[i++];

cout<<age;

shown();

cout<<endl<<endl;

}

};

int main()

{

int age,code;

char name[30];

cout<<"Enter Faculty name:"<<endl;

cin>>name;

cout<<"Enter age:"<<endl;

cin>>age;

cout<<"Enter subject code (ranging from 2050 to 2059):"<<endl;

cin>>code;

teacher(code,name,age);

cout<<"Enter student name:"<<endl;

cin>>name;

cout<<"Enter age:"<<endl;

cin>>age;

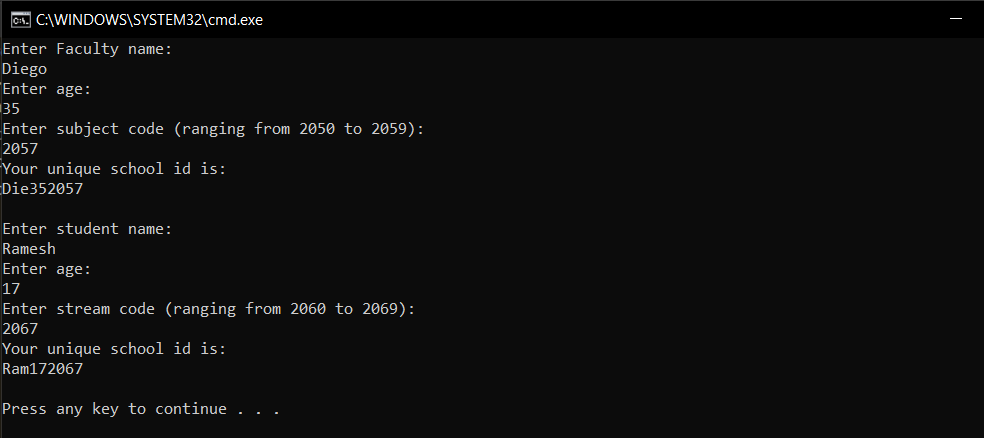
cout<<"Enter stream code (ranging from 2060 to 2069):"<<endl;

cin>>code;

student(code,name,age);

return 0;

}



[B] Write a C++ program to study implementation of Hybrid form of Inheritance(Virtual Base Class)

#include<iostream>

using namespace std;

class employee

{

int id;

protected:void getid(int x)

{

id=x;

}

public:void showid()

{

cout<<"Employee id:"<<id<<endl;

}

};

class time:virtual public employee

{

int hours;

protected: void geth(int h)

{

hours=h;

}

public: void showh()

{

cout<<"Hours worked last week:"<<hours<<endl;

}

int sendh()

{

return hours;

}

};

class project:virtual public employee

{

int proj;

protected: void getpro(int p)

{

proj=p;

}

public: void showpro()

{

cout<<"Total projects worked on in the week:"<<proj<<endl;

}

int sendp()

{

return proj;

}

};

class salary:public time, public project

{

int s;

public: void getsalary(int id,int p,int h)

{

getid(id);

getpro(p);

geth(h);

s=sendh()\*500+700\*sendp();

showid();

showh();

showpro();

shows();

}

void shows()

{

cout<<"Total salary earned in the respective week:"<<s<<endl;

}

};

int main()

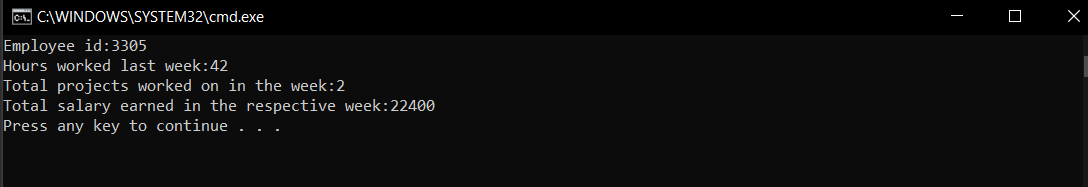
{

salary a;

a.getsalary(3305,2,42);

return 0;

}



[C] Write a C++ program to study the concept of constructors in derived classes

#include<iostream>

#include<iomanip>

using namespace std;

class hospital

{

char illness[30];

public: hospital(const char \*i)

{

strcpy(illness,i);

}

protected: void display1()

{

cout<<setw(46)<<"ILLNESS:"<<illness<<endl;

}

};

class patient

{

int age;

char name[30];

public: patient(int a, const char\*n)

{

strcpy(name,n);

age=a;

}

protected:void display2()

{

cout<<setw(50)<<"NAME:"<<name<<endl<<setw(52)<<"AGE:"<<age<<endl;

}

};

class person:public hospital,public patient

{

char dob[20];

public: person(const char \*i,int a,const char\*n,const char \*d):hospital(i),patient(a,n)

{

strcpy(dob,d);

display3();

}

protected:void display3()

{

display2();

display1();

cout<<setw(47)<<"DOB:"<<dob<<endl;

}

};

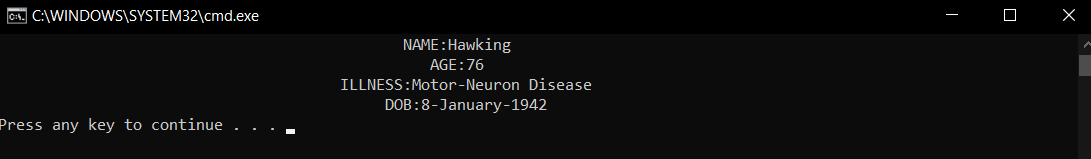
int main()

{

person a("Motor-Neuron Disease",76,"Hawking","8-January-1942");

return 0;

}



Conclusion: All programs were successfully run and executed with main emphasis on the C++ concept of Inheritance. The following was also observed.

1. Inheritance is one of the striking features of Object-Oriented Programming as it allows code reusability to a large extent.
2. In case of hybrid Inheritance, it is necessary to ensure that the respective base class is made virtual, to avoid duplication of the base class members.
3. An abstract class is only meant to act as base class and is not used to create objects.