Fast**National University of Computer & Emerging Sciences, Karachi  
Spring 2020 CS-Department  
CS 217 – Object-oriented Programming**

**Inheritance in C++**

Inheritance is one of the feature of [Object Oriented Programming System(OOPs)](https://beginnersbook.com/2017/08/cpp-oops-concepts/), it allows the child class to acquire the properties (the data members) and functionality (the member functions) of parent class.

**What is child class?**  
A class that inherits another class is known as child class, it is also known as derived class or subclass.

**What is parent class?**  
The class that is being inherited by other class is known as parent class, super class or base class.

**Syntax of Inheritance**

class parent\_class

{

//Body of parent class

};

class child\_class : access\_modifier parent\_class

{

//Body of child class

};

## **Advantage of**[**C++ Inheritance**](https://www.javatpoint.com/cpp-inheritance)

**Code reusability:** Now you can reuse the members of your parent class. So, there is no need to define the member again. So less code is required in the class.

## **C++ Single Inheritance**

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

C++ Inheritance

Where 'A' is the base class, and 'B' is the derived class.

## **C++ Single Level Inheritance Example: Inheriting Fields**

When one class inherits another class, it is known as single level inheritance. Let's see the example of single level inheritance which inherits the fields only.

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Account {
4. **public**:
5. **float** salary = 60000;
6. };
7. **class** Programmer: **public** Account {
8. **public**:
9. **float** bonus = 5000;
10. };
11. **int** main(**void**) {
12. Programmer p1;
13. cout<<"Salary: "<<p1.salary<<endl;
14. cout<<"Bonus: "<<p1.bonus<<endl;
15. **return** 0;
16. }

**Output:**

Salary: 60000

Bonus: 5000

## **C++ Single Level Inheritance Example: Inheriting Methods**

Let's see another example of inheritance in C++ which inherits methods only.

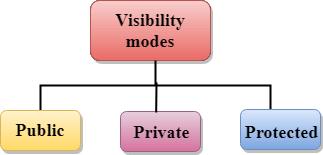
1. #include <iostream>
2. **using** **namespace** std;
3. **class** Animal {
4. **public**:
5. **void** eat() {
6. cout<<"Eating..."<<endl;
7. }
8. };
9. **class** Dog: **public** Animal
10. {
11. **public**:
12. **void** bark(){
13. cout<<"Barking...";
14. }
15. };
16. **int** main(**void**) {
17. Dog d1;
18. d1.eat();
19. d1.bark();
20. **return** 0;
21. }

**Output:**

Eating...

Barking...

**Visibility modes can be classified into three categories:**



* **Public**: When the member is declared as public, it is accessible to all the functions of the program.
* **Private**: When the member is declared as private, it is accessible within the class only.
* **Protected**: When the member is declared as protected, it is accessible within its own class as well as the class immediately derived from it.

## **Visibility of Inherited Members**

|  |  |  |  |
| --- | --- | --- | --- |
| **Base class visibility** | **Derived class visibility** | | |
| **Public** | **Private** | **Protected** |
| Private | Not Inherited | Not Inherited | Not Inherited |
| Protected | Protected | Private | Protected |
| Public | Public | Private | Protected |

## **C++ Multilevel Inheritance**

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

C++ Inheritance

## **C++ Multi Level Inheritance Example**

When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C++. Inheritance is transitive so the last derived class acquires all the members of all its base classes.

Let's see the example of multi level inheritance in C++.

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Animal {
4. **public**:
5. **void** eat() {
6. cout<<"Eating..."<<endl;
7. }
8. };
9. **class** Dog: **public** Animal
10. {
11. **public**:
12. **void** bark(){
13. cout<<"Barking..."<<endl;
14. }
15. };
16. **class** BabyDog: **public** Dog
17. {
18. **public**:
19. **void** weep() {
20. cout<<"Weeping...";
21. }
22. };
23. **int** main(**void**) {
24. BabyDog d1;
25. d1.eat();
26. d1.bark();
27. d1.weep();
28. **return** 0;
29. }

**Output:**

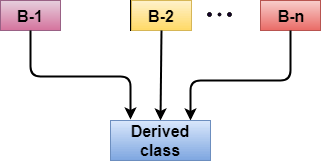
Eating...

Barking...

Weeping...

## **C++ Multiple Inheritance**

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



**Syntax of the Derived class:**

1. **class** D : visibility B-1, visibility B-2, ?
2. {
3. // Body of the class;
4. }

Let's see a simple example of multiple inheritance.

1. #include <iostream>
2. **using** **namespace** std;
3. **class** A
4. {
5. **protected**:
6. **int** a;
7. **public**:
8. **void** get\_a(**int** n)
9. {
10. a = n;
11. }
12. };
14. **class** B
15. {
16. **protected**:
17. **int** b;
18. **public**:
19. **void** get\_b(**int** n)
20. {
21. b = n;
22. }
23. };
24. **class** C : **public** A,**public** B
25. {
26. **public**:
27. **void** display()
28. {
29. std::cout << "The value of a is : " <<a<< std::endl;
30. std::cout << "The value of b is : " <<b<< std::endl;
31. cout<<"Addition of a and b is : "<<a+b;
32. }
33. };
34. **int** main()
35. {
36. C c;
37. c.get\_a(10);
38. c.get\_b(20);
39. c.display();
41. **return** 0;
42. }

Output:

The value of a is : 10

The value of b is : 20

Addition of a and b is : 30

In the above example, class 'C' inherits two base classes 'A' and 'B' in a public mode.

## **Ambiguity Resolution in Inheritance**

Ambiguity can be occurred in using the multiple inheritance when a function with the same name occurs in more than one base class.

Let's understand this through an example:

1. #include <iostream>
2. **using** **namespace** std;
3. **class** A
4. {
5. **public**:
6. **void** display()
7. {
8. std::cout << "Class A" << std::endl;
9. }
10. };
11. **class** B
12. {
13. **public**:
14. **void** display()
15. {
16. std::cout << "Class B" << std::endl;
17. }
18. };
19. **class** C : **public** A, **public** B
20. {
21. **void** view()
22. {
23. display();
24. }
25. };
26. **int** main()
27. {
28. C c;
29. c.display();
30. **return** 0;
31. }

**Output:**

error: reference to 'display' is ambiguous

display();

* The above issue can be resolved by using the class resolution operator with the function. In the above example, the derived class code can be rewritten as:

1. **class** C : **public** A, **public** B
2. {
3. **void** view()
4. {
5. A :: display();         // Calling the display() function of class A.
6. B :: display();         // Calling the display() function of class B.
8. }
9. };

An ambiguity can also occur in single inheritance.

Consider the following situation:

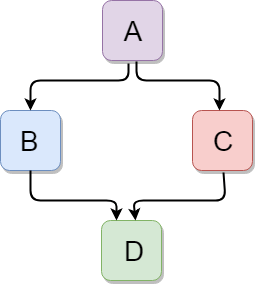
1. **class** A
2. {
3. **public**:
4. **void** display()
5. {
6. cout<<?Class A?;
7. }
8. } ;
9. **class** B
10. {
11. **public**:
12. **void** display()
13. {
14. cout<<?Class B?;
15. }
16. } ;

In the above case, the function of the derived class overrides the method of the base class. Therefore, call to the display() function will simply call the function defined in the derived class. If we want to invoke the base class function, we can use the class resolution operator.

1. **int** main()
2. {
3. B b;
4. b.display();               // Calling the display() function of B class.
5. b.B :: display();       // Calling the display() function defined in B class.
6. }

## **C++ Hybrid Inheritance**

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



Let's see a simple example:

1. #include <iostream>
2. **using** **namespace** std;
3. **class** A
4. {
5. **protected**:
6. **int** a;
7. **public**:
8. **void** get\_a()
9. {
10. std::cout << "Enter the value of 'a' : " << std::endl;
11. cin>>a;
12. }
13. };
15. **class** B : **public** A
16. {
17. **protected**:
18. **int** b;
19. **public**:
20. **void** get\_b()
21. {
22. std::cout << "Enter the value of 'b' : " << std::endl;
23. cin>>b;
24. }
25. };
26. **class** C
27. {
28. **protected**:
29. **int** c;
30. **public**:
31. **void** get\_c()
32. {
33. std::cout << "Enter the value of c is : " << std::endl;
34. cin>>c;
35. }
36. };
38. **class** D : **public** B, **public** C
39. {
40. **protected**:
41. **int** d;
42. **public**:
43. **void** mul()
44. {
45. get\_a();
46. get\_b();
47. get\_c();
48. std::cout << "Multiplication of a,b,c is : " <<a\*b\*c<< std::endl;
49. }
50. };
51. **int** main()
52. {
53. D d;
54. d.mul();
55. **return** 0;
56. }

Output:

Enter the value of 'a' :

10

Enter the value of 'b' :

20

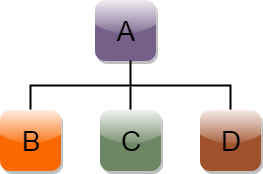
Enter the value of c is :

30

Multiplication of a,b,c is : 6000

## **C++ Hierarchical Inheritance**

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



**Syntax of Hierarchical inheritance:**

1. **class** A
2. {
3. // body of the class A.
4. }
5. **class** B : **public** A
6. {
7. // body of class B.
8. }
9. **class** C : **public** A
10. {
11. // body of class C.
12. }
13. **class** D : **public** A
14. {
15. // body of class D.
16. }

Let's see a simple example:

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Shape                 // Declaration of base class.
4. {
5. **public**:
6. **int** a;
7. **int** b;
8. **void** get\_data(**int** n,**int** m)
9. {
10. a= n;
11. b = m;
12. }
13. };
14. **class** Rectangle : **public** Shape  // inheriting Shape class
15. {
16. **public**:
17. **int** rect\_area()
18. {
19. **int** result = a\*b;
20. **return** result;
21. }
22. };
23. **class** Triangle : **public** Shape    // inheriting Shape class
24. {
25. **public**:
26. **int** triangle\_area()
27. {
28. **float** result = 0.5\*a\*b;
29. **return** result;
30. }
31. };
32. **int** main()
33. {
34. Rectangle r;
35. Triangle t;
36. **int** length,breadth,base,height;
37. std::cout << "Enter the length and breadth of a rectangle: " << std::endl;
38. cin>>length>>breadth;
39. r.get\_data(length,breadth);
40. **int** m = r.rect\_area();
41. std::cout << "Area of the rectangle is : " <<m<< std::endl;
42. std::cout << "Enter the base and height of the triangle: " << std::endl;
43. cin>>base>>height;
44. t.get\_data(base,height);
45. **float** n = t.triangle\_area();
46. std::cout <<"Area of the triangle is : "  << n<<std::endl;
47. **return** 0;
48. }

**Output:**

Enter the length and breadth of a rectangle:

23

20

Area of the rectangle is : 460

Enter the base and height of the triangle:

2

5

Area of the triangle is : 5

**Exercises:**

* We want to calculate the total marks of each student of a class in Physics, Chemistry and Mathematics and the average marks of the class. The number of students in the class are entered by the user. Create a class named Marks with data members for roll number, name and marks. Create three other classes inheriting the Marks class, namely Physics, Chemistry and Mathematics, which are used to define marks in individual subject of each student. Roll number of each student will be generated automatically.
* Make a class named Fruit with a data member to calculate the number of fruits in a basket. Create two other class named Apples and Mangoes to calculate the number of apples and mangoes in the basket. Print the number of fruits of each type and the total number of fruits in the basket.

**Make the following classes and perform required inheritance**

* (1) Birthdate (class birthday) contain: year, month, day and other data members   
  (2) Teacher (class teacher) contain: num, name, sex and other data members   
  (3) Professor (Teaching) contains data members: teacher class and class birthday.

**Requirement:**

(1)Through the use of inheritance and combination of the Teacher and Birthdate methods to design Professor  
(2)Define a Professor class object Prof, and gives the initial value of all data  
(3)The birthday data modification Prof  
(4)All the latest data output Prof

## **Input**

num,name,sex,year,month,Day and modified year,month,day

**Output**

num,name,sex,year,month,day

## **Sample Input**

2001 Huang m1970 1 11994 5 26

## **Sample Output**

* num:2001name:Huangsex:mbirthday:1994/5/26