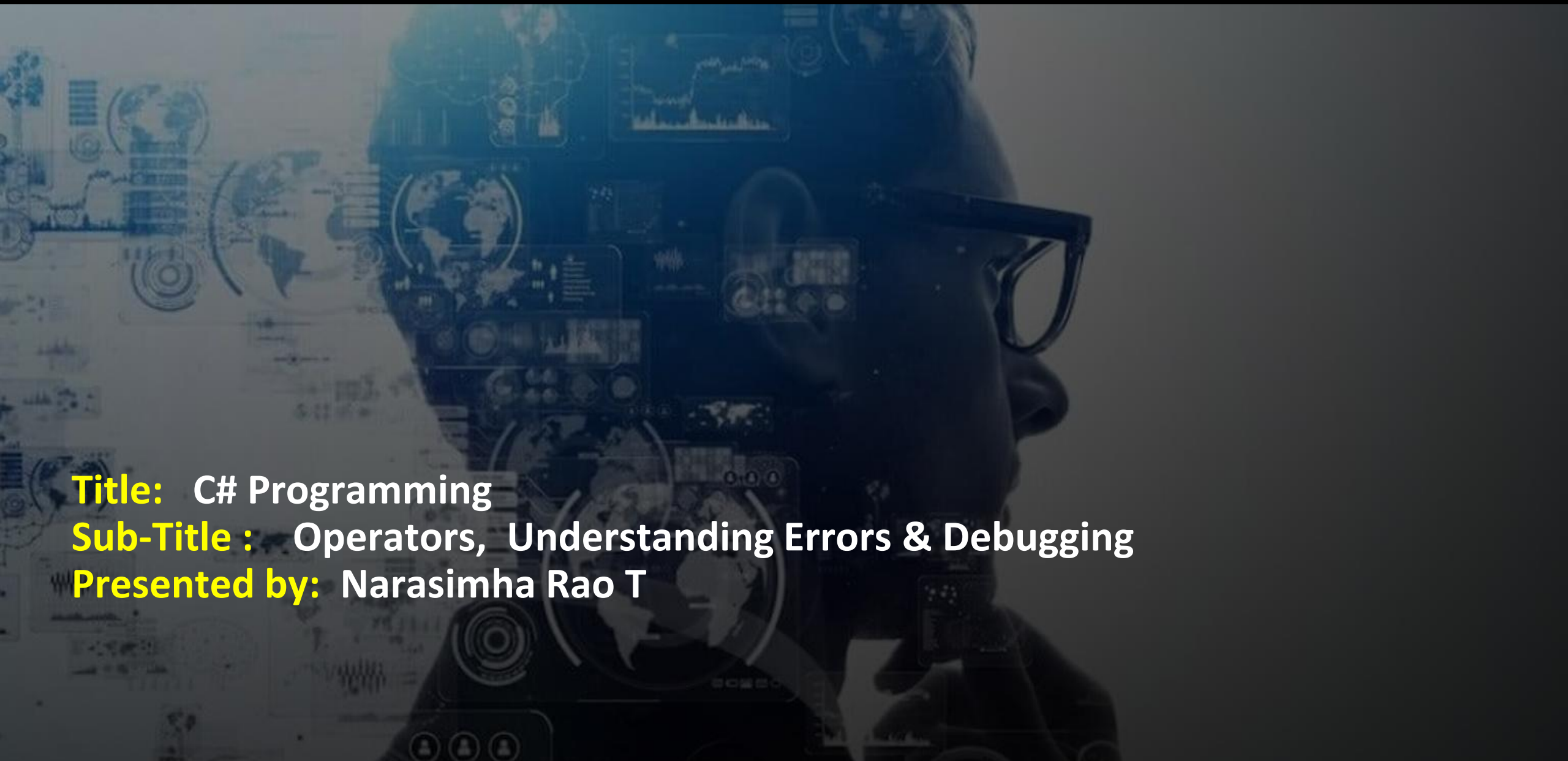
upGrad ENTERPRISE

.NET FSD Bootcamp

11th July, 2025

upGrad ENTERPRISE



Title: C# Programming

Sub-Title : Operators, Understanding Errors & Debugging

Presented by: Narasimha Rao T

upGrad ENTERPRISE

C# Language : Operators, Error Handling & Debugging

By

Narasimha Rao T

Microsoft.Net FSD Trainer

Professional Development Trainer

Day-3

Index

1. Arithmetic, Relational, Logical & Assignment Operators

Arithmetic Operators

Operator	Use	Example
`+`	Addition	`3 + 2` → `5`
`-`	Subtraction	`3 - 2` → `1`
`*`	Multiply	`3 * 2` → `6`
`/`	Division	`7 / 2` → `3` (int)
`%`	Modulo	`7 % 2` → `1`

Relational (Comparison)

Operator	Meaning	Example
`==`	Equal to	`a == b`
`!=`	Not equal	`a != b`
`>`	Greater than	`a > b`
`<`	Less than	`a < b`
`>=`	Greater/equal	`a >= b`
`<=`	Less/equal	`a <= b`

Logical Operators

Operator	Meaning	Example				
`&&`	AND	`a > 0 && b < 10`				
`		`	OR	`a > 0		b < 10`
`!`	NOT	`!(a > 0)`				

Assignment Operators

```
int x = 5;
x += 3; // same as x = x + 3
```

2. Ternary & Compound Assignment

Ternary Operator

```
int age = 20;
string result = age >= 18 ? "Adult" : "Minor";
```

Structure: `condition ? trueValue : falseValue;`

Compound Assignment

Operator	Equivalent To
`+=`	`x = x + y`
`-=`	`x = x - y`
`*=`	`x = x * y`
`/=`	`x = x / y`

3. Operator Precedence

Precedence Order:

1. Parentheses `()`
2. Unary: `!`, `++`, `--`
3. Multiplicative: `*`, `/`, `%`
4. Additive: `+`, `-`
5. Relational: `<`, `>`, `==`, `!=`
6. Logical: `&&`, `||`
7. Assignment: `=`, `+=`, `-=`

```
int result = 2 + 3 * 4; // 2 + (3*4) = 14
int another = (2 + 3) * 4; // 20
```

4. String Operations

Concatenation

```
string fullName = "John" + " " + "Doe";
```

Interpolation

```
string name = "Alice";
int age = 25;
Console.WriteLine($"Name: {name}, Age: {age}");
```

5. Basic Error Handling (try-catch)

Syntax

```
try
{
    int x = int.Parse("abc");
}
catch (FormatException ex)
{
    Console.WriteLine("Invalid number format.");
}
```

- Prevents crashes from runtime errors
- `catch` can handle specific or general exceptions

6. Introduction to Debugging & Breakpoints

Debugging Basics:

- Set **breakpoints** to pause execution
- Watch variables and step through logic

Types of Steps:

- **Step Over (F10):** Skip into next line
- **Step Into (F11):** Go inside called method
- **Step Out (Shift+F11):** Exit current method

7. Demo: VS/VS Code Debugger

Steps:

1. Open a project in VS or VS Code
2. Set a **breakpoint** on a line
3. Press `F5` to **start debugging**
4. Use `F10/F11/Shift+F11` to control flow
5. Inspect **locals**, **watch**, and **call stack**

Watch Window:

- Add variables to monitor changes
- Hover to inspect values inline

8. Predict the Output Challenge

Present the following to the class:

```
int a = 10, b = 3;
int result = a / b * b;
Console.WriteLine(result); // ?
```

```
bool x = true;
bool y = false;
Console.WriteLine(x || y && false); // ?
```

9. Quiz: Spot the Operator Bug

Example Snippets:

```
int a = 10;
if (a = 5) //  wrong
```

```
string name = null;
if (name.Length > 0) // Wrong
```

```
int x = 5;
Console.WriteLine("Value is " + x - 1); //
```

10. Wrap-up Summary

Concept	You Learned
Operators	Arithmetic, relational, logical
Assignment & Ternary	Compound logic & decisions
Precedence	Order of execution
Strings	Combine and format data
Error Handling	`try-catch` structure
Debugging	Step through and fix logic

Q & A