Access Specifiers in C++

Access specifiers in C++ determine the visibility and accessibility of class members (variables and functions). They are used to specify which parts of a class are accessible from outside the class, and which parts are restricted.

# 1. Public Access Specifier (public)

Members declared as public are accessible from anywhere, both inside and outside the class. It is typically used for functions and variables that you want to make accessible to the outside world.

Example:

#include <iostream>  
using namespace std;  
  
class MyClass {  
public:  
 int x; // Public member variable  
  
 void display() { // Public member function  
 cout << "Value of x: " << x << endl;  
 }  
};  
  
int main() {  
 MyClass obj;  
 obj.x = 10; // Accessing public variable directly  
 obj.display(); // Calling public function  
 return 0;  
}

Output:

Value of x: 10

# 2. Private Access Specifier (private)

Members declared as private are only accessible inside the class and not from outside. It is used to restrict access to variables and functions that should only be modified or used within the class itself.

Example:

#include <iostream>  
using namespace std;  
  
class MyClass {  
private:  
 int x; // Private member variable  
  
public:  
 void setX(int val) { // Public member function to set x  
 x = val;  
 }  
  
 void display() { // Public member function to display x  
 cout << "Value of x: " << x << endl;  
 }  
};  
  
int main() {  
 MyClass obj;  
 obj.setX(10); // Accessing private member through a public function  
 obj.display(); // Calling public function to access private member  
 // obj.x = 10; // Error: Cannot access private member directly  
 return 0;  
}

Output:

Value of x: 10

# 3. Protected Access Specifier (protected)

Members declared as protected are accessible within the class and by derived classes, but not from outside the class. It is commonly used in inheritance scenarios, where a base class wants to allow access to certain members for derived classes but not for general users.

Example:

#include <iostream>  
using namespace std;  
  
class Base {  
protected:  
 int x; // Protected member variable  
  
public:  
 void setX(int val) { // Public member function to set x  
 x = val;  
 }  
};  
  
class Derived : public Base {  
public:  
 void display() { // Derived class can access the protected member of Base class  
 cout << "Value of x from Derived class: " << x << endl;  
 }  
};  
  
int main() {  
 Derived obj;  
 obj.setX(10); // Accessing protected member through a public function of Base class  
 obj.display(); // Calling derived class function that accesses protected member  
 // obj.x = 10; // Error: Cannot access protected member directly  
 return 0;  
}

Output:

Value of x from Derived class: 10

# Key Differences Between Access Specifiers

|  |  |  |
| --- | --- | --- |
| Access Specifier | Accessible by | Usage |
| public | Anywhere (inside/outside the class) | Use for members that need to be accessible publicly |
| private | Only within the class itself | Use to hide data members or functions that should not be modified directly by outside code |
| protected | Within the class and derived classes | Use when you want derived classes to access base class members, but prevent access from outside the class hierarchy |

# Default Access Level

For classes: If no access specifier is provided, class members are private by default.

For structs: If no access specifier is provided, struct members are public by default.

# Inheritance and Access Specifiers

In the context of inheritance, the access level of base class members can be modified using access specifiers:

1. Public inheritance: Public members of the base class remain public in the derived class, protected members remain protected, and private members remain inaccessible.  
2. Protected inheritance: Public and protected members of the base class become protected in the derived class, and private members remain inaccessible.  
3. Private inheritance: Public and protected members of the base class become private in the derived class.

# Access Control Summary

Access specifiers in C++ are essential for controlling access to class members. They help in implementing encapsulation, one of the core principles of object-oriented programming (OOP), by protecting class members from unauthorized access and modification.