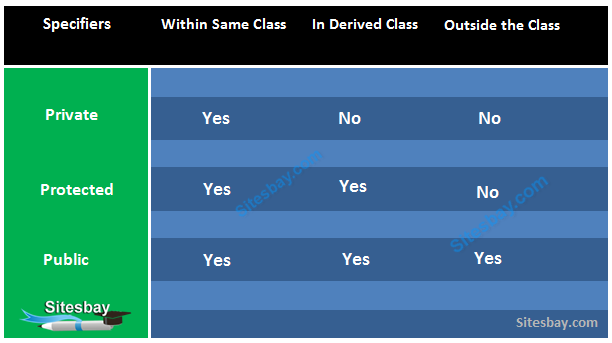
**Access Specifiers in C++**

**Access specifiers** in C++ define how the members of the class can be accessed. C++ has 3 new keywords introduced, namely.

* public
* private
* protected



The keywords public, private, and protected are called access specifiers. A class can have multiple public, protected, or private labeled sections.

**Note:**By default, all members and function of a class is private i.e if no access specifier is specified.

**Syntax of Declaring Access Specifiers in C++**

**Syntax**

class

{

private:

// private members and function

public:

// public members and function

protected:

// protected members and function

};

**Public Access Specifier in C++**

Public class members are accessible out side the class and it is available for every one.

**Syntax**

class Public\_Access\_Specifier

{

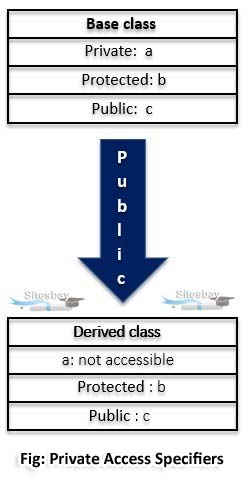
public: // public access specifier

int a; // Data Member Declaration

void display(); // Member Function declaration

}

While derive class is created, if public access specifier is used, the public data members of the base class become the public member of the derived class and protected members become the protected in the derived class but the private members of the base class are inaccessible. The following figure depicts the inheritance of data members of the base class when the access mode of the derived class is public.



**Public Access Specifier Example in C++**

**Public Access Specifier Example in C++**

#include<iostream.h>

**using** **namespace** std;

**class** baseclass

{

**private**:

**int** u;

**protected**:

**int** v;

**public**:

**int** w;

baseclass()

{

u = 3;

v = 4;

w = 5;

}

};

**class** deriveclass: **public** baseclass

{

//v becomes protected and w becomes public members of class derive

**public**:

**void** show()

{

cout << "u is not accessible";

cout << "\nvalue of v is " << v;

cout << "]\nvalue of w is " << w;

}

};

**int** main()

{

deriveclass c;

c.show();

//c.u = 3; not valid: private members are inaccessible outside the class

//c.v = 4; not valid: v is now protected member of derived class

//c.w = 5; valid: w is now a public member of derived class

**return** 0;

}

**Output**

u is not accessible

value of v is 4

value of w is 5

**Private Access Specifier in C++**

Private class members are accessible with the class and it is not accessible out side the class. If some one try to access out side the it gives compile time error. By default class variables and member functions are private.

**Syntax**

class Private\_Access\_Specifier

{

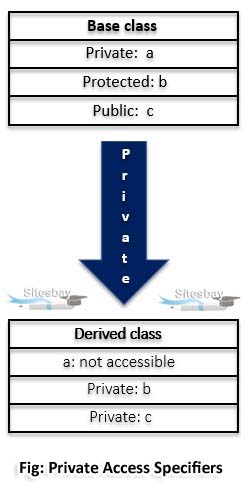
private: // private access specifier

int a; // Data Member Declaration

void display(); // Member Function declaration

}

While creating a class using access specifier as private, the base class’ public and protected data members become the derived class’ private member and base class’ private member stays private. Hence, the members of the base class can be used only inside the derived class but are inaccessible through the object created for the derived class. The other way to access them is to create a function in the derived class. The below figure depicts the inheritance of data members of the base class when the access mode of the derived class is private.



**Private Access Specifier Example in C++**

**Private Access Specifier Example in C++**

#include<iostream.h>

**using** **namespace** std;

**class** baseclass

{

**private**:

**int** s;

**protected**:

**int** t;

**public**:

**int** u;

baseclass()

{

s = 11;

t = 12;

u = 13;

}

};

**class** deriveclass: **private** baseclass

{

//t and u becomes private members of deriveclass and s will remain private

**public**:

**void** show ()

{

cout << "s is not accessible";

cout << "\nt is " << t;

cout << "\nu is " << u;

}

};

**int** main()

{

deriveclass l; //object created

l.show();

//l.s = 11; not valid : private members are inaccessible outside the class

//l.t = 12; not valid

//l.u = 13; not valid : t and u have become derived class’ private members

**return** 0;

}

**Output**

s is not accessible

t is 12

u is 13

**Private and Public Access Specifier Example in C++**

**Example**

#include<iostream.h>

#include<conio.h>

**class** A

{

**private**:

**int** a;

**public**:

**int** b;

**public**:

**void** show()

{

a=10 ;

b=20;

clrscr();

//Every members can be access here, same class

cout<<"\nAccessing variable within the class"<<endl;

cout<<"Value of a: "<<a<<endl;

cout<<"Value of b: "<<b<<endl;

}

};

**void** main()

{

A obj; // create object

obj.show();

cout<<"\nAccessing variable outside the class"<<endl;

//'a' cannot be accessed as it is private

//cout<<"value of a: "<<obj.a<<endl;

//'b' is public as can be accessed from any where

cout<<"value of b: "<<obj.b<<endl;

getch();

}

**Note:**If here, we access variable a in side main method it will give compile time error

**Output**

Accessing variable within the class

value of a: 10

value of b: 20

value of c: 30

Accessing variable outside the class

Value of b: 20

**Protected Access Specifier in C++**

It is similar to private access specifier. It makes class member inaccessible outside the class. But they can be accessed by any subclass of that class.

**Syntax**

class Protected\_Access\_Specifier

{

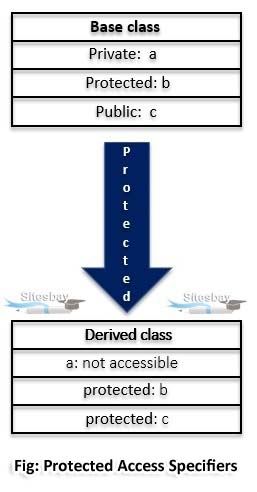
protected: // protected access specifier

int a; // Data Member Declaration

void display(); // Member Function Declaration

}

In derived class, when protected access specifier is used, the public and protected data members of the base class becomes the derived class’ protected member and base class’ private member are not accessible. Hence, the members of the base class can be used only inside the derived class as protected members. The below figure depicts the inheritance of data members of the base class when the access mode of the derived class is protected.



**Protected Access Specifier Example in C++**

**Protected Access Specifier Example in C++**

#include<iostream.h>

**using** **namespace** std;

**class** baseclass

{

**private**:

**int** a;

**protected**:

**int** b;

**public**:

**int** c;

baseclass()

{

a = 10;

b = 11;

c = 12;

}

};

**class** deriveclass: **protected** baseclass

{

//b and c becomes protected members of deriveclass

**public**:

**void** show ()

{

cout << "a is not accessible";

cout << "\nb is " << b;

cout << "\nc is " << c;

}

};

**int** main()

{

deriveclass d; // object created

d.show();

//d.a = 10; not valid : private members are inaccessible outside the class

//d.b = 11; not valid

//d.c = 12; not valid : b and c have become derived class’ private member

**return** 0;

}

**Output**

a is not accessible

b is 11

c is 12

**Access Specifier Example in C++**

In below example I will show you all these access specifier public, private and protected.

**Access Specifier Example in C++**

#include<iostream.h>

#include<conio.h>

//using namespace std;

**class** Declaration

{

**private**:

**int** a;

**public**:

**int** b;

**protected**:

**int** c;

**public**:

**void** show()

{

a=10;

b=20;

c=30;

//Every members can be access here, same class

cout<<"\nAccessing variable within the class"<<endl;

cout<<"Value of a: "<<a<<endl;

cout<<"Value of b: "<<b<<endl;

cout<<"Value of c: "<<c<<endl;

}

};

**class** Sub\_class:**public** Declaration

{

**public**:

**void** show()

{

b=5;

c=6;

cout<<"\nAccessing variable in sub the class"<<endl;

// a is not accessible here it is private

//cout<<"Value of a: "<<a<<endl;

//b is public so it is accessible any where

cout<<"Value of b: "<<b<<endl;

//'c' is declared as protected, so it is accessible in sub class

cout<<"Value of c: "<<c<<endl;

}

};

**void** main()

{

clrscr();

Declaration d; // create object

d.show();

Sub\_class s; // create object

s.show(); // Sub class show() function

cout<<"\nAccessing variable outside the class"<<endl;

//'a' cannot be accessed as it is private

//cout<<"value of a: "<<d.a<<endl;

//'b' is public as can be accessed from any where

cout<<"value of b: "<<d.b<<endl;

//'c' is protected and cannot be accesed here

//cout<<"value of c: "<<d.c<<endl;

getch();

}

**Output**

Accessing variable within the class

value of a: 10

value of b: 20

value of c: 30

Accessing variable in sub class

value of b: 5

value of c: 6

Accessing variable outside the class

Value of b: 20