Access Specifier in C++

By:
freeCSIT

C++ course with PDF Notes

What is **Access Specifier**?

Access specifiers in C++ are keywords that determine the accessibility or visibility of class members (data members and member functions) to other parts of a program

Classes and Abstraction in C++

keywords

1 Access Specifier

Access specifiers define how the members (attributes and methods) of a class can be accessed.

Access modifiers are used to implement an important aspect of OOP known as **Data Hiding**

public

Data members can be accessible from any where(outside the class)

private

Data members can not be accessible from the outside the class

protected

Accessible within the class and derived classes.

Classes and Abstraction in C++

keywords

1 Access Specifier

Specifier	Same class	Outside class	Derived class
Public	Yes	Yes	Yes
Private	Yes	No	No
protected	Yes	No	Yes

public

Data members can be accessible from any where(outside the class)

private

Data members can not be accessible from the outside the class

protected

Accessible within the class and derived classes.

public

```
#include <iostream>
using namespace std;

class Example {
public:
    int publicVar; // Public data member

    void display() { // Public member function
        cout << "Public variable value: " << publicVar << endl;
    }
};

int main() {
    Example obj;
    obj.publicVar = 10;  // Accessing public member directly
    obj.display();       // Accessing public function
    return 0;
}
```

private

```
#include <iostream>
using namespace std;

class Example {
private:
   int privateVar; // Private data member

public:
   void setPrivateVar(int value) { // Public function to set private variable
      privateVar = value;
   }

   void display() { // Public function to access private variable
      cout << "Private variable value: " << privateVar << endl;
   }
};

int main() {
   Example obj;
   // obj.privateVar = 10; // Error: privateVar is inaccessible
   obj.setPrivateVar(20);  // Accessing privateVar through public function
   obj.display();
   return 0;
}
```

protected

```
#include <iostream>
using namespace std;

class Base {
protected:
   int protectedVar; // Protected data member

public:
   void setProtectedVar(int value) {
      protectedVar = value;
   }
};

class Derived : public Base {
public:
   void display() {
      cout << "Protected variable value: " << protectedVar << endl;
   }
};

int main() {
   Derived obj;
   obj.setProtectedVar(30);  // Accessing protectedVar through public function
   obj.display();          // Accessing protectedVar in derived class
   return 0;
}
```

Key Points to Remember:

1. **Default Access Specifier:**
   - For **classes**, the default access specifier is private.
   - For **structs**, the default access specifier is public.
2. **Best Practices:**
   - Keep data members private or protected and provide public member
   - functions (getters and setters) to control access.
   - This approach helps maintain data integrity and hides the
   - implementation details.
3. **Friend Functions and Classes:**

- A friend function or class can access private and protected members
- of another class, even though they are not public.

Code demonstrate:

```
#include <iostream>
using namespace std;

class Example {
private:
   int privateVar; // Private data member

   // Friend function declaration
   friend void displayPrivateVar(Example obj);
};

// Friend function definition
void displayPrivateVar(Example obj) {
   obj.privateVar = 50; // Accessing private member
   cout << "Private variable value: " << obj.privateVar << endl;
}

int main() {
   Example obj;        // Creating an object of the class
   displayPrivateVar(obj); // Calling the friend function
   return 0;
}
```

Thanks
for
Watching

Please Subscribe