CodInClub
Powered By BridgeLabz

Wk6 /S3/ Lecture # : DSOOPS-27

Topics Covered

- Access Specifiers: public, private, and protected
- What Gets Access To What?
- Static Data Members
- Static Member Functions
- Local vs Instance vs Static vs Global Variables
- Practice Problems (2 Easy, 1 Medium, 1 Hard)

Access Specifiers: public, private, and protected

Access specifiers control *who can access* the members (variables and functions) inside a class.

`public`

- Members are accessible from anywhere (including outside the class — e.g., in main or other classes).

`private`

- Members are hidden from outside the class; accessible only by other members of the same class.

`protected`

- Members are accessible inside the class and by derived classes (subclasses), but not from outside code (like main or unrelated classes).

Example Showing All Three Access Specifiers

```
class Base {
public:
    int pub;          // Accessible everywhere
```

```
protected:
    int prot;         // Accessible in this class AND derived
classes
private:
    int priv;         // Only accessible inside Base
};

class Derived : public Base {
public:
    void foo() {
        pub = 1;    // OK
        prot = 2;   // OK (Derived is subclass of Base)
        // priv = 3; // ERROR! priv is private to Base only
    }
};

int main() {
    Base b;
    b.pub = 4;      // OK
    // b.prot = 5;  // ERROR! prot is protected, not accessible
here
    // b.priv = 6;  // ERROR! priv is private
}
```

What Gets Access To What?

Specifier	Accessible In Class	Accessible In Derived Classes	Accessible Outside Class
public	Yes	Yes	Yes

protected	Yes	Yes	No
private	Yes	No	No

Key Points

- Use public for members meant to be accessed from anywhere.
- Use private to hide sensitive members completely.
- Use protected to allow subclasses to access some members, but keep them hidden from other outside code.

Static Data Members

A static data member belongs to the *class itself*, not to any single object. All instances share the same static variable.

Declaring and Using Static Data Members

```
class Demo {
public:
    static int counter;     // Declaration inside class
    Demo() { counter++; }
};

int Demo::counter = 0;      // Definition outside class

int main() {
    Demo a, b;
    std::cout << Demo::counter << std::endl; // Prints 2
    return 0;
}
```

- Static variables are accessed using the class name: `Demo::counter`.
- All objects see the same shared value.

Static Member Functions

A static member function:

- Can be called without creating an object.
- Can only access static members (not instance members).

Example:

```
class Test {
public:
    static int square(int x) { return x * x; }
};

int main() {
    std::cout << Test::square(5) << std::endl; // Prints 25; No object needed
    return 0;
}
```

- Static member functions have no `this` pointer and can't access non-static members.

Local vs Instance vs Static vs Global Variables

Here's how these variable types differ in C++:

Variable Type	Location of Declaration	Lifetime	Scope	Shared Among Objects?	Access Syntax
Local	Inside function/block	During function/block execution	Only inside function/block	No	`variable`
Instance	Inside class (non-static)	Lifetime of object	Through object	No	`obj.member`
Static	Inside class with `static`	Entire program lifetime	Through class or object	Yes	`ClassName::member`
Global	Outside all functions/classes	Entire program lifetime	Whole file/program	Yes	`variable`

Local Variables

- Declared inside functions or blocks.
- Created on entering function/block; destroyed on exit.
- Only accessible within that block.

```
void func() {
```

```
    int x = 5;  // local variable
    std::cout << x << std::endl;
}
```

Instance Variables (Non-Static Members)

- Declared inside a class without `static`.
- Each object has its own copy.
- Created and destroyed with objects.

```
class Car {
public:
    int speed;  // instance variable
};

int main() {
    Car car1, car2;
    car1.speed = 50;
    car2.speed = 100;
    std::cout << car1.speed << ", " << car2.speed << std::endl;
// 50, 100
    return 0;
}
```

Static Variables (Static Data Members)

- Belong to the class, shared by all objects.
- Created at program start; live till termination.

```
class Car {
public:
    static int totalCars;
```

```
    Car() { totalCars++; }
};

int Car::totalCars = 0;

int main() {
    Car c1, c2;
    std::cout << Car::totalCars << std::endl; // 2
    return 0;
}
```

Global Variables

- Declared outside all classes/functions.
- Accessible anywhere (unless restricted by `static` keyword at file-level).
- Lifetime is the entire execution.

```
int globalCount = 0;

void increment() {
    globalCount++;
}

int main() {
    increment();
    std::cout << globalCount << std::endl; // 1
    return 0;
}
```

CodInClub

Powered By BridgeLabz

Practice Problems and Activities

Easy 1

What is the output? Explain why the commented line causes an error.

```
class SecretHolder {
private:
    int secret = 11;
public:
    int get() { return secret; }
};

int main() {
    SecretHolder s;
    // std::cout << s.secret << std::endl; // ERROR: secret is
private
    std::cout << s.get() << std::endl;      // Outputs 11
    return 0;
}
```

Easy 2

Add a static int variable `count` to the class Student that increments each time an object is created. Create 3 students and print `Student::count`.

Medium

Given this class:

```
class MathBox {
public:
    int n;
    static int instances;
```

```
    MathBox(int x) { n = x; instances++; }
};

int MathBox::instances = 0;

int main() {
    MathBox a(5), b(10);
    std::cout << MathBox::instances << std::endl; // What is
printed?
    return 0;
}
```

Hard

Write a class `IDGenerator` with:

- A private static int `nextID` initialized to 1,
- A public static function `generate()` that returns `nextID` and increments it each call.

In main, without creating an object, generate and print 3 IDs.

Wrap-Up & Key Takeaways

- Access specifiers:
  - public: accessible everywhere
  - protected: accessible inside class & subclasses
  - private: only inside class
- Static members belong to the class and are shared by all objects.
- Static member functions can be called without an object and only access static members.
- Local variables exist only inside functions/blocks.
- Instance variables belong to specific objects.
- Global variables exist throughout program execution and are accessible globally.
- Knowing the differences helps write safer, better-organized C++ programs.