Advanced Programming Lab
CS6150

Anantha Padmanabha
ananthap@cse.iitm.ac.in

Meghana Nasre
meghana@cse.iitm.ac.in

August 2025

Compensation Lab

- 05-09-2025 (Friday) is a holiday

- Is 04-09-2025 (Thursday) ok for the alternative lab?
    - Finalized?s

Advanced Programming Lab
CS6150

Week 2

Class, Objects, Constructors, Destructors

(Slides Courtesy : Rupesh Nasre)

Abstraction

- Abstraction simplifies complexity.
  - When we drive a two-wheeler, we need not know how the engine operates.
  - We know to click gmail send button; we need not know how UDP packets are transmitted.

- Interface defines an abstraction.

- A Class is used to abstract / hide implementation details from thes user

Interface and Implementation

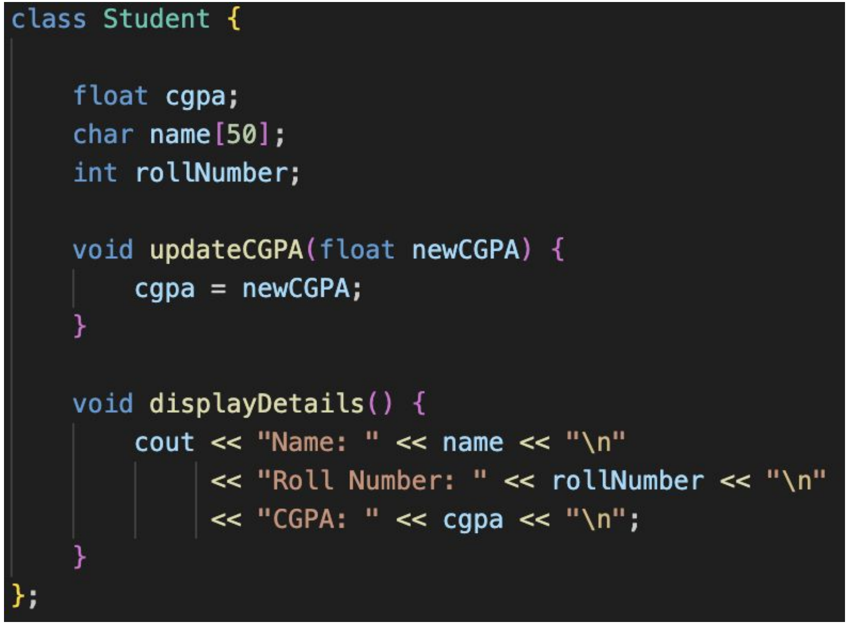
- C++ allows us to separate interface from the implementation.
  - Similar to declaration and definition.
- This helps in shipping the interface with compiled implementation as a library.
  - User would not have access to C++ source of the implementation.
- Interface is often part of the header files
- Implementation can be in .so or .a file, compiled from .cpp files.
  - e.g., <math.h> and libm.so

Class and Object

- **Class**: Can be potentially any Type
  - Contains data and functionalities

- **Object**: Individual instances of the Class / Type
  - Ex: Car tn07bw156;
    Student s;

- Each object has all the properties defined for its class.
  - It has all the corresponding data and functionalities.



```
class Student {

    float cgpa;
    char name[50];
    int rollNumber;

    void updateCGPA(float newCGPA) {
        cgpa = newCGPA;
    }

    void displayDetails() {
        cout << "Name: " << name << "\n"
             << "Roll Number: " << rollNumber << "\n"
             << "CGPA: " << cgpa << "\n";
    }
};
```

Class and Object

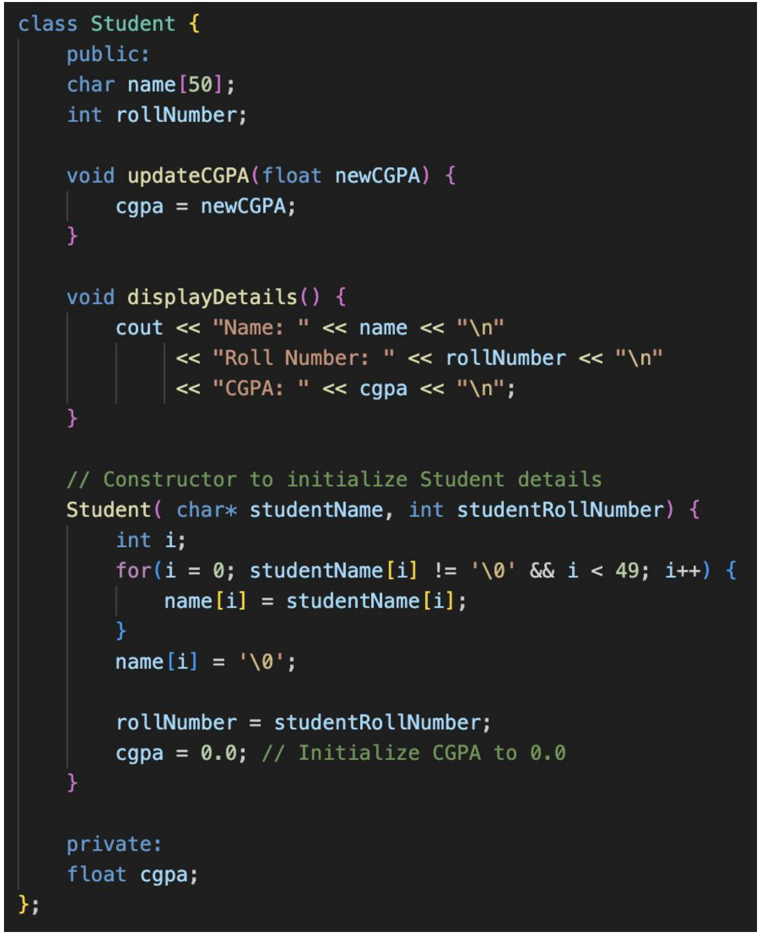
- **Class**: Can be potentially any Type
  - Contains data and functionalities

- **Object**: Individual instances of the Class / Type

  - Ex: Car tn07bw156;
    Student s;

- Each object has all the properties defined for its class.

  - It has all the corresponding data and functionalities.

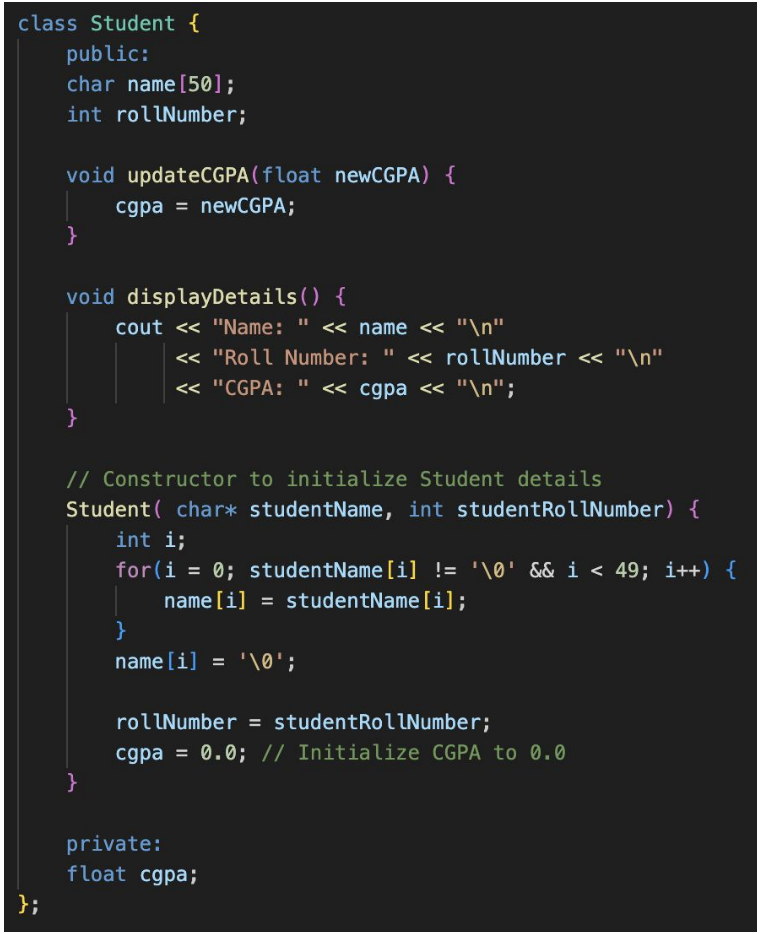


```
class Student {
    public:
    char name[50];
    int rollNumber;

    void updateCGPA(float newCGPA) {
        cgpa = newCGPA;
    }

    void displayDetails() {
        cout << "Name: " << name << "\n"
             << "Roll Number: " << rollNumber << "\n"
             << "CGPA: " << cgpa << "\n";
    }

    // Constructor to initialize Student details
    Student( char* studentName, int studentRollNumber) {
        int i;
        for(i = 0; studentName[i] != '\0' && i < 49; i++) {
            name[i] = studentName[i];
        }
        name[i] = '\0';

        rollNumber = studentRollNumber;
        cgpa = 0.0; // Initialize CGPA to 0.0
    }

    private:
    float cgpa;
};
```

Class and Object

- When we create objects of a class, we need to initialize an object with certain parameters.
  - Name and roll number
  - CGPA should be set to 0

- Constructors help us achieve this.

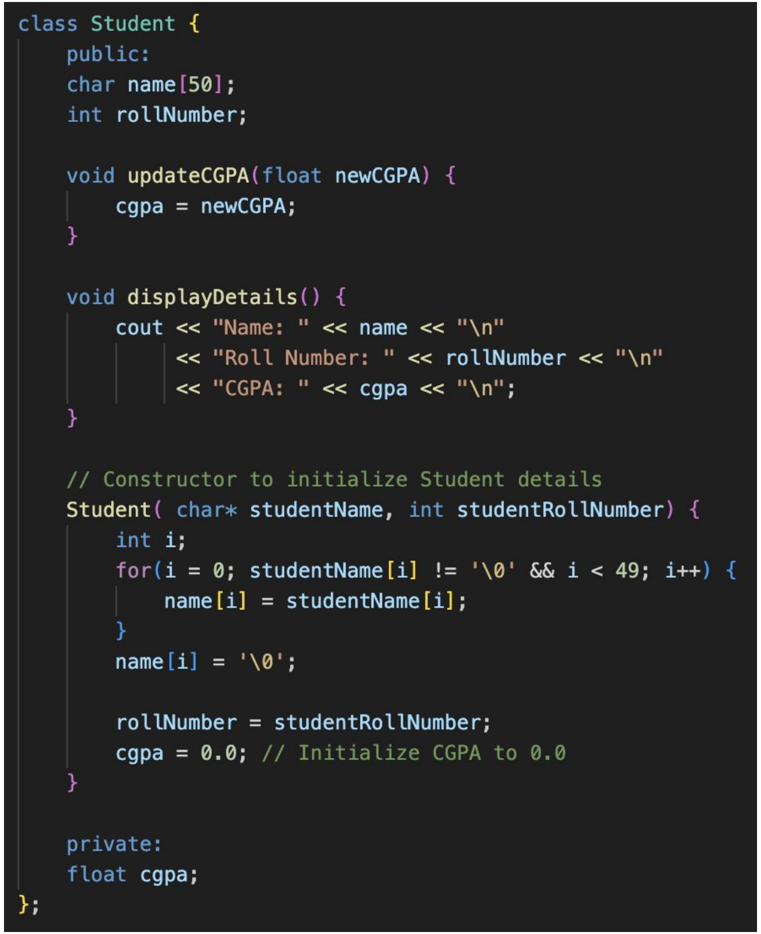


```
class Student {
    public:
    char name[50];
    int rollNumber;

    void updateCGPA(float newCGPA) {
        cgpa = newCGPA;
    }

    void displayDetails() {
        cout << "Name: " << name << "\n"
            << "Roll Number: " << rollNumber << "\n"
            << "CGPA: " << cgpa << "\n";
    }

    // Constructor to initialize Student details
    Student( char* studentName, int studentRollNumber) {
        int i;
        for(i = 0; studentName[i] != '\0' && i < 49; i++) {
            name[i] = studentName[i];
        }
        name[i] = '\0';

        rollNumber = studentRollNumber;
        cgpa = 0.0; // Initialize CGPA to 0.0
    }

    private:
    float cgpa;
};
```

Class and Object

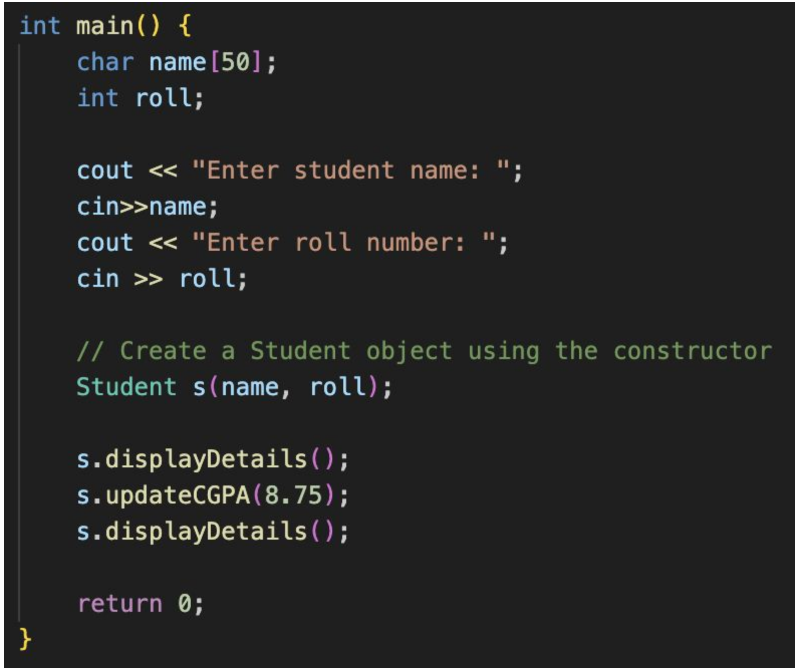
- A constructor is called when an object is created / instantiated.

- Constructor typically assigns initial values to fields and allocates resources.

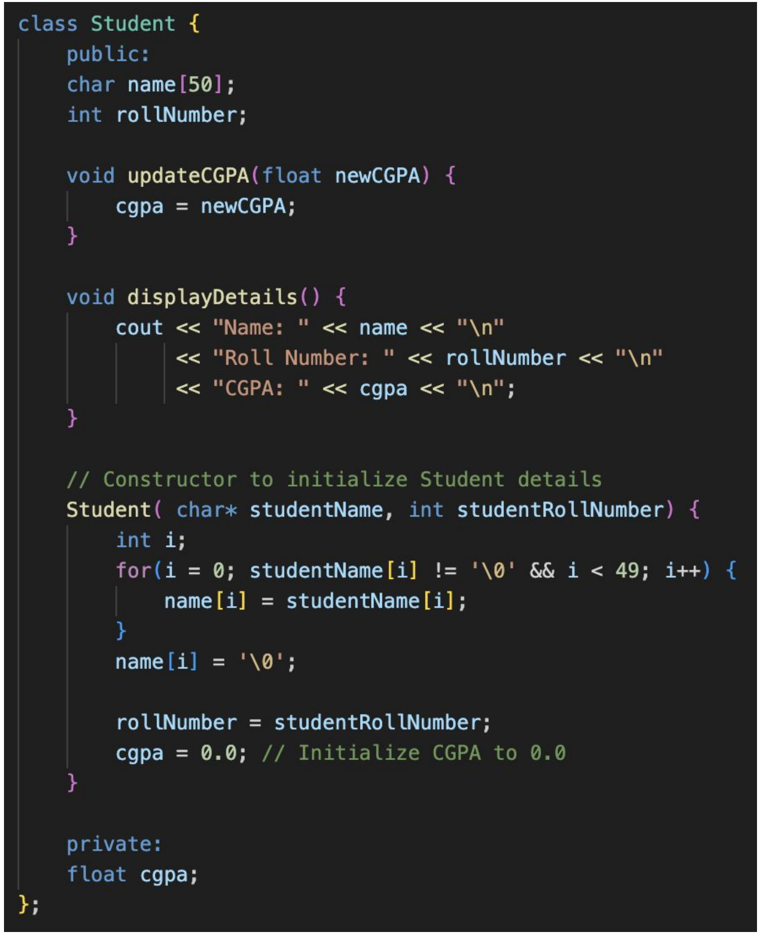


```
class Student {
    public:
    char name[50];
    int rollNumber;

    void updateCGPA(float newCGPA) {
        cgpa = newCGPA;
    }

    void displayDetails() {
        cout << "Name: " << name << "\n"
             << "Roll Number: " << rollNumber << "\n"
             << "CGPA: " << cgpa << "\n";
    }

    // Constructor to initialize Student details
    Student( char* studentName, int studentRollNumber) {
        int i;
        for(i = 0; studentName[i] != '\0' && i < 49; i++) {
            name[i] = studentName[i];
        }
        name[i] = '\0';

        rollNumber = studentRollNumber;
        cgpa = 0.0; // Initialize CGPA to 0.0
    }

    private:
    float cgpa;
};
```

Student Constructor



```cpp
int main() {
    char name[50];
    int roll;

    cout << "Enter student name: ";
    cin>>name;
    cout << "Enter roll number: ";
    cin >> roll;

    // Create a Student object using the constructor
    Student s(name, roll);

    s.displayDetails();
    s.updateCGPA(8.75);
    s.displayDetails();

    return 0;
}
```



```cpp
class Student {
    public:
    char name[50];
    int rollNumber;

    void updateCGPA(float newCGPA) {
        cgpa = newCGPA;
    }

    void displayDetails() {
        cout << "Name: " << name << "\n"
             << "Roll Number: " << rollNumber << "\n"
             << "CGPA: " << cgpa << "\n";
    }

    // Constructor to initialize Student details
    Student( char* studentName, int studentRollNumber) {
        int i;
        for(i = 0; studentName[i] != '\0' && i < 49; i++) {
            name[i] = studentName[i];
        }
        name[i] = '\0';

        rollNumber = studentRollNumber;
        cgpa = 0.0; // Initialize CGPA to 0.0
    }

    private:
    float cgpa;
};
```

Constructors

- If we do not define one, C++ provides a default (with zero arguments).
  - Student s; // okay: default constructor.
  - Student s(name, rollNo);   // compilation error.

- If we define one, C++ doesn't provide the default.
  - Student s(name, rollNo);   // okay: defined constructor.
  - Student s;   // compilation error.

- We can define multiple constructors, with different arguments (polymorphism).
  - Student s(name, rollNo);   // okay: defined.
  - Student s (name);   // okay: defined.

Destructor

- A destructor is helpful when some cleanup is required at the end of life of an object.
  - fopen – fclose
  - malloc – free

Class versus Object Variables

- Each object of a class has a different copy of its fields.
  - STUDENT a, b; a.name and b.name are different fields.
  - These are called object variables.
- If a field is defined as ***static***, it has a single copy across all instances (zero or more).
  - STUDENT a, b; a.studentCount and b.studentCount are same fields
  - These are called class variables.

Class versus Object Variables

- Static variables exist even when no objects of the class exist.
- A static method can be invoked even when no objects of the class exist.
- A static method can be called as Classname::fun(...).

  - It can as well be called using the object variable.
- A static method cannot use non-static variables (that is, cannot use object variables).

  - But a non-static method can use static as well as non-static variables.

Access Permissions

- C++ classes have access permissions
  - public, private, protected

- C++ enforces access checks.
  - Helps programmers avoid inadvertent or unintentional accesses.
  - Improve the overall software design.

Access Permissions

- A class has two types of members: fields and methods.
- We divide the world into three parts:
  - class, immediate children (inheritance), rest of the world

	public	protected	private
class	✓	✓	✓
children	✓	✓	×
rest	✓	×	×

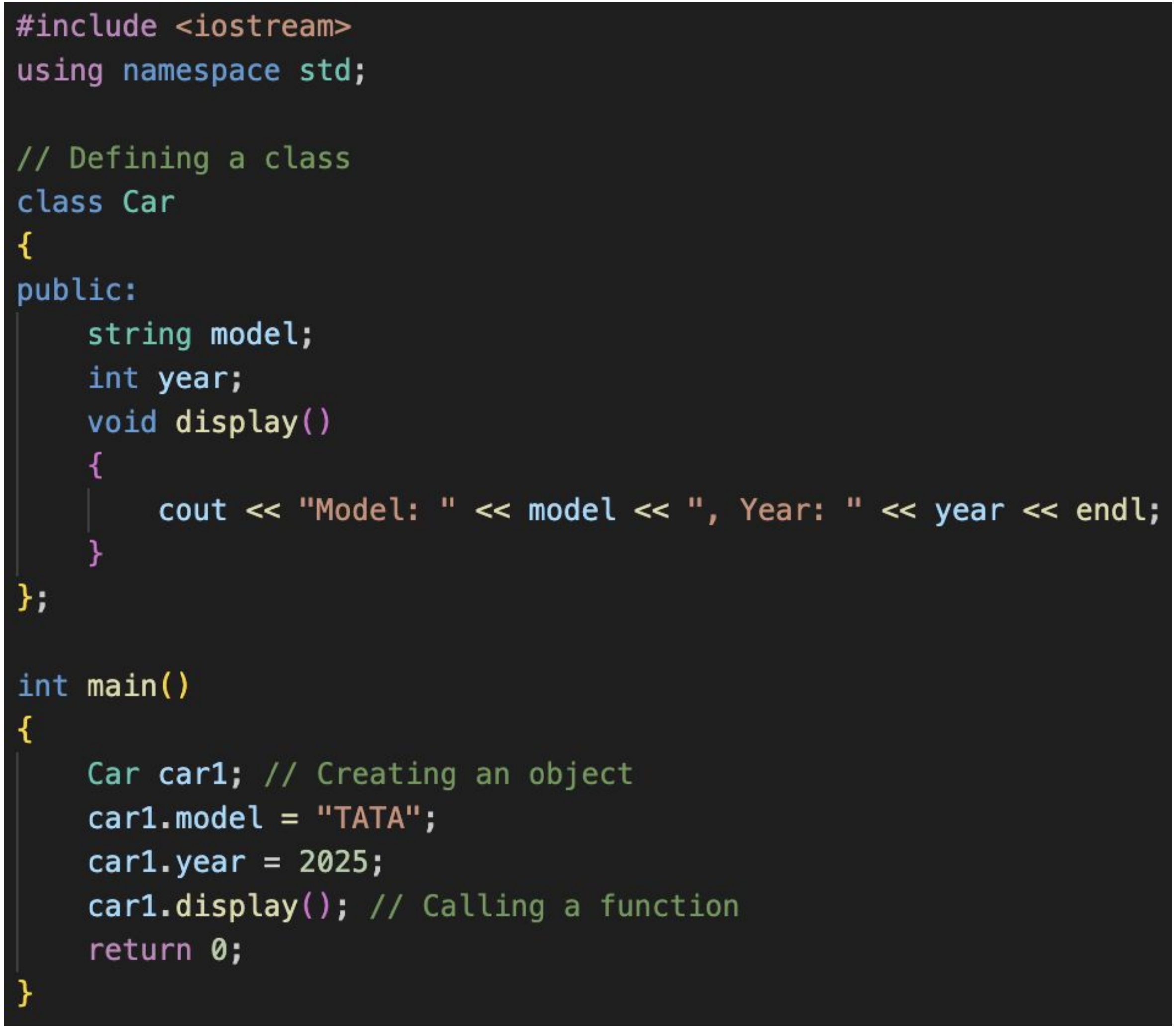
Advanced Programming Lab
CS6150

Week 2

Sample Programs

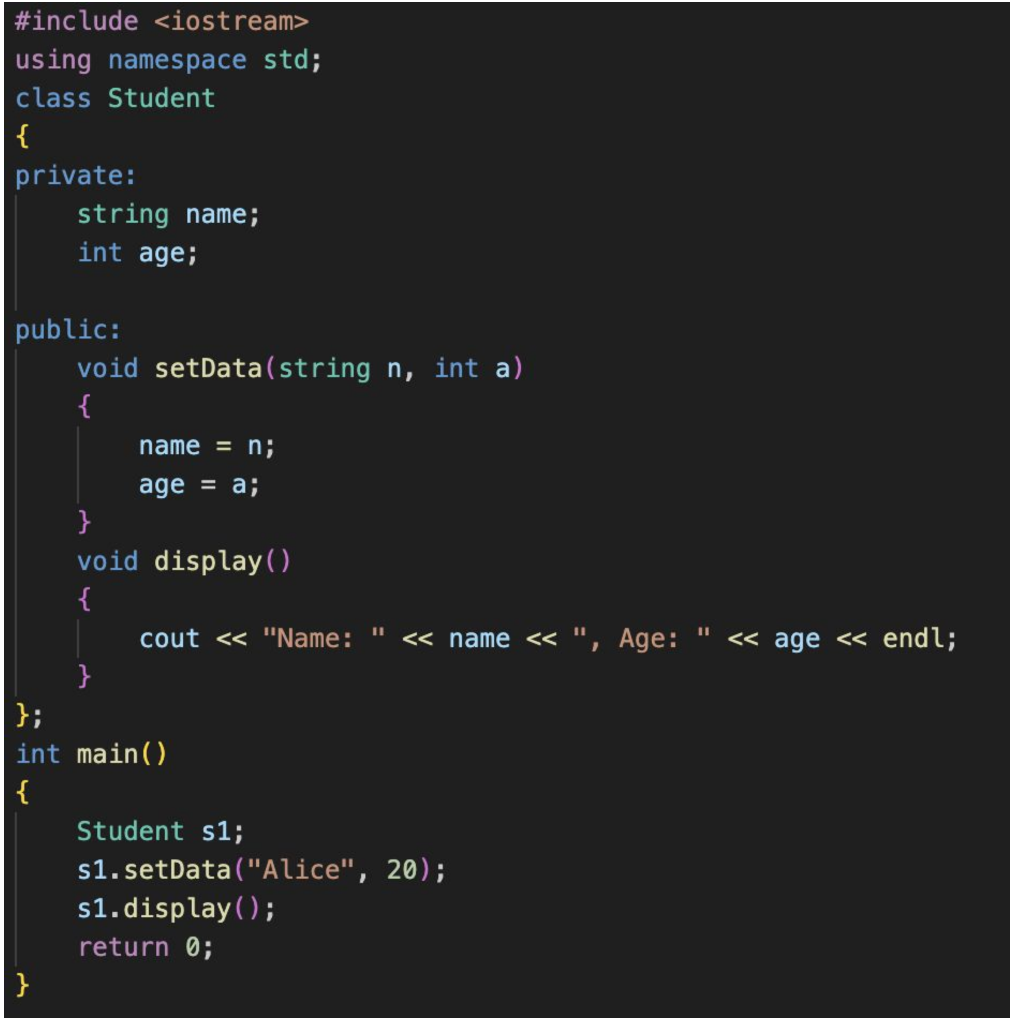
Code Courtesy :
Sirigineedi Dhanush Tata Phani Srikar and Dinesh Kumar S

Example 1



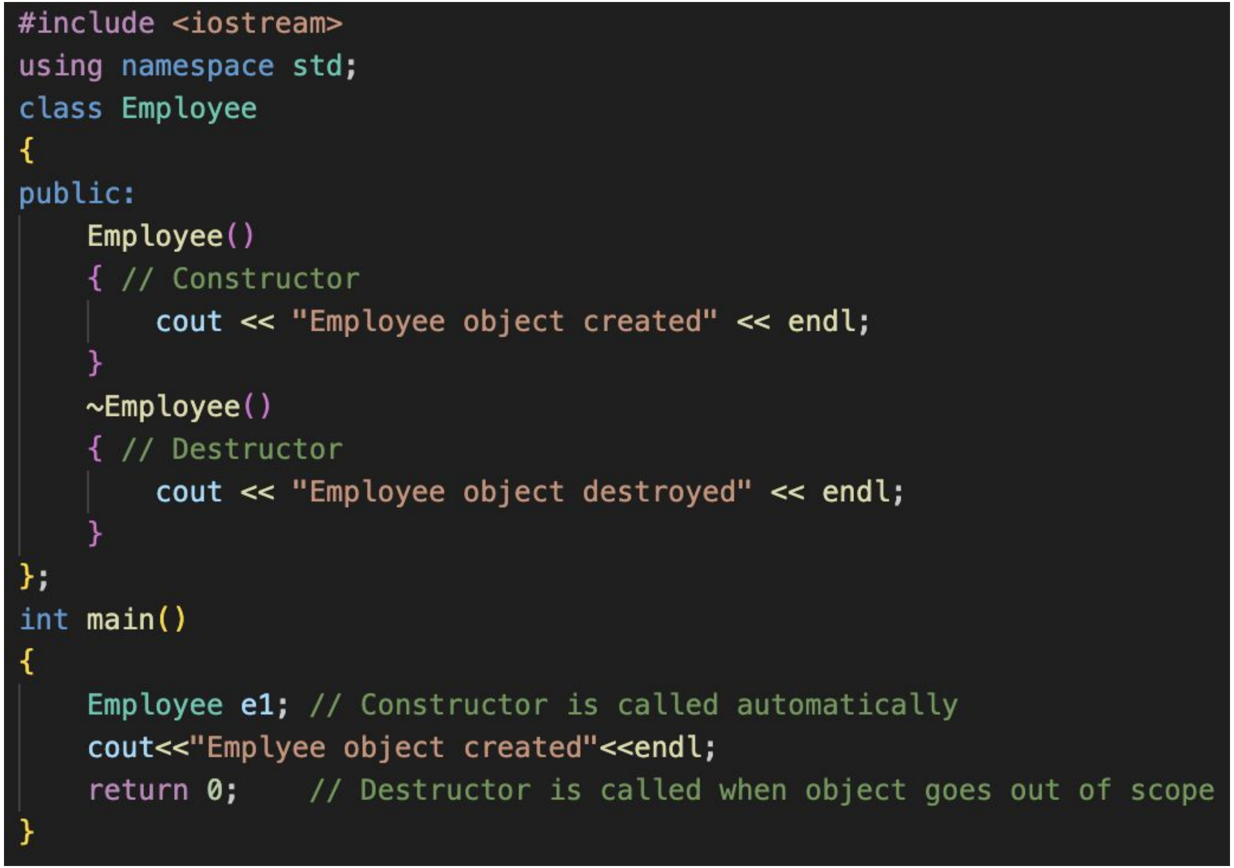
```cpp
#include <iostream>
using namespace std;

// Defining a class
class Car
{
public:
    string model;
    int year;
    void display()
    {
        cout << "Model: " << model << ", Year: " << year << endl;
    }
};

int main()
{
    Car car1; // Creating an object
    car1.model = "TATA";
    car1.year = 2025;
    car1.display(); // Calling a function
    return 0;
}
```

Example 2



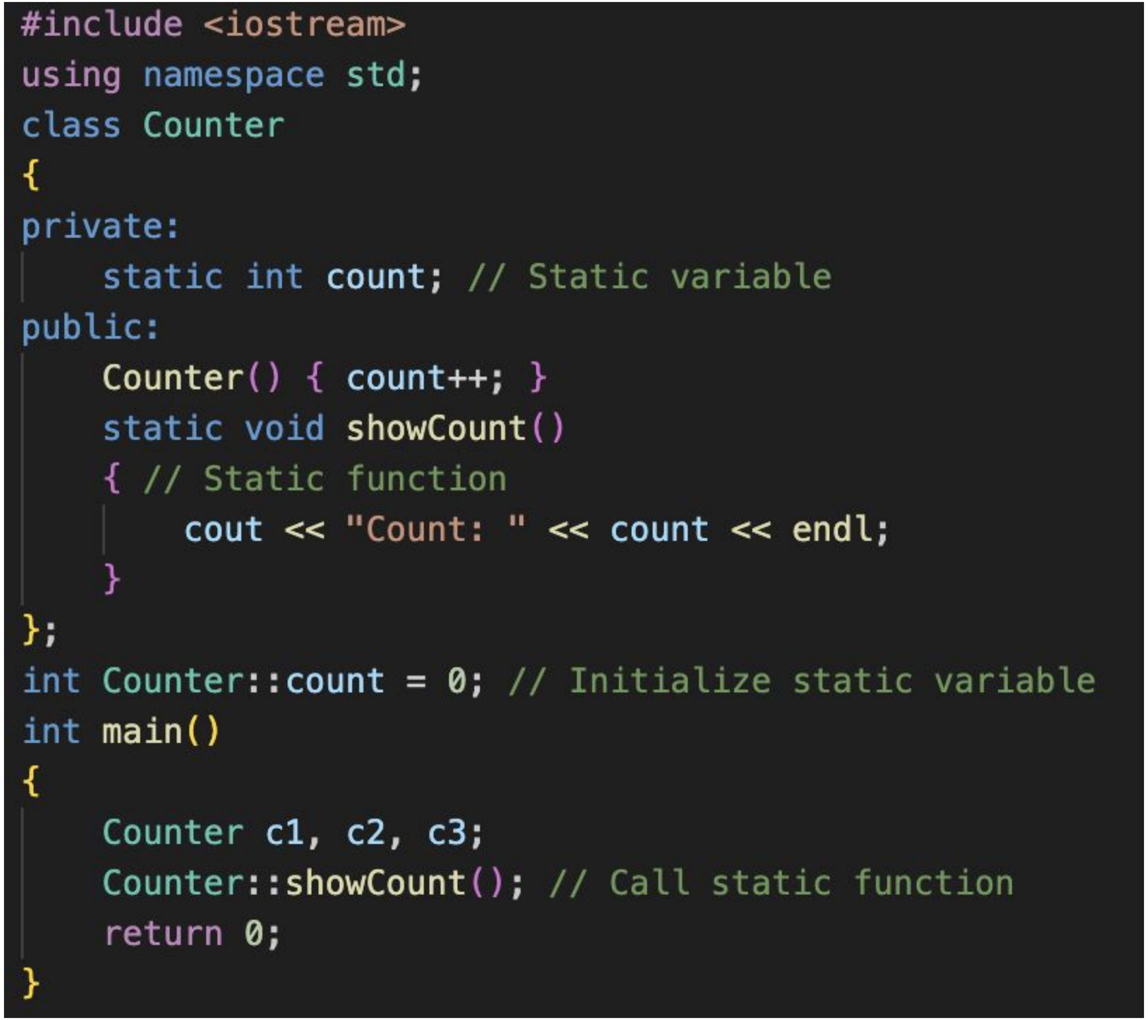
```cpp
#include <iostream>
using namespace std;
class Student
{
private:
    string name;
    int age;

public:
    void setData(string n, int a)
    {
        name = n;
        age = a;
    }
    void display()
    {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
int main()
{
    Student s1;
    s1.setData("Alice", 20);
    s1.display();
    return 0;
}
```

Example 3



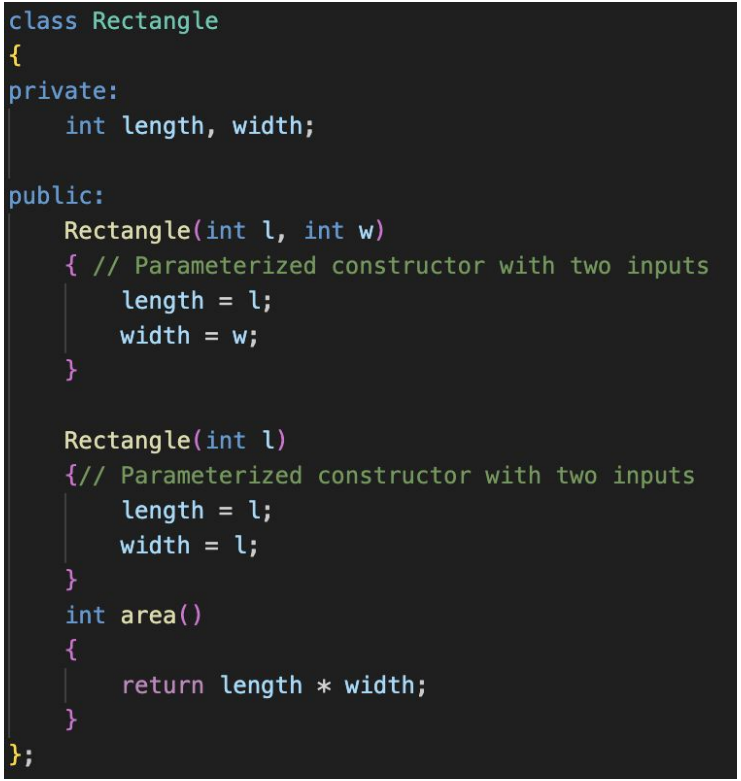
```
#include <iostream>
using namespace std;
class Employee
{
public:
    Employee()
    { // Constructor
        cout << "Employee object created" << endl;
    }
    ~Employee()
    { // Destructor
        cout << "Employee object destroyed" << endl;
    }
};
int main()
{
    Employee e1; // Constructor is called automatically
    cout<<"Emplyee object created"<<endl;
    return 0;    // Destructor is called when object goes out of scope
}
```

Example 4

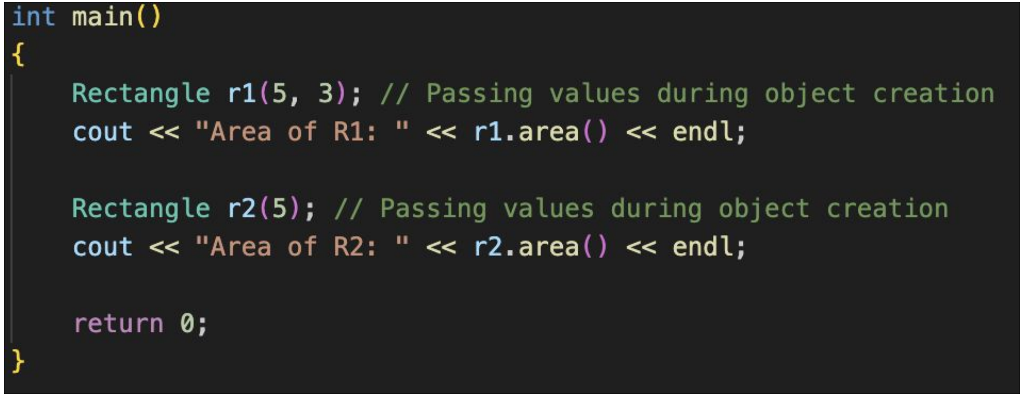


```cpp
#include <iostream>
using namespace std;
class Counter
{
private:
    static int count; // Static variable
public:
    Counter() { count++; }
    static void showCount()
    { // Static function
        cout << "Count: " << count << endl;
    }
};
int Counter::count = 0; // Initialize static variable
int main()
{
    Counter c1, c2, c3;
    Counter::showCount(); // Call static function
    return 0;
}
```

Example 5



```
class Rectangle
{
private:
    int length, width;

public:
    Rectangle(int l, int w)
    { // Parameterized constructor with two inputs
        length = l;
        width = w;
    }

    Rectangle(int l)
    {// Parameterized constructor with two inputs
        length = l;
        width = l;
    }
    int area()
    {
        return length * width;
    }
};
```



```
int main()
{
    Rectangle r1(5, 3); // Passing values during object creation
    cout << "Area of R1: " << r1.area() << endl;

    Rectangle r2(5); // Passing values during object creation
    cout << "Area of R2: " << r2.area() << endl;

    return 0;
}
```

See you in the lab on Friday

Try out examples

Practise problems will be available by tomorrow