TASK 6:

```
using System;

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        int marks;
        int total;

        Console.Write("Enter marks: ");
        bool ok1 = int.TryParse(Console.ReadLine(), out marks);

        Console.Write("Enter total: ");
        bool ok2 = int.TryParse(Console.ReadLine(), out total);

        if (!ok1 || !ok2)
        {
            Console.WriteLine("Invalid input. Please enter integers.");
            return;
        }

        // BREAKPOINT 1: Set here before calculation

        double percentage = marks / total * 100;   // Problem happens here

        // BREAKPOINT 2: Set here after calculation

        Console.WriteLine("Percentage = " + percentage);
    }
}
```

The error occurs because int division does not accept float characters. Hence the output will be 0.

To fix this we use double, so float values are accepted.

```
using System;

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        int marks;
        int total;

        Console.Write("Enter marks: ");
        bool ok1 = int.TryParse(Console.ReadLine(), out marks);

        Console.Write("Enter total: ");
        bool ok2 = int.TryParse(Console.ReadLine(), out total);

        if (!ok1 || !ok2)
        {
            Console.WriteLine("Invalid input.");
            return;
        }

        // Breakpoint 1 here

        double percentage = (double)marks / total * 100;

        // Breakpoint 2 here

        Console.WriteLine("Percentage = " + percentage);
    }
}
```

TASK 7:

Topic 1: How Constructors Help in Software Development

1. Introduction

In object-oriented programming (OOP), a **constructor** is a special method that is automatically executed when an object is created. It plays a major role in object initialization, code reliability, and maintainability, ensuring that objects begin their life in a valid, predictable state.

2. How Constructors Contribute to Software Development

a) Object Initialization

Constructors allow developers to set initial values for object attributes.
This ensures that every object starts with meaningful and correct data.

b) Code Reliability

Constructors prevent uninitialized or invalid objects.
By enforcing the necessary values during object creation, they reduce runtime errors.

c) Maintainability

Constructors centralize the initialization logic.
If the initialization process needs to change, you modify only the constructor instead of updating multiple parts of the code.

This improves code readability, simplifies debugging, and reduces duplication.

3. Real-World Use Cases of Constructors

Example 1: Initializing Configuration Settings

Applications often load database settings, API keys, or file paths during startup.

```
public class AppConfig {
   public string ApiUrl { get; }
   public string ApiKey { get; }
```

```
    public AppConfig(string apiUrl, string apiKey) {
        ApiUrl = apiUrl;
        ApiKey = apiKey;
    }
}
```

Example 2: Creating Game Characters

Games use constructors to ensure every character starts with default health, level, and abilities.

```
public class Player {
    public int Health;
    public int Level;

    public Player() {
        Health = 100;
        Level = 1;
    }
}
```

Example 3: Initializing Bank Accounts

Banking applications require important data such as account number, balance, and user identity at the moment of creation.

```
public class BankAccount {
    public string AccountNumber;
    public double Balance;
```

```
    public BankAccount(string accNo, double balance) {
        AccountNumber = accNo;
        Balance = balance;
    }
}
```

Topic 2: Research on an OOP Principle — Encapsulation

1. Introduction to Encapsulation

Encapsulation is the OOP principle that hides internal data and exposes only necessary methods to interact with it. It protects the data from unauthorized access or accidental modification.

Encapsulation is typically achieved using:

- **Private fields**
- **Public getters/setters**

2. Importance of Encapsulation

✓ Protects data

Prevents invalid or harmful changes to objects.

✓ Increases maintainability

Changes to the internal implementation do not affect other parts of the program.

✓ Improves readability

Clear separation between internal logic and external interaction.

Class

A class is a blueprint for creating objects. It defines attributes (variables) and behaviours (methods).

Object

An object is an instance of a class. It represents a real entity created based on the class structure.