 C++ 101 – Session 12 Notes

 Topic: Constructors in C++

What is a Constructor?

A **constructor** is a **special function** that is **automatically called** when an object of a class is **created**.

It is used to **initialize objects** assigning default or user-defined values to the class attributes as soon as the object is created.

🔷 Rules of Constructors

Here are the key rules of defining a constructor in C++:

Rule	Explanation
✅ Has the **same name** as the class	e.g., `Car()` is a constructor for class `Car`
✅ **No return type** (not even `void`)	It must not return anything
✅ Usually declared as **public**	So it can be called automatically during object creation
✅ Called **automatically** when an object is created	You don't call it manually

---

✅ Example: Default Constructor

```
class Car {
public:
    Car() {
        cout << "Default Constructor Called" << endl;
    }
};
```

Creating an object:

```
Car myCar;  // Constructor is automatically called
```

🔷 Constructor with Parameters

Constructors can also accept **parameters**, allowing you to set initial values at the time of object creation:

```
Car(string b, string m, string c, int y, float w, float p){
    brand = b;
    model = m;
    color = c;
    year = y;
    weight = w;
    price = p;
}
```

Usage:

```
Car myCar("Toyota", "Corolla", "Red", 2020, 1300.5, 20000.0);
```

🔷 Defining a Constructor Outside the Class

You can also write the constructor definition **outside** the class using the **scope resolution operator** `::`:

```
// Inside class
class Car {
public:
    Car(string b, string m, int y); // Declaration only
};

// Outside class
Car::Car(string b, string m, int y) {
    brand = b;
    model = m;
    year = y;
}
```

This helps keep your code cleaner and more organized.

🔷 Constructor Overloading

In C++, you can define **multiple constructors** in the same class as long as they have **different parameters**. This is called **constructor overloading**.

✅ Example:

```
Car() {
    // Default constructor
}

Car(string b, string m, int y Car(string b, string m, string c, int y, float w,
float p)

    // Constructor with 6 parameters
}
```

The compiler automatically chooses which constructor to use based on the arguments provided.

❓ Why Use Constructor Overloading?

- To allow **flexibility** when creating objects
- To provide both **default** and **custom** object initialization
- To **reduce repetitive code** when setting attribute values

---

✅ Code Used in Class

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

class Car {
    public:         // Access specifier
        string brand;
        string model;
        string color;
        int year;
```

```
        float weight;
        float price;

    Car() {
        cout << "Default Constructor Called" << endl;
        brand = "Unknown";
        model = "Unknown";
        color = "Unknown";
        year = 0;
        weight = 0.0;
        price = 0.0;
    }

    Car(string b, string m, string c, int y, float w, float p){
        cout << "Constructor Called" << endl;
        brand = b;
        model = m;
        color = c;
        year = y;
        weight = w;
        price = p;
    }

    void printDetails() {
        cout << "Brand: " << brand << endl;
        cout << "Model: " << model << endl;
        cout << "Color: " << color << endl;
        cout << "Year: " << year << endl;
        cout << "Weight: " << weight << " kg" << endl;
        cout << "Price: $" << price << endl;
    }
};

int main() {

    Car car1;
    car1.printDetails();

    cout << endl;

    Car car2("Toyota", "Corolla", "Red", 2020, 1300.5, 20000.0);
    car2.printDetails();

    return 0;
}
```

🎯 Assignment

Come prepared to **present a working demo** of a class and objects with:

- Attributes (variables)
- Methods (functions)
- At least one **constructor** (default or parameterized)
- Correct use of **object creation** and **printing details**

You'll explain your code and show the output in the next session.

🧠 Summary Table

Concept	Description
Constructor	Special function that initializes objects when they are created
Default Constructor	Takes no parameters, used to assign default values
Parameterized Constructor	Accepts values to initialize object attributes
Constructor Overloading	Using multiple constructors with different parameter sets
Scope Resolution (::)	Allows defining constructor outside the class
Assignment	Build your own class with constructors and present it