Part01

- **1) Problem: Add both single-line and multi-line comments in the following code segment explaining its purpose:**
  int x = 10;
  int y = 20;
  int sum = x + y;
  Console.WriteLine(sum);

  ```
  // Declare an integer variable x and assign it the value 10
  int x = 10;

  //  Declare an integer variable y and assign it the value 20
  int y = 20;

  // Calculate the sum of x and y and store the result in sum
  int sum = x + y;

  // Print the value of sum to the console
  Console.WriteLine(sum);
  ```

- **2) Question: What is the shortcut to comment and uncomment a selected block of code in Visual Studio?**

  Comment: Ctrl + K   Ctrl + C
  Uncomment: Ctrl + K   Ctrl + U

- **3) Problem: Identify and fix the errors in this code snippet:**
  int x = "10";
  console.WriteLine(x + y);

  ```
  int x = 10;          // Declare an integer variable x
  int y = 5;           // Declare the variable y
  Console.WriteLine(x + y);  // Print the sum of x and y
  ```

- **4) Question: Explain the difference between a runtime error and a logical error with examples.**

  runtime error: The program stops during execution

  ```
  int x = 10;
  int y = 0;
  int result = x / y; // Runtime error: division by zero
  ```

  Logical error: The program runs but the output is incorrect.

  ```
  int x = 10;
  int y = 20;
  int sum = x - y; // Logical error: wrong operation
  Console.WriteLine(sum);
  ```

- **5) Problem: Declare variables using proper naming conventions to store:**
  Your full name.
  Your age.
  Your monthly salary.
  Whether you are a student.

  ```
  string fullName = "Ahmed Hussein";   // Stores the full name
  int age = 21;                        // Stores the age
  double monthlySalary = 5000.50;      // Stores the monthly salary
  bool isStudent = true;               // Indicates whether the person is a student
  ```

- **6) Question: Why is it important to follow naming conventions such as PascalCase in C#?**

  Following naming conventions like PascalCase makes code clear, easy to read, and professional.

- **7) Problem: Write a program to demonstrate that changing the value of a reference type affects all references pointing to that object.**

  If you have an object of a reference type (like a class or an array), and you have multiple variables pointing to the same object,
  changing the object through any of the variables will affect all the other variables.

  In other words, they all refer to the same memory location, so any change is seen by all.

```
internal class Point
{
   public int x;
   public int y;
}
static void Main()
{
   Point p1 = new Point();

   Point p2 = new Point();

   p2 = p1;

   Console.WriteLine(p1.x); //0
   p1.x = 19;
   Console.WriteLine(p2.x); //19
}
```

- **8) Question: Explain the difference between value types and reference types in terms of memory allocation.**

  Value Types (like int, double, bool, struct):
  Stored directly in the Stack.
  Each variable has its own independent copy.
  Copying a variable does not affect the original.

  Reference Types (like class, array, string):
  The variable stores the address of the object in the Heap, while the reference itself is in the Stack.
  Copying the variable makes another reference to the same object.
  Changing the object through any reference affects all references.

- **9) Problem: Create a program that calculates the following using variables x = 15 and y = 4: Sum o Difference o Product o Division result o Remainder**

```
 int x = 15;
 int y = 4;

//Sum
 int sum = x + y;

//Difference
int difference = x - y;

//Product
int product = x * y;

//Division result
int division = x / y;

//Remainder
int remainder = x % y;

//output
 Console.WriteLine("sum = " + sum);
 Console.WriteLine("difference = " + difference);
 Console.WriteLine("Product = " + product);
 Console.WriteLine("division = " + division);
 Console.WriteLine("remainder = " + remainder);
```

- **10) Question: What will be the output of the following code? Explain why:**
  int a = 2, b = 7;
  Console.WriteLine(a % b);

  The output is 2   because the a less than the b

- **11) Problem: Write a program that checks if a given number is both:**
  o Greater than 10.
  o Even.

```
int num;
Console.WriteLine("enter the number");
num=int.Parse(Console.ReadLine());
if(num>10 && num%2==0)
{
    Console.WriteLine(num + " greater than 10 and even.");
}
else
{
    Console.WriteLine(num +" not satisfy both conditions.");
}
```

- **12) Question: How does the && (logical AND) operator differ from the & (bitwise AND) operator?**

  && is a logical AND operator that works on booleans and stops checking if the first condition is false.

  & is a bitwise AND operator that works on integers (bits) and checks all values without stopping.

- **13) Problem: Implement a program that takes a double input from the user and casts it to an int. Use both implicit and explicit casting, then print the results.**

```
         Console.WriteLine("enter a double number: ");
        double num=double.Parse(Console.ReadLine());

        //explicit
        int n1;
        checked
        {
            n1 = (int)num;
        }

        Console.WriteLine(n1);

        //implicit
        int i = 10;
        double n2 = i;
        Console.WriteLine(n2);
```

- **14) Question: Why is explicit casting required when converting a double to an int?**

  Explicit casting is required when converting a double to an int because the conversion may lose the fractional part, so the compiler needs a clear instruction to perform the conversion.

- **15) Question: What exception might occur if the input is invalid and how can you handle it**

  If the input is invalid, a Format Exception may occur.
  It can be handled using try-catch to prevent the program from crashing.

- **16) Problem: Write a program that demonstrates the difference between prefix and postfix increment operators using a variable x.**

```
int x = 5;
Console.WriteLine(x++); //print 5   and store 6
Console.WriteLine(++x); //print 7    and store 7
```

- **17) Problem: Write a program that: (G01 Bonus, G02 mandatory)**
  Prompts the user for their age as a string.
  Converts the string to an integer using Parse
  Checks if the age is valid (e.g., greater than 0).

```
class Program
{
    static void Main()
    {
        Console.WriteLine("enter your age");
        string ageInput=Console.ReadLine();

        int age=int.Parse(ageInput);

        if(age > 0)
        {
            Console.WriteLine("your age is valid " + age);
        }
        else
        {
            Console.WriteLine("invalid age. age must be greater than 0. ");
        }
    }
}
```

- **18) Question: Given the code below, what is the value of x after execution? Explain why**
  int x = 5; int y = ++x + x++;

  x=5

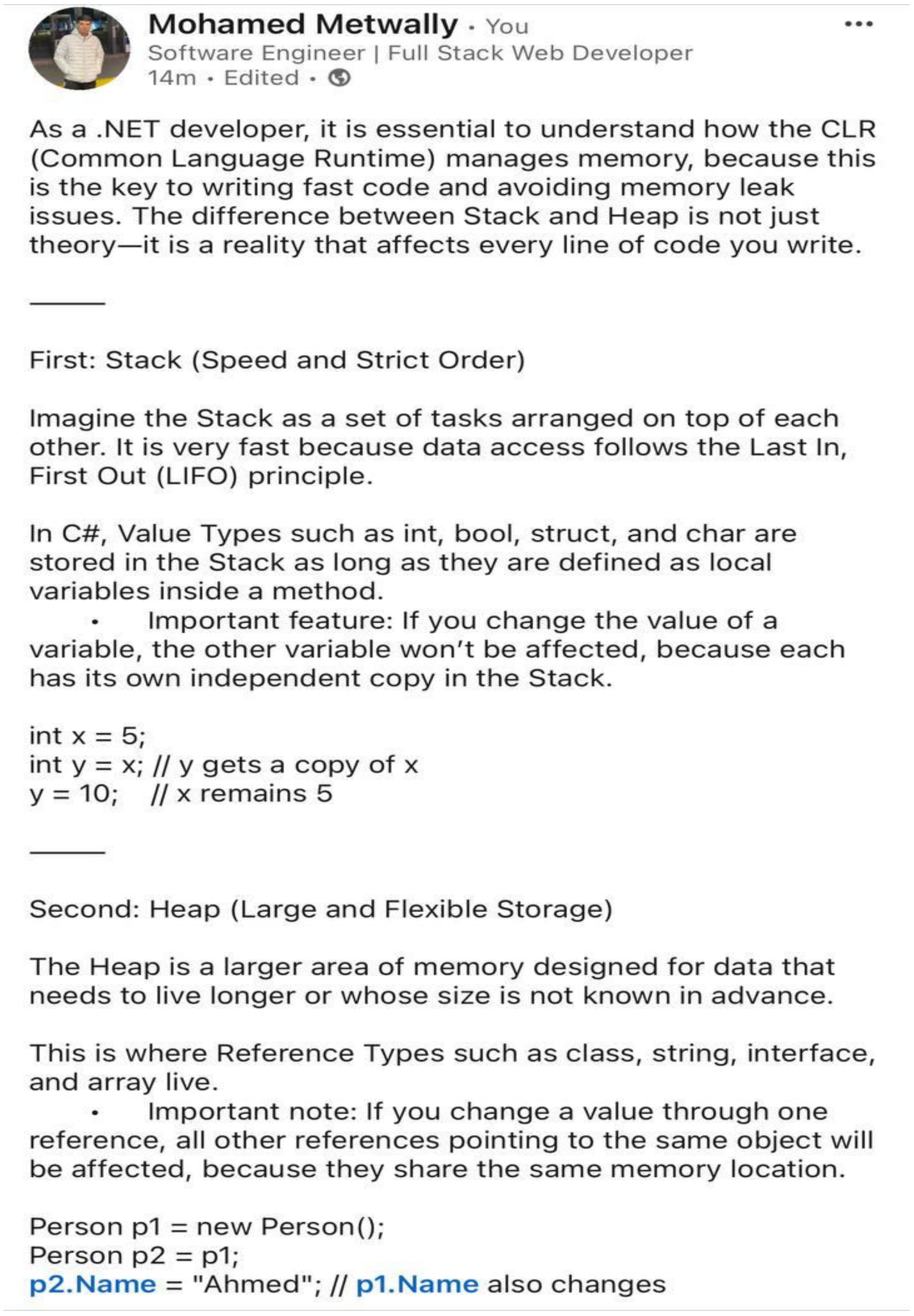
  ++x (prefix):
  increment x first >> x=6
  Use the value x=6

  X++ (postfix):
  use current value x=6
  after using it, increments x >> x=7

  y=6+6=13
  final value of x >> x=7

Part02

- **1) LinkedIn article about variables allocation in stack and heap for both value and ref types**



Mohamed Metwally • You
Software Engineer | Full Stack Web Developer
14m • Edited •

As a .NET developer, it is essential to understand how the CLR (Common Language Runtime) manages memory, because this is the key to writing fast code and avoiding memory leak issues. The difference between Stack and Heap is not just theory—it is a reality that affects every line of code you write.

———

First: Stack (Speed and Strict Order)

Imagine the Stack as a set of tasks arranged on top of each other. It is very fast because data access follows the Last In, First Out (LIFO) principle.

In C#, Value Types such as int, bool, struct, and char are stored in the Stack as long as they are defined as local variables inside a method.

- Important feature: If you change the value of a variable, the other variable won't be affected, because each has its own independent copy in the Stack.

```
int x = 5;
int y = x; // y gets a copy of x
y = 10;    // x remains 5
```

———

Second: Heap (Large and Flexible Storage)

The Heap is a larger area of memory designed for data that needs to live longer or whose size is not known in advance.

This is where Reference Types such as class, string, interface, and array live.

- Important note: If you change a value through one reference, all other references pointing to the same object will be affected, because they share the same memory location.

```
Person p1 = new Person();
Person p2 = p1;
p2.Name = "Ahmed"; // p1.Name also changes
```

- **2) what's the difference between compiled and interpreted languages and in this way what about Csharp?**

  Compiled vs Interpreted Languages
   Compiled: Code is converted to machine code before running (e.g., C, C++).
   Fast and catches errors early.
   Needs recompilation for changes.
   Interpreted: Code runs line by line (e.g., Python, JavaScript).
   Easy to test, good for quick changes.
   Slower and errors show at runtime.

  C#
   C# is compiled to Intermediate Language (IL) first.
   CLR converts IL to machine code at runtime.
   Hybrid: fast like compiled code, but managed and safe like interpreted.

- **3) Compare between implicit, explicit, Convert and parse casting**

  Implicit Casting
   Happens automatically.
   Only between compatible types where no data is lost.
  Example:
  ```
  int x = 10;
  double y = x; // int automatically converts to double
  ```

  Explicit Casting (Cast)
   Requires manual conversion using (type).
   May lose data if converting from larger to smaller type.
  Example:
  ```
  double d = 9.78;
  int i = (int)d; // becomes 9, fractional part is lost
  ```

  Convert Class
   Provided by C# in System.Convert.
   Can convert between types even if not directly compatible.
   Works with value types and strings

Example:

```
string s = “123”;
int x = Convert.ToInt32(s); // string → int
```

Parse Method

 Used only for strings to convert to numeric or date types.

Example:

```
string s = “456”;
int x = int.Parse(s); // string → int
```