Research paper on Inheritance and its type in Object Oriented Programming using C++

Dr. Pratik I Prajapati
I/c Principal (Assistant Professor)
St. Stephen Institute of Business Management & Technology, Anand.

Abstract:

Reusability is one of most important advantage of C++ programming language. C++ classes can be reused in several ways. Once the parent (Base) class has been written it can be modified by another programmer to suit their requirements. The main idea of inheritance is creating new classes, reusing the properties of the existing base class. The mechanism of deriving a new class (Child/Derived Class) from an Existing class (Base/Parent Class) is called inheritance.

The old class is referred to as the base (Parent) class and the new class is called the derived class (Child) or subclass. A derived class includes all features of the generic base class and then adds qualities specific to the derived class.

This paper reflects the study of the Inheritance concept and its types using C++ (oops).

Keywords:

Base (Parent) class, Reusability,– Sub (Derived/Child) Class, Visibility Modes and Types of Inheritance.

Introduction

Inheritance is the process by which objects of one class acquires the properties of objects of another class in the hierarchy.

The capability of a class to derive properties and characteristics from another class is called Inheritance**.** Inheritance is one of the most important feature of Object Oriented Programming.

A. Sub Class:

The class that inherits properties from another class is called Sub class or Derived class.

B. Super Class:

The class whose properties are inherited by sub class is called Base Class or Super class.

Sub classes can be created from the existing classes. It means that we can add additional features to a Base class without modifying it. The new class is referred as derived class or subclass and the original class is known as base classes or super class.

Review of Literature

Suvarnalata Hiremath & C M Tavade (May2016), According to Author "Review Paper on Inheritance and issues in Object Oriented Languages", The objective of this research paper is to review concept of inheritance in object oriented languages. The review paper begins upon the survey of inheritance and reusability of object oriented language. Inheritance plays an important role for code reusability. Since object oriented has been widely acclaimed as the technology that will support creation of reusable software, particularly because of the inheritance feature. Then discuss a new approach of inheritance mechanism, which overcomes the encapsulation issues and other issues derived from inheritance, which compromises severely reusability in object oriented language and also we explore the connection between inheritance and code reusability.

Shyamapriya Chowdhury & Sudip Chatterjee (February 2016). In this research paper "A Thorough Study of Different Types of Inheritance using Object Oriented Programming with JAVA" the authors demonstrate the concept of inheritance in our daily life. Creation of a new class from an existing one is called inheritance. Without modifying the previous data new features can be added in a class. The newly created class is called child class and from which it is created is known as parent class. Just like human being, child class can automatically access all the properties of Parent class and new methods can also be entered in child class. Concept of reusability is directly supported by JAVA using this inheritance.

Bjarne Stroustrup, he wrote research paper titled Multiple Inheritance for C++, In this research paper the author describe that Multiple Inheritance is the ability of a class to have more than one base class (super class). In a language where multiple inheritance is supported a program can be structured as a set of inheritance lattices instead of (just) as a set of inheritance trees. This is widely believed to be an important structuring tool. It is also widely believed that multiple inheritance complicates a programming language significantly, is hard to implement, and is expensive to run. I will demonstrate that none of these last three conjectures are true.

Generalized Syntax for inheritance

```
class baseclass
{
        visibility mode:
        data member declaration;
        member function declaration;
        .       .       .
        .       .       .
        .       .       .
};
class derivedclass : visibility mode baseclass name
{
        visibility mode:
                data member declaration;
                member function declaration;
        .        .        .
        .        .        .
        .       .      .
};
```

Visibility mode is used in the inheritance of C++ to show or relate how base classes are viewed with respect to derived class. When one class gets inherited from another, visibility mode is used to inherit all the public and protected members of the base class. Private members never get inherited and hence do not take part in visibility. By default, visibility mode remains "private".

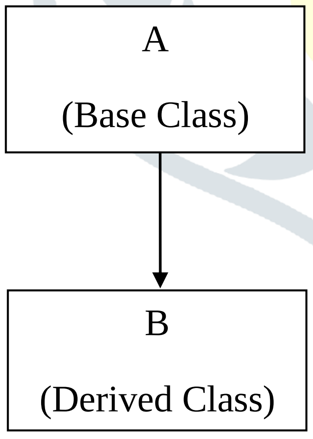
Types of Inheritance

1. Single Level Inheritance
2. Multiple Inheritance
3. Hierarchical inheritance
4. Multilevel Inheritance
5. Hybrid Inheritance.

1. Single Level Inheritance

If a derived class is created from only one base class then such a inheritance is called single level inheritance.

Consider a simple example of single inheritance.



(Chart No: 1 - Single Level Inheritance)

The above diagram shows single inheritance. Class A is Base class and class B is consider as a Derived class. Class B has all the properties of its own as well as Base Class (i.e. A)

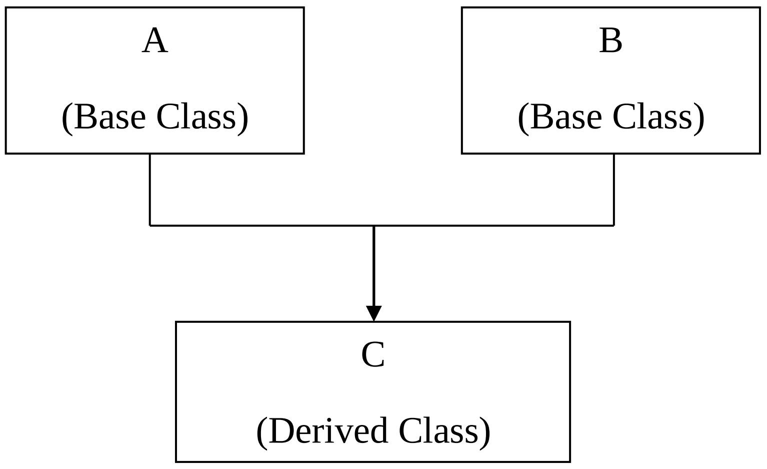
Example of single level inheritance

```
#include <iostream.h>
class A
{
	public:
		int y;
		void read()
		{
			cout<<"Enter Y : ";
			cin>>y;
		}
		void print()
		{
			cout<<"\n Y = "<<y;

		}

};

class B: public A

{
	int p;
	public:
		void read1()
		{
			read();
			cout<<"Enter P : ";
			cin>>p;
		}
		void print1()
		{
			print();
			cout<<"\n P = "<<p;
		}
};
void main()
{
	B b;
	clrscr();
	b.read1();
	b.print1();
	getch();
}
Output :
Enter Y : 12
Enter  P : 15
Y = 12
P = 15
```

2. Multiple Inheritance

If a derived class is created from more than one base class then such a inheritance is called multiple inheritance.

Consider a simple example of Multiple Inheritance.



(Char No : 2 - Multiple Inheritance)

The above diagram shows multiple inheritance. Class A is Base class and class B is also a base class and class C is created from two base class A & B. So class C is called Derived class. Class C has all the properties of its own as well as class A and class B.

Multiple inheritances allow us to merge the features of several base classes as a starting point for defining new classes. It is just like a child inherits the some properties of father and mother both.

Example of Multiple Inheritance

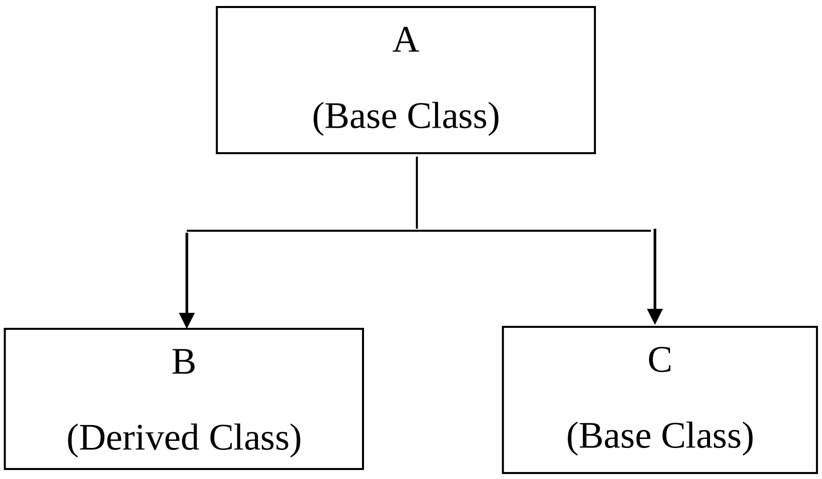
```
#include <iostream.h>
class A
{
	public:
		int x;
		void read()
		{
			cout<<"Enter X : ";
			cin>>x;
		}
		void print()
		{
			cout<<"\n X = "<<x;
		}
};
class B
{
	public:
		int y;
		void read1()
		{
			cout<<"Enter Y : ";
			cin>>y;
		}
		void print1()
		{
```

```
                        cout<<"\n Y = "<<y;
                }
};

class C: public A, public B
{
        int z;
        public:
                void read2()
                {
                        read();
                        read1();
                        cout<<"Enter Z : ";
                        cin>>z;
                }
                void print2()
                {
                        print();
                        print1();
                        cout<<"\n Z = "<<z;
                }
};
void main()
{
        C c;
        clrscr();
        c.read2();
        c.print2();
        getch();
}

Output :
Enter  X : 12
Enter  Y : 15
Enter  Z :  18
X = 12
Y = 15
Z  =  18
```

3. Hierarchical Inheritance

If more than one derived classes are created from same base class then such a inheritance is called Hierarchical inheritance.

Consider a simple example of Hierarchical Inheritance.



(Chart No: 3 - Hierarchical Inheritance)

The above diagram shows hierarchical inheritance. Class A is Base class and class B and class C are derived class. Class B has properties of its own and class A. Class C has all the properties of its own as well as class A.

Example of Hierarchical Inheritance

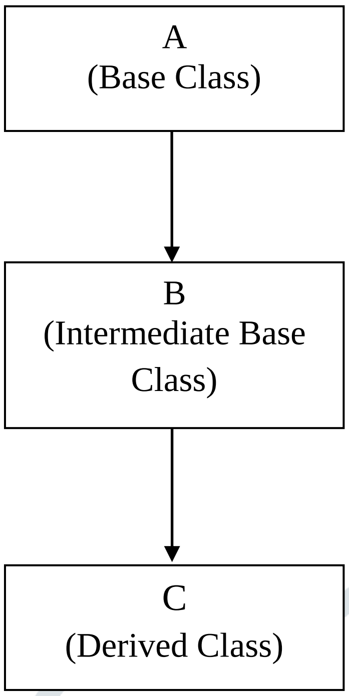
```
#include <iostream.h>
class A
{
        public:
                int x;
                void read()
                {
                        cout<<"Enter X : ";
                        cin>>x;
                }
                void print()
                {
                        cout<<"\n X = "<<x;
                }
};
class B:public A
{
        public:
                int y;
                void read1()
                {
                        read();
                        cout<<"Enter Y : ";
                        cin>>y;
                }
                void print1()
                {
                        print();
                        cout<<"\n Y = "<<y;
                }
};
```

```
class C: public A
{
        int z;
        public:
                void read2()
                {
                        read();
                        cout<<"Enter Z : ";
                        cin>>z;
                }
                void print2()
                {
                        print();
                        cout<<"\n Z = "<<z;
                }
};
void main()
{
        B b;
        C c;
        clrscr();
        b.read1();
        c.read2();
        b.print1();
        c.print2();
        getch();
}

Output :
Enter  X : 5
Enter  Y : 11

Enter  X :  20
Enter  Z :  25
X = 5
Y = 11

X = 20
Z  =  25
```

4. Multi Level Inheritance

If a derived class is created from another derived class (intermediate base class) then such a inheritance is called Multilevel inheritance.

Consider a simple example of Multi level Inheritance.



(Chart No: 4 - Multilevel Inheritance)

The above diagram shows multilevel inheritance. Class A is Base class and class B is created from class A. Class B has properties of its own and class A. Class C is created from class B(intermediate base class) so class C has the properties of its own as well as class A and class B.

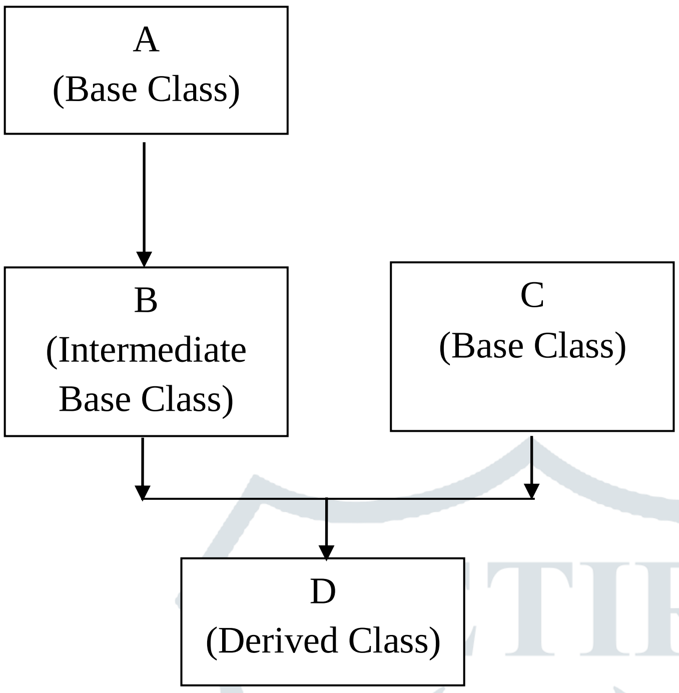
Example of Multilevel Inheritance

```
#include <iostream.h>
class A
{
        public:
                int x;
                void read()
                {
                        cout<<"Enter X : ";
                        cin>>x;
                }
                void print()
                {
                        cout<<"\n X = "<<x;
                }
};
class B:public A
{
        public:
                int y;
                void read1()
                {
                        read();
                        cout<<"Enter Y : ";
                        cin>>y;
                }
                void print1()
                {
                        print();
                        cout<<"\n Y = "<<y;
                }
};
class C: public B
{
        int z;
        public:

                void read2()
                {
                        read1();
                        cout<<"Enter Z : ";
                        cin>>z;
                }
                void print2()
                {
                        print1();
                        cout<<"\n Z = "<<z;
                }
};
void main()
{
        C c;
        clrscr();
        c.read2();
        c.print2();
        getch();
}

Output :
Enter  X : 10
Enter  Y : 20
Enter  Z :  30
X = 10
Y = 20
Z  =  30
```

5. Hybrid Inheritance

Combination of one or more inheritance is known as Hybrid Inheritance.

Consider a simple example of hybrid inheritance.



(Chart No: 5 - Hybrid Inheritance)

Accessibility in Public Inheritance

Accessibility	**private variables**	**protected variables**	**public variables**
Accessible from own class?	yes	yes	yes
Accessible from derived class?	no	yes	yes
Accessible from 2nd derived class?	no	yes	yes

Accessibility in Protected Inheritance

Accessibility	**private variables**	**protected variables**	**public variables**
Accessible from own class?	yes	yes	yes
Accessible from derived class?	no	yes	yes (inherited as protected variables)
Accessible from 2nd derived class?	no	yes	yes

Accessibility in Private Inheritance

Accessibility	**private variables**	**protected variables**	**public variables**
Accessible from own class?	yes	yes	yes
Accessible from derived class?	no	yes (inherited as private variables)	yes (inherited as private variables)
Accessible from 2nd derived class?	no	no	no

Conclusion

The mechanism (process) of deriving a new class from existing (old) class is called inheritance, by using inheritance we can reuse the features of existing class and that is the most important concept in C++. All the inheritance has its own features and its use to provide users to reusability concepts strongly, to save time and reduce the complexity.

In this paper we have to study the above five types of inheritance. We have to find that inheritance is central concepts in C++ that allows deriving a class from multiple classes at a time.

References:

1. b3e6ba44.pdf
2. Bjarne Stroustrup, The C++ Programming Language, 4th edition, Pearson Education Inc.
3. E Balagurusamy, Object oriented Programming with C++, 6th Edition, New Delhi: Tata McGraw-Hill Publishing Company Limited.
4. http://www.ijarcsms.com/docs/paper/volume1/issue2/V1I2-0005.pdf
5. https://pdfs.semanticscholar.org/a61d/a617de6cc75ae1bbbc0b02bea7ba
6. https://www.geeksforgeeks.org/inheritance-in-c/
7. https://www.programiz.com/cpp-programming/public-protected-private-inheritance
8. https://www.researchgate.net/profile/Bjarne_Stroustrup/publication/2396782
9. Multiple_Inheritance_for_C/links/00b7d514319dc8875a000000/Multiple-Inheritance-for-C.pdf