---

1. Introduction to Problem Solving (C# Context)

1.1 What is Problem Solving in Programming?

Problem solving in programming is the process of:

- **Understanding** a real-world problem,
- **Designing** an efficient logical approach (algorithm),
- **Translating** it into a working program (code),
- **Testing and validating** the solution.

 In **C#**, it means thinking clearly about:

- The data you have (inputs)
- The results you want (outputs)
- The process (steps/logic) to transform inputs into outputs.

1.2 Importance of Algorithms

An **algorithm** is a step-by-step method for solving a problem.

 Why are algorithms important?

- **Efficiency**: A bad algorithm can make a program slow, even if the code is bug-free.
- **Reusability**: Good algorithms can be used across different programs.
- **Scalability**: Helps your program handle large inputs gracefully.

 Example: Algorithm for finding the maximum of two numbers

```
// C# Example
int FindMaximum(int num1, int num2)
{
    if (num1 > num2)
        return num1;
    else
        return num2;
}
```

1.3 Characteristics of a Good Algorithm

A **good algorithm** typically has:

- **Correctness**: It should solve the problem properly.
- **Efficiency**: Minimal time and space usage.
- **Finiteness**: It should complete in a finite number of steps.
- **Clarity**: Easy to understand and implement.
- **Generality**: Works for all valid inputs, not just a few cases.

 Good C# coding practices help in achieving these goals.

1.4 Problem-Solving Strategies

Here are **popular strategies** you'll apply while coding in C#: | Strategy | Description | C# Example | |:---------|:------------|:-----------| | **Understanding the problem** | Break down and identify exactly what is needed | Reading the problem carefully before coding | | **Divide and conquer** | Break into smaller parts, solve each, then combine | Methods/functions to solve sub-problems | | **Pattern recognition** | Recognize familiar patterns in the problem | Identifying loops or conditional structures | | **Simplify and generalize** | Solve a simpler version first, then expand | Test with simple inputs | | **Trial and Error** | Try different approaches if stuck | Debugging and running multiple versions of code |

1.5 Example: Solving a Simple Problem (Step-by-Step)

Problem:
Write a program to calculate the sum of two numbers entered by the user.

Step 1: Understand Inputs/Outputs

- Input: Two integers
- Output: Sum of the two integers

Step 2: Write Algorithm (in English)

1. Ask the user for the first number.
2. Ask the user for the second number.
3. Add both numbers.
4. Display the result.

Step 3: Write Pseudocode

```
START
Input number1
Input number2
sum = number1 + number2
Display sum
END
```

Step 4: Implement in C#

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Enter the first number:");
        int number1 = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Enter the second number:");
        int number2 = Convert.ToInt32(Console.ReadLine());

        int sum = number1 + number2;
```

```
        Console.WriteLine("The sum is: " + sum);
    }
}
```

 You solved a problem by moving systematically:

- Understand Plan Code Test.

Quick Summary for Part 1:

- Think before you code.
- Write the steps clearly.
- Choose the right strategy.
- Implement clean, readable C# code.
- Always test with different inputs.
