**http://www.trytoprogram.com/cplusplus-programming/access-specifiers/**

**Briefly discuss about the structure of a C++ Class.**

**Structure of a C++ Class**

A **class** is user-defined datatype which has data members and member functions. In C++, classes are divided into the following three sections:

**1**. Class Declaration Section  
**2**. Member Function Section  
**3**. Main Function Section

**Class Declaration Section**

* You can declare class using the keyword class
* You can add data members and member functions inside the class.
* Accessing the data members and member functions depends solely on the access modifiers, i.e., **public, private** or **protected.**
* A class always starts with { and ends with };
* class Animal{  
     string animalName;  
     public:  
     Animal(string name);  
     void animalSound();  
     void printName();  
  };

**Member Function Section**

* Member functions are methods used to access data members of the class.
* Definition of member functions can be inside the class or outside the class.
* This section contains definition of the member functions that are declared inside the class.
* void Animal ::Animal(string name){  
        animalName = name;  
     }  
     void Animal :: animalSound(){  
        cout << "Woof Woof!" << endl;  
     }  
     void Animal :: printName(){  
        cout << animalName << endl;  
     }

**Main Function Section**

* Execution of the code starts from main.
* In this section, we can create an object of a class.
* We can call the functions defined inside the class.
* int main(){  
     Animal a("Dog");  
     a.printName();  
     a.animalSound();  
  }
* #include <iostream>     // Header files
* using namespace std;
* class Animal{    // class name
* string animalName;      // class data member
* public:
* Animal(string name);   // Animal constructor
* void animalSound();    // Member function declaration
* void printName();
* };
* Animal :: Animal(string name){   // function defintion
* animalName = name;
* }
* void Animal :: animalSound(){
* cout << "Woof Woof!" << endl;
* }
* void Animal :: printName(){
* cout << animalName << endl;
* }
* int main(){
* Animal a("Dog");    // creating object of class Animal
* a.printName();  // calling member function
* a.animalSound();  // calling member function
* }

# Explain C++ Access Specifiers with suitable example.

# C++ Access Specifiers – Private, Public and Protected

**C++ access specifiers** are used for determining or setting the boundary for the availability of class members (data members and [member functions](http://www.trytoprogram.com/cplusplus-programming/class-and-functions/)) beyond that class.

For example, the class members are grouped into sections, private protected and public. These keywords are called **access specifiers** which define the accessibility or visibility level of class members.

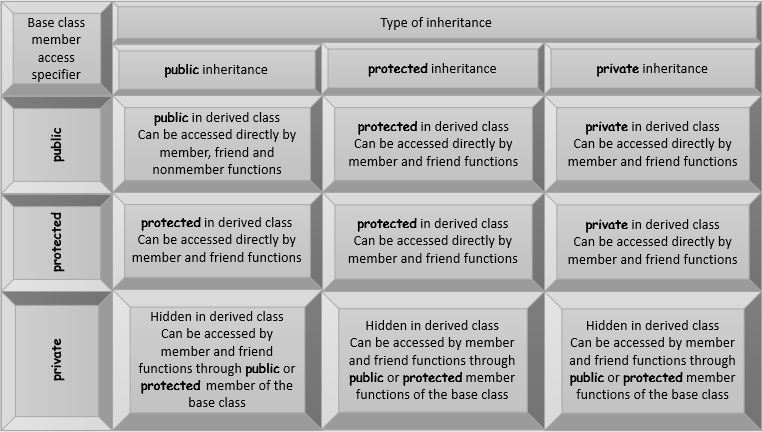
By default the class members are private. So if the visibility labels are missing then by default all the class members are private.

In [inheritance](http://www.trytoprogram.com/cplusplus-programming/inheritance/), it is important to know when a member function in the base class can be used by the objects of the derived class. This is called accessibility and the access specifiers are used to determine this.

## C++ Access Specifiers

**Access specifier** can be either private or protected or public. In general access specifiers are the access restriction imposed during the derivation of different subclasses from the base class.

* private access specifier
* protected access specifier
* public access specifier

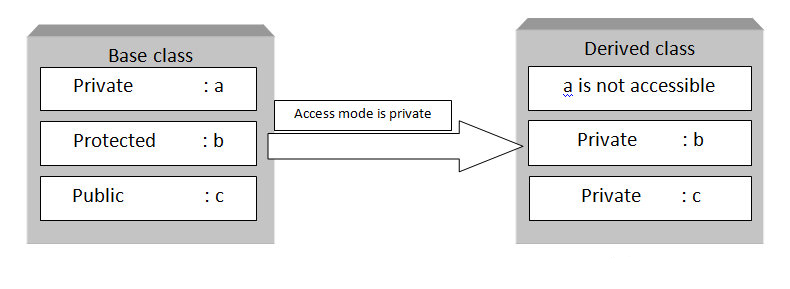


## ****private access specifier****

If **private access specifier** is used while creating a class, then the public and protected data members of the base class become the private member of the derived class and private member of base class remains private.

In this case, the members of the base class can be used only within the derived class and cannot be accessed through the object of derived class whereas they can be accessed by creating a function in the derived class.

Following block diagram explain how data members of base class are inherited when derived class access mode is private.



Note: Declaring data members with private access specifier is known as data hiding.

Sample program demonstrating private access specifier

// private access specifier.cpp

#include <iostream>

using namespace std;

class base

{

private:

int x;

protected:

int y;

public:

int z;

base() //constructor to initialize data members

{

x = 1;

y = 2;

z = 3;

}

};

class derive: private base

{

//y and z becomes private members of class derive and x remains private

public:

void showdata()

{

cout << "x is not accessible" << endl;

cout << "value of y is " << y << endl;

cout << "value of z is " << z << endl;

}

};

int main()

{

derive a; //object of derived class

a.showdata();

//a.x = 1; not valid : private member can't be accessed outside of class

//a.y = 2; not valid : y is now private member of derived class

//a.z = 3; not valid : z is also now a private member of derived class

return 0;

} //end of program

**Output**

x is not accessible

value of y is 2

value of z is 3

**Explanation**  
When a class is derived from the base class with private access specifier the private members of the base class can’t be accessed. So in above program, the derived class cannot access the

So in above program, the derived class cannot access the member x which is private in the base class, however, derive class has access to the protected and public members of the base class. So the

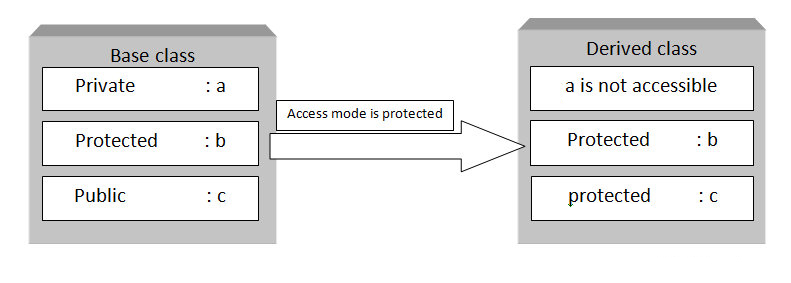
Hence the function showdata in derived class can access the public and protected member of the base class.

## ****Protected Access Specifier****

If **protected access specifier** is used while deriving class then the public and protected data members of the base class becomes the protected member of the derived class and private member of the base class are inaccessible.

In this case, the members of the base class can be used only within the derived class as protected members except for the private members.

Following block diagram explain how data members of base class are inherited when derived class access mode is protected.



### Sample program demonstrating protected access specifier

// protected access specifier.cpp

#include <iostream>

using namespace std;

class base

{

private:

int x;

protected:

int y;

public:

int z;

base() //constructor to initialize data members

{

x = 1;

y = 2;

z = 3;

}

};

class derive: protected base

{

//y and z becomes protected members of class derive

public:

void showdata()

{

cout << "x is not accessible" << endl;

cout << "value of y is " << y << endl;

cout << "value of z is " << z << endl;

}

};

int main()

{

derive a; //object of derived class

a.showdata();

//a.x = 1; not valid : private member can't be accessed outside of class

//a.y = 2; not valid : y is now private member of derived class

//a.z = 3; not valid : z is also now a private member of derived class

return 0;

} //end of program

**Output**

x is not accessible

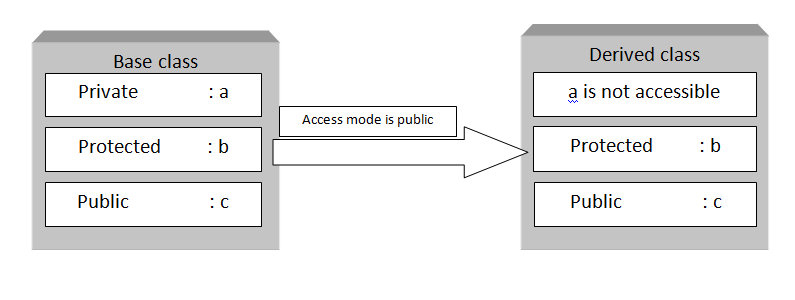
value of y is 2

value of z is 3

## ****public access specifier****

If **public access specifie**r is used while deriving class then the public data members of the base class becomes the public member of the derived class and protected members becomes the protected in the derived class but the private members of the base class are inaccessible.

Following block diagram explain how data members of base class are inherited when derived class access mode is public



### Sample program demonstrating public access specifier

// public access specifier.cpp

#include <iostream>

using namespace std;

class base

{

private:

int x;

protected:

int y;

public:

int z;

base() //constructor to initialize data members

{

x = 1;

y = 2;

z = 3;

}

};

class derive: public base

{

//y becomes protected and z becomes public members of class derive

public:

void showdata()

{

cout << "x is not accessible" << endl;

cout << "value of y is " << y << endl;

cout << "value of z is " << z << endl;

}

};

int main()

{

derive a; //object of derived class

a.showdata();

//a.x = 1; not valid : private member can't be accessed outside of class

//a.y = 2; not valid : y is now private member of derived class

//a.z = 3; not valid : z is also now a private member of derived class

return 0;

} //end of program

**Output**

x is not accessible

value of y is 2

value of z is 3

http://www.tutorialdost.com/Cpp-Programming-Tutorial/34-CPP-Class-Array.aspx

## What is C++ Array of Objects?

## An object of class represents a single record in memory, if we want more than one record of class type, we have to create an array of class or object. As we know, an array is a collection of similar type, therefore an array can be a collection of class type.

### Mention the Syntax for Array of object

class class-name

{

datatype var1;

datatype var2;

- - - - - - - - - -

datatype varN;

method1();

method2();

- - - - - - - - - -

methodN();

};

class-name obj[ size ];

### Give an Example for Array of object

#include<iostream.h>

#include<conio.h>

class Employee

{

int Id;

char Name[25];

int Age;

long Salary;

public:

void GetData() //Statement 1 : Defining GetData()

{

cout<<"\n\tEnter Employee Id : ";

cin>>Id;

cout<<"\n\tEnter Employee Name : ";

cin>>Name;

cout<<"\n\tEnter Employee Age : ";

cin>>Age;

cout<<"\n\tEnter Employee Salary : ";

cin>>Salary;

}

void PutData() //Statement 2 : Defining PutData()

{

cout<<"\n"<<Id<<"\t"<<Name<<"\t"<<Age<<"\t"<<Salary;

}

};

void main()

{

int i;

Employee E[3]; //Statement 3 : Creating Array of 3 Employees

for(i=0;i<3;i++)

{

cout<<"\nEnter details of "<<i+1<<" Employee";

E[i].GetData();

}

cout<<"\nDetails of Employees";

for(i=0;i<3;i++)

E[i].PutData();

}

Output :

Enter details of 1 Employee

Enter Employee Id : 101

Enter Employee Name : Suresh

Enter Employee Age : 29

Enter Employee Salary : 45000

Enter details of 2 Employee

Enter Employee Id : 102

Enter Employee Name : Mukesh

Enter Employee Age : 31

Enter Employee Salary : 51000

Enter details of 3 Employee

Enter Employee Id : 103

Enter Employee Name : Ramesh

Enter Employee Age : 28

Enter Employee Salary : 47000

Details of Employees

101 Suresh 29 45000

102 Mukesh 31 51000

103 Ramesh 28 47000

In the above example, we are getting and displaying the data of 3 employee using array of object. Statement 1 is creating an array of Employee Emp to store the records of 3 employees.

**https://www.geeksforgeeks.org/difference-between-argument-and-parameter-in-c-c-with-examples/**

**Difference between Argument and Parameter in C++ with Examples**

**Argument**

An **argument**is referred to the values that are passed within a function when the function is called. These values are generally the source of the function that require the arguments during the process of execution. These values are assigned to the variables in the definition of the function that is called. The type of the values passed in the function is the same as that of the variables defined in the function definition. These are also called **Actual arguments** or **Actual Parameters**.

**Example:** Suppose a sum() function is needed to be called with two numbers to add. These two numbers are referred to as the arguments and are passed to the sum() when it called from somewhere else.

|  |
| --- |
| // C++ code to illustrate Arguments  #include <iostream>  using namespace std;    // sum: Function defintion  int sum(int a, int b)  {      // returning the addition      return a + b;  }    // Driver code  int main()  {      int num1 = 10, num2 = 20, res;        // sum() is called with      // num1 & num2 as ARGUMENTS.      res = sum(num1, num2);        // Displaying the result      cout << "The summation is " << res;      return 0;  } |

**Output:**

The summation is 30

**Parameters**

The parameter is referred to as the variables that are defined during a function declaration or definition. These variables are used to receive the arguments that are passed during a function call. These parameters within the function prototype are used during the execution of the function for which it is defined. These are also called Formal arguments or Formal Parameters.  
**Example:** Suppose a Mult() function is needed to be defined to multiply two numbers. These two numbers are referred to as the parameters and are defined while defining the function Mult().

|  |
| --- |
| // C++ code to illustrate Parameters    #include <iostream>  using namespace std;    // Mult: Function defintion  // a and b are the parameters  int Mult(int a, int b)  {      // returning the multiplication      return a \* b;  }    // Driver code  int main()  {      int num1 = 10, num2 = 20, res;        // Mult() is called with      // num1 & num2 as ARGUMENTS.      res = Mult(num1, num2);        // Displaying the result      cout << "The multiplication is " << res;      return 0;  } |

**Output:**

The multiplication is 200