C++

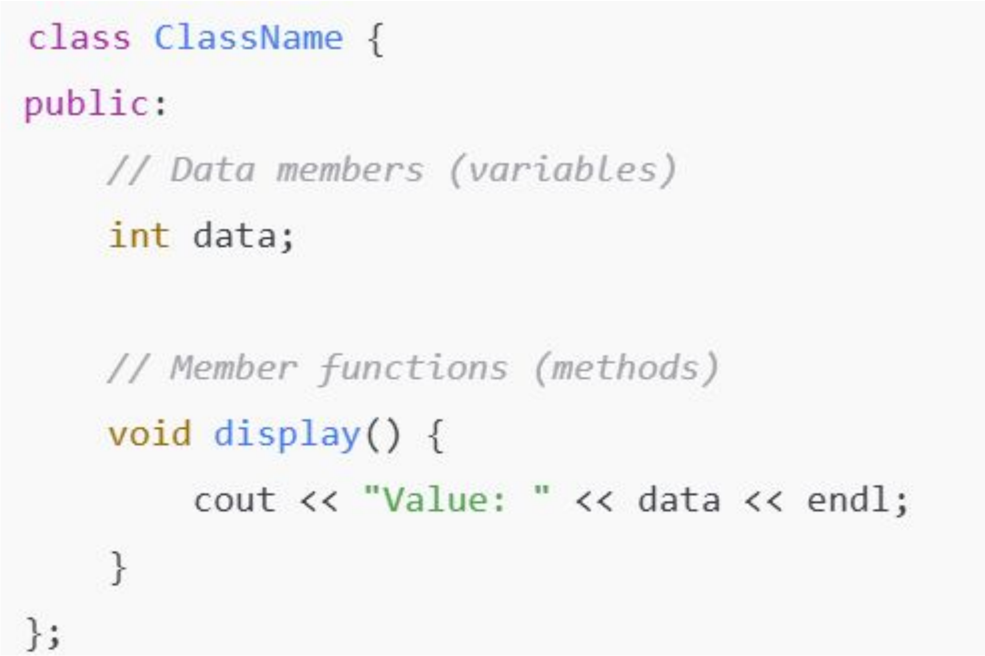
Things you need to know to get started with Data Structures

Class

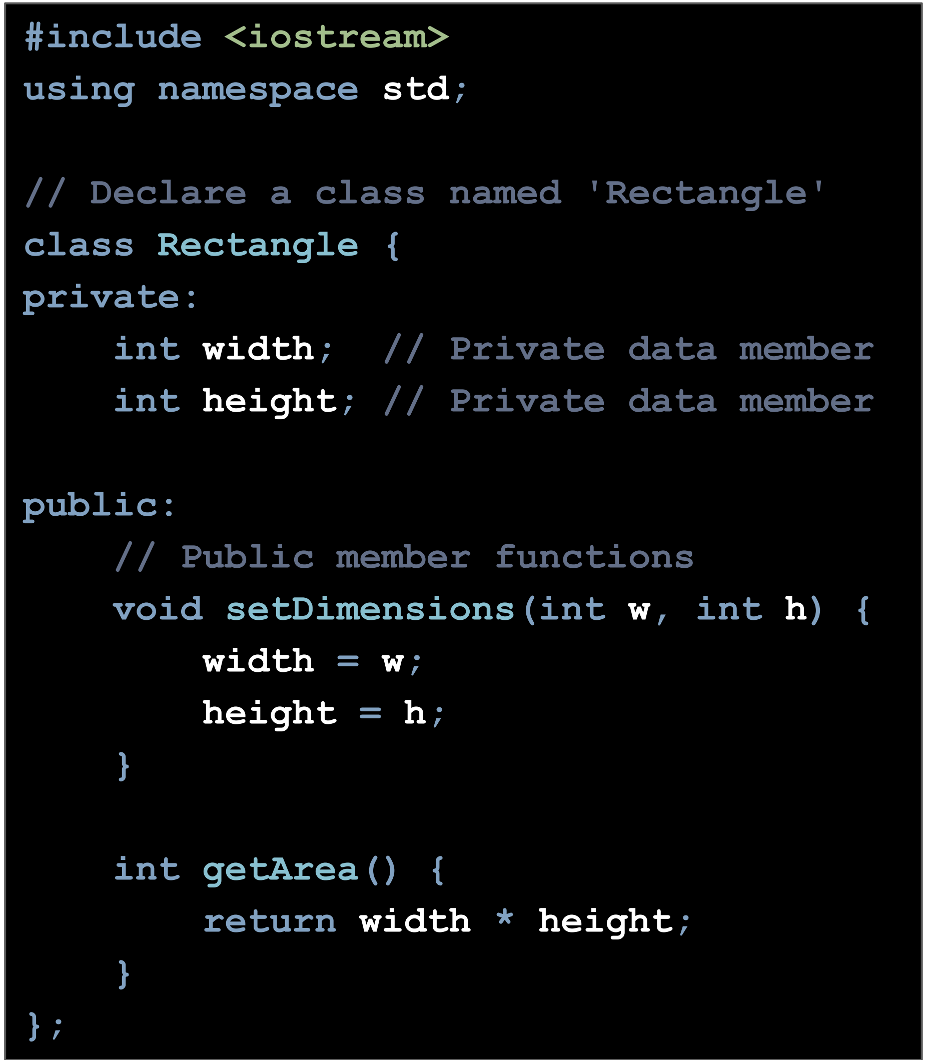
- In C++, a **class** is declared using the class keyword.
- A class is a **blueprint** for creating objects (instances) and can contain **data members** (variables) and **member functions** (methods).
- Here's the basic syntax for declaring a class:

Class

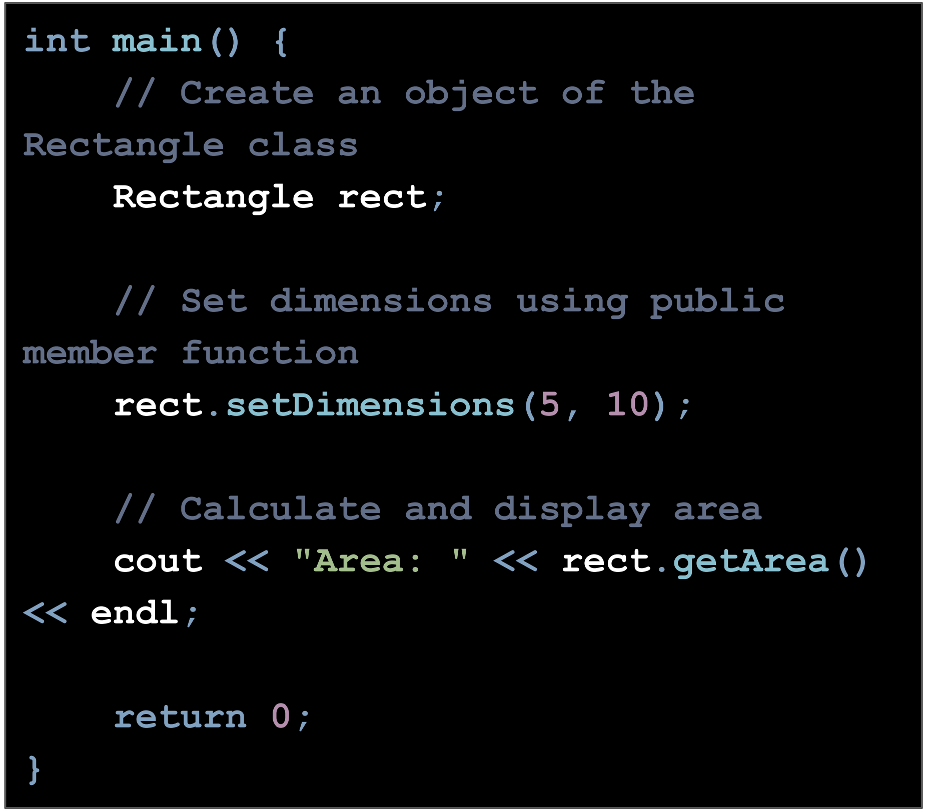
- In C++, a **class** is declared using the class keyword.
- A class is a **blueprint** for creating objects (instances) and can contain **data members** (variables) and **member functions** (methods).
- Here's the basic syntax for declaring a class:



```
class ClassName {
public:
    // Data members (variables)
    int data;

    // Member functions (methods)
    void display() {
        cout << "Value: " << data << endl;
    }
};
```



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

// Declare a class named 'Rectangle'
class Rectangle {
private:
    int width;  // Private data member
    int height; // Private data member

public:
    // Public member functions
    void setDimensions(int w, int h) {
        width = w;
        height = h;
    }

    int getArea() {
        return width * height;
    }
};
```



```cpp
int main() {
    // Create an object of the
Rectangle class
    Rectangle rect;

    // Set dimensions using public
member function
    rect.setDimensions(5, 10);

    // Calculate and display area
    cout << "Area: " << rect.getArea()
<< endl;

    return 0;
}
```

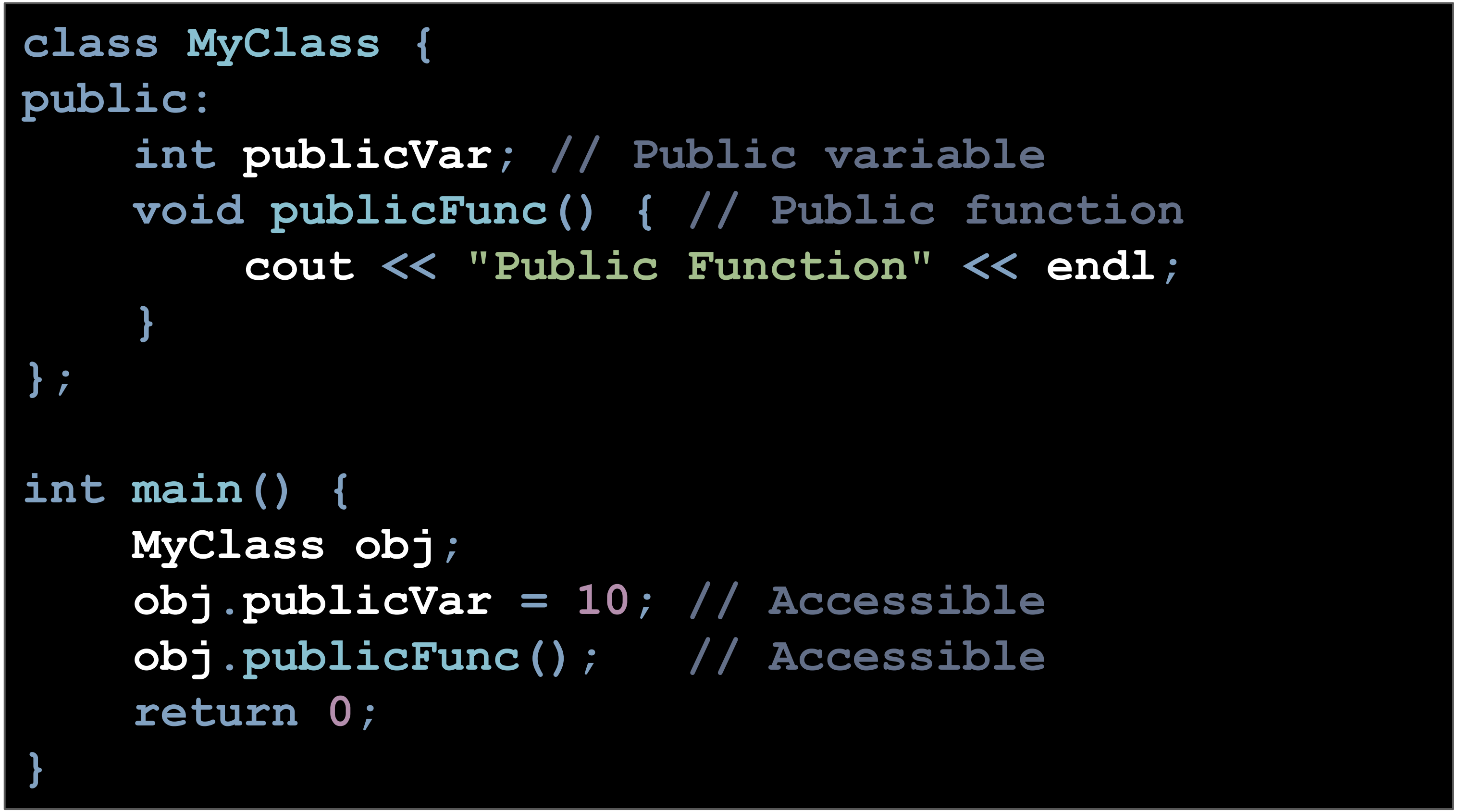
public, static, and private

In C++, ***public***, ***static***, and ***private*** are access specifiers and modifiers that control the **visibility, lifetime, and behavior** of class members (variables and functions).

public

- **Purpose**: Members declared as public are **accessible from anywhere** in the program, including outside the class.
- **Usage**: Used to define the **interface** of a class, allowing external code to interact with the class.
- **Example**:

public



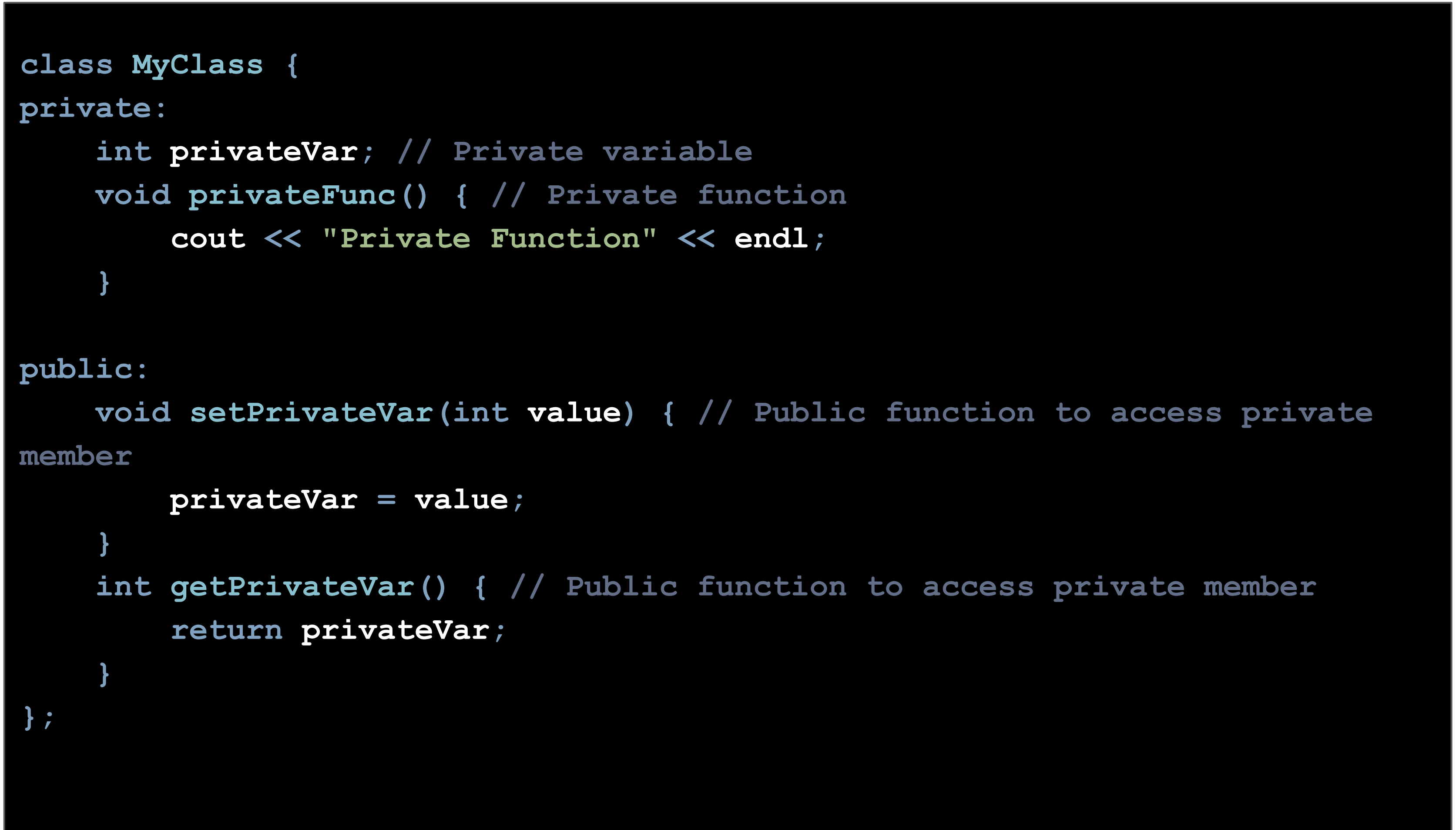
```
class MyClass {
public:
    int publicVar; // Public variable
    void publicFunc() { // Public function
        cout << "Public Function" << endl;
    }
};

int main() {
    MyClass obj;
    obj.publicVar = 10; // Accessible
    obj.publicFunc();   // Accessible
    return 0;
}
```

private

- **Purpose**: Members declared as private are ***accessible only within the class*** itself. They cannot be accessed directly from outside the class.

- **Usage**: Used to **encapsulate and hide implementation details**, ensuring data integrity and security.

private



```
class MyClass {
private:
    int privateVar; // Private variable
    void privateFunc() { // Private function
        cout << "Private Function" << endl;
    }

public:
    void setPrivateVar(int value) { // Public function to access private
member
        privateVar = value;
    }
    int getPrivateVar() { // Public function to access private member
        return privateVar;
    }
};
```

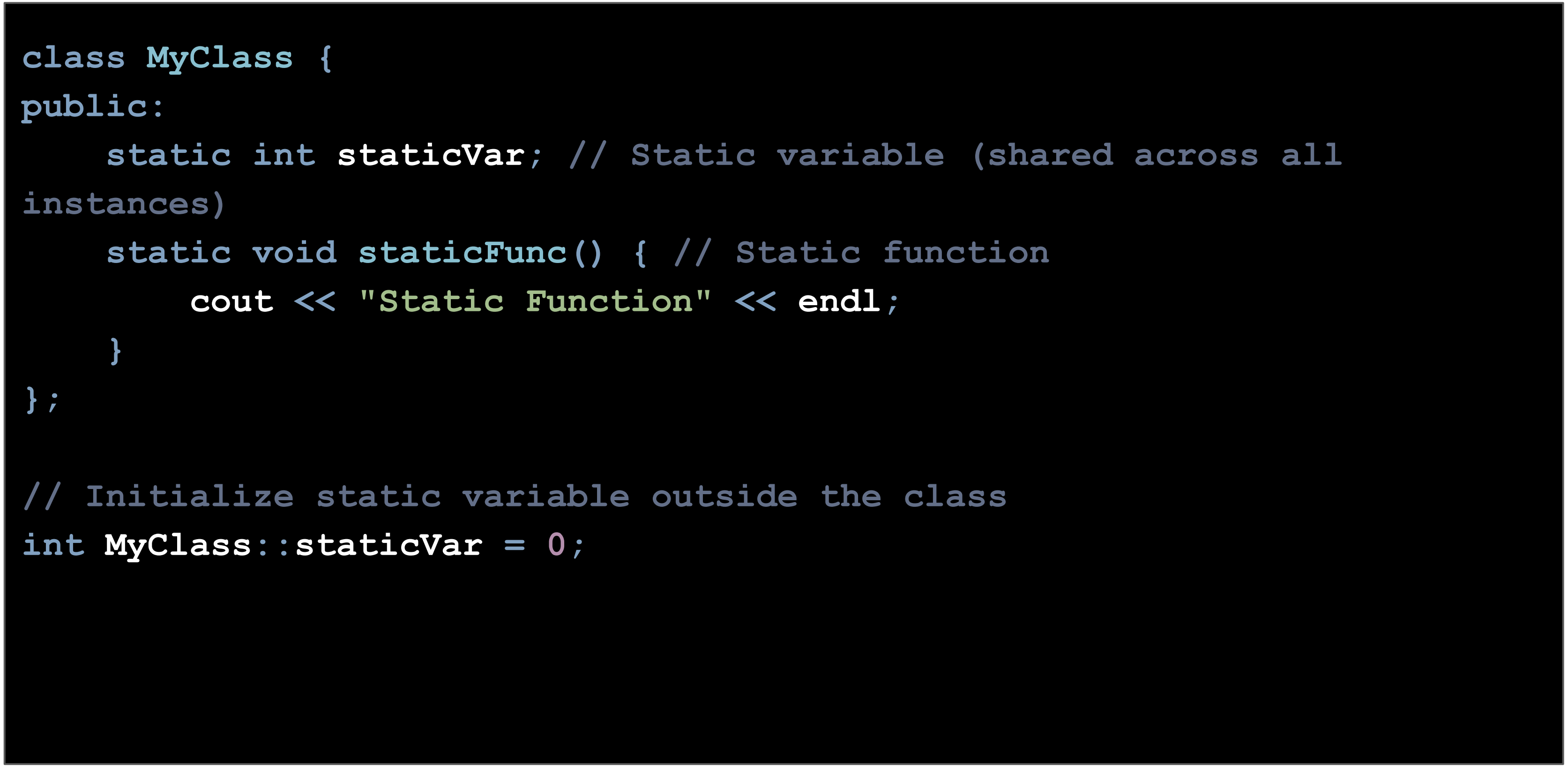
private

```
int main() {
    MyClass obj;
    // obj.privateVar = 10; // Error: privateVar is private
    // obj.privateFunc();    // Error: privateFunc is private
    obj.setPrivateVar(20); // Accessible via public function
    cout << obj.getPrivateVar() << endl; // Accessible via
public function
    return 0;
}
```

static

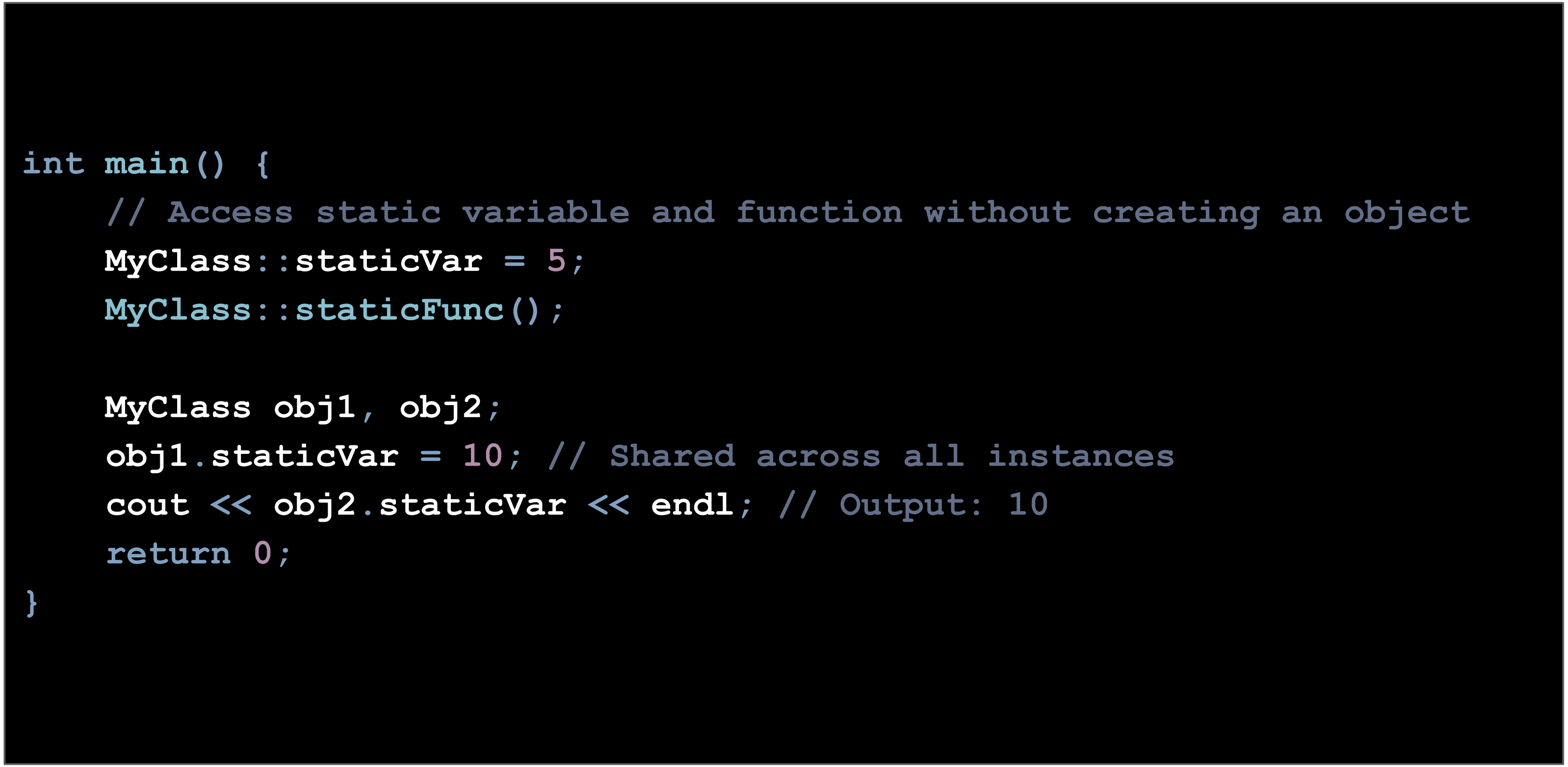
- **Purpose**: A static member **belongs to the class itself** rather than to any specific instance of the class. It is **shared across all instances of the class**.
- **Usage**:
  - For variables: Used to **maintain a single shared** value across all objects of the class.
  - For functions: Used to define functions that can be **called without creating an instance** of the class.

static



```
class MyClass {
public:
    static int staticVar; // Static variable (shared across all
instances)
    static void staticFunc() { // Static function
        cout << "Static Function" << endl;
    }
};

// Initialize static variable outside the class
int MyClass::staticVar = 0;
```

static



```cpp
int main() {
    // Access static variable and function without creating an object
    MyClass::staticVar = 5;
    MyClass::staticFunc();

    MyClass obj1, obj2;
    obj1.staticVar = 10; // Shared across all instances
    cout << obj2.staticVar << endl; // Output: 10
    return 0;
}
```

pointer

A **pointer** in C++ is a variable that stores the **memory address** of another variable.

```
data_type *pointer_name;
```

- The * (asterisk) indicates that the variable is a pointer.
- data_type specifies the type of data the pointer will point to.

pointer

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

int main() {
    int num = 10;
    int *ptr = &num; // Pointer stores the address of num

    cout << "Value of num: " << num << endl;
    cout << "Address of num: " << &num << endl;
    cout << "Pointer (ptr) stores address: " << ptr << endl;
    cout << "Value at pointer location (*ptr): " << *ptr << endl;

    return 0;
}
```

Dynamic Memory Allocation

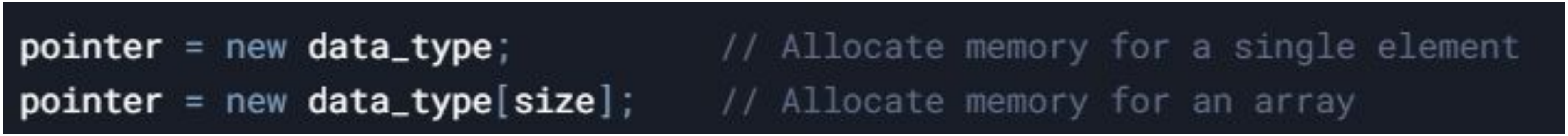
In C++, new and delete are used for **dynamic memory allocation**.

Dynamic Memory Allocation (new)

new Keyword:

- **Purpose**: Allocates memory dynamically.
- **Usage**: It *returns a pointer* to the allocated memory.

Syntax



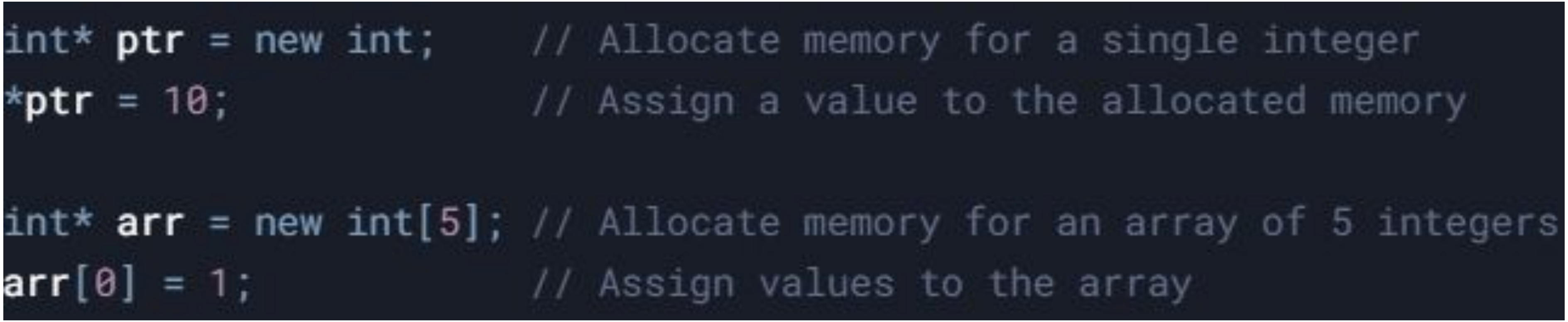
```
pointer = new data_type;          // Allocate memory for a single element
pointer = new data_type[size];    // Allocate memory for an array
```

Dynamic Memory Allocation (new)

new Keyword:

- **Purpose**: Allocates memory dynamically.
- **Usage**: It *returns a pointer* to the allocated memory.

Example



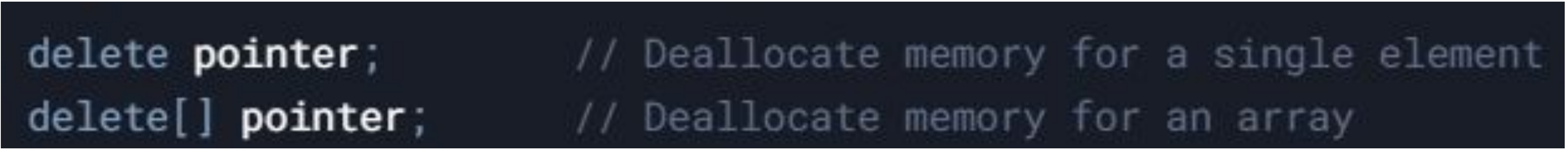
```
int* ptr = new int;      // Allocate memory for a single integer
*ptr = 10;               // Assign a value to the allocated memory

int* arr = new int[5];  // Allocate memory for an array of 5 integers
arr[0] = 1;              // Assign values to the array
```

Dynamic Memory Allocation (delete)

delete Keyword:

- **Purpose**: Deallocates memory that was previously allocated using new.
- **Usage**: Frees memory to avoid memory leaks.

Syntax



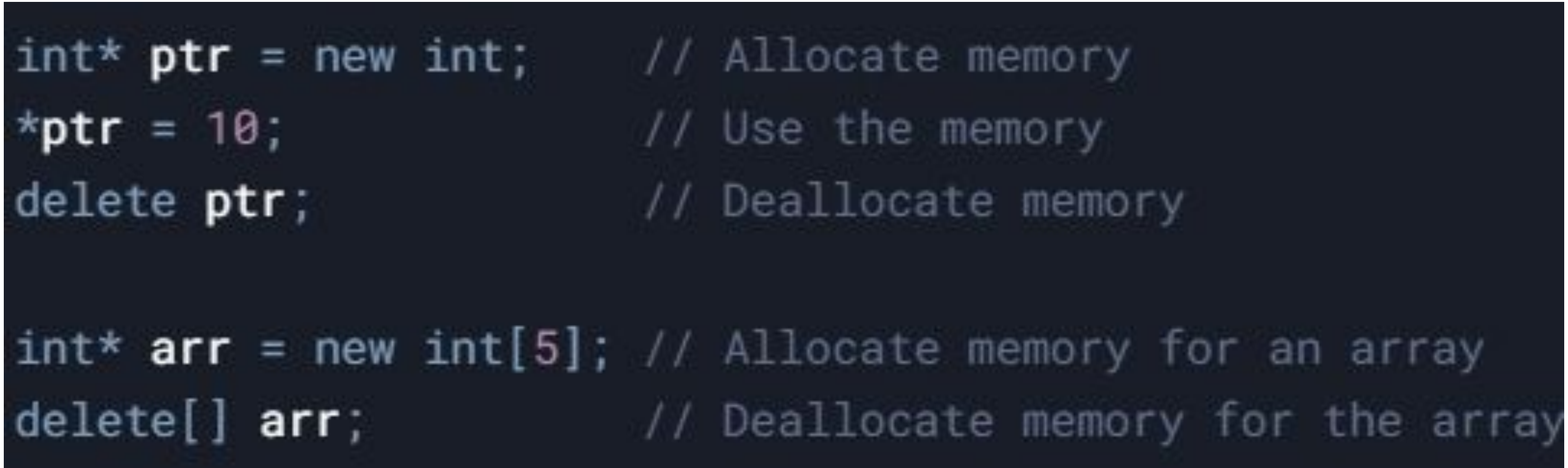
```
delete pointer;          // Deallocate memory for a single element
delete[] pointer;        // Deallocate memory for an array
```

Dynamic Memory Allocation (delete)

delete Keyword:

- **Purpose**: Deallocates memory that was previously allocated using new.
- **Usage**: Frees memory to avoid memory leaks.

Example



```
int* ptr = new int;    // Allocate memory
*ptr = 10;             // Use the memory
delete ptr;            // Deallocate memory

int* arr = new int[5]; // Allocate memory for an array
delete[] arr;          // Deallocate memory for the array
```