

Part01:

```
1  System;
2
3  namespace assignmentRowad
4  {
5      internal class Program
6      {
7          static void Main(string[] args)
8          {
9              //This program declares two integer variables and calculates their sum
10             int x = 10;    // Declare an integer variable x and assign it the value 10
11             int y = 20;    // Declare an integer variable y and assign it the value 20
12
13             /*
14                 Calculate the sum of x and y
15                 and store the result in the variable sum
16             */
17             int sum = x + y;
18
19             // Print the value of sum to the console
20             Console.WriteLine(sum);
21
22         }
23     }
24 }
```

In Visual Studio, the shortcuts are:

- Comment selected code:**
Ctrl + K, Ctrl + C
- Uncomment selected code:**
Ctrl + K, Ctrl + U

```
1     System;
2
3     pace assignmentRowad
4
5     references
6     internal class Program
7
8         0 references
9         static void Main(string[] args)
10        {
11            int x = 10;
12            // Original error: x was assigned "10" (a string) instead of an integer
13
14            int y = 5;
15            // Original error: y was used but never declared
16
17            Console.WriteLine(x + y);
18            // Original error: "console" was written in lowercase instead of "Console"
19        }
20
21
22
```

Runtime Error:

An error that occurs while the program is running. The program compiles but crashes during execution.

Example: dividing by zero.

Example:

```
int x = 10;
```

```
int y = 0;
```

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

Logical Error

An error in the program logic. The program runs normally but gives incorrect results.

Example: using subtraction instead of addition.

Example:

```
int x = 10;
```

```
int y = 5;
```

```
int result = x - y; // Logical error: should be x + y
```

```
Console.WriteLine(result);
```

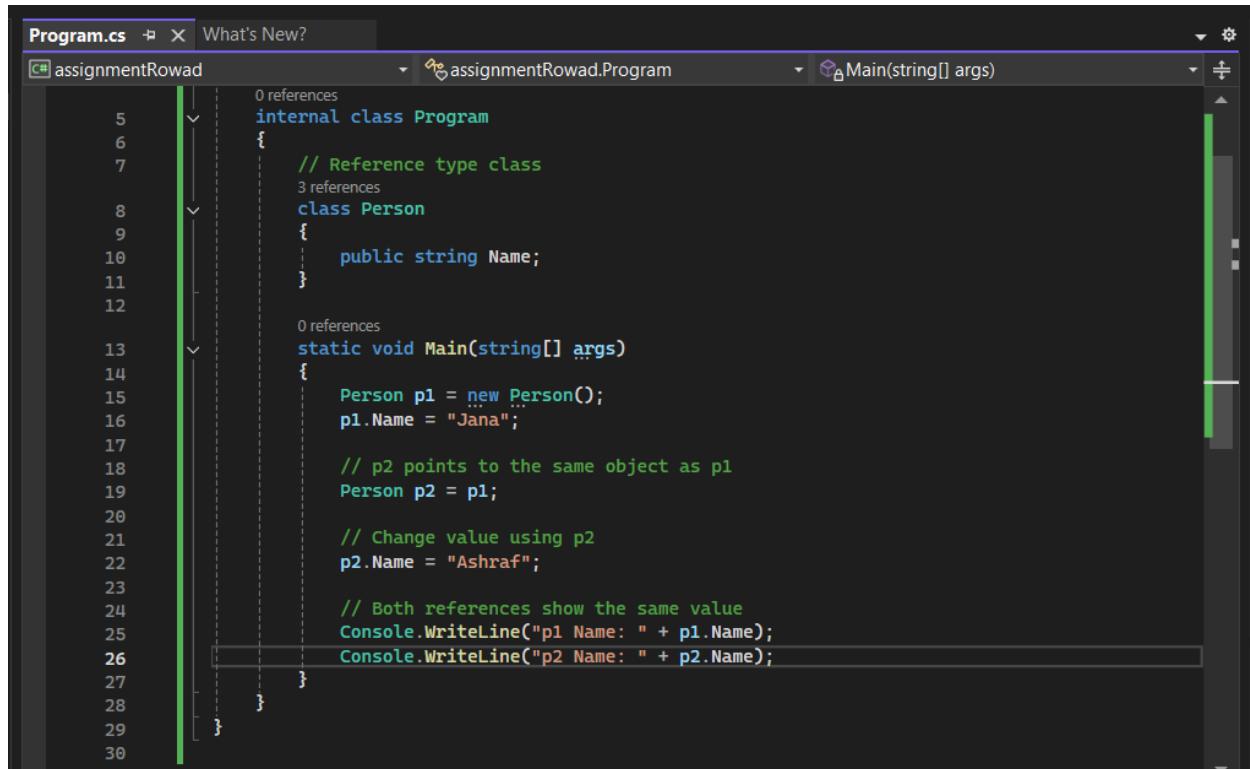
The screenshot shows a code editor window for a C# project named 'assignmentRowad'. The main file is 'Program.cs'. The code defines a class 'Program' with a static void method 'Main'. Inside 'Main', there are four variable declarations: 'fullName' (string), 'age' (int), 'monthlySalary' (double), and 'isStudent' (bool). Each variable is followed by a comment describing its purpose. The code editor interface includes tabs for 'Program.cs', 'assignmentRowad.Program', and 'Main(string[] args)'. The status bar at the bottom shows 'Ln: 14 Ch: 13 SPC CRLF'.

```
1  using System;
2
3  namespace assignmentRowad
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              string fullName = "Your Full Name";    // Stores your full name
10             int age = 21;                        // Stores your age
11             double monthlySalary = 5000.50;       // Stores your monthly salary
12             bool isStudent = true;              // Indicates whether you are a student
13         }
14     }
15
16
17
18 }
```

**It's important to follow naming conventions like
PascalCase in C# because:**

- Improves code readability – code is easier to read and understand.**

- **Makes code more maintainable – easier for you and others to modify later.**
- **Follows C# standards – matches .NET guidelines and professional practices.**
- **Reduces confusion – helps distinguish between variables, methods, and classes.**
- **Improves teamwork – everyone writes code in a consistent way.**



```

Program.cs  X  What's New?
assignmentRowad  assignmentRowad.Program  Main(string[] args)

internal class Program
{
    // Reference type class
    class Person
    {
        public string Name;
    }

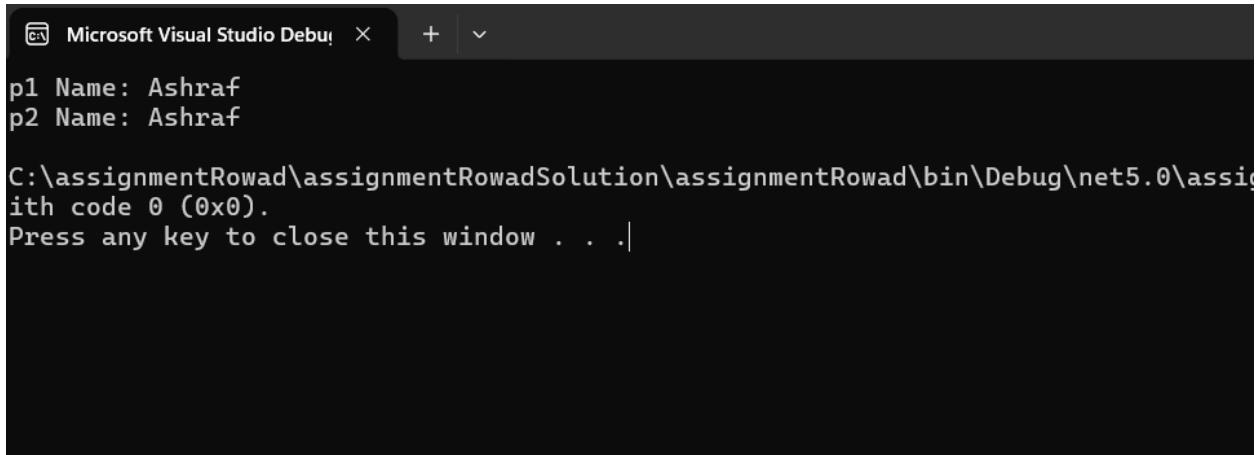
    static void Main(string[] args)
    {
        Person p1 = new Person();
        p1.Name = "Jana";

        // p2 points to the same object as p1
        Person p2 = p1;

        // Change value using p2
        p2.Name = "Ashraf";

        // Both references show the same value
        Console.WriteLine("p1 Name: " + p1.Name);
        Console.WriteLine("p2 Name: " + p2.Name);
    }
}

```



```
p1 Name: Ashraf
p2 Name: Ashraf

C:\assignmentRowad\assignmentRowadSolution\assignmentRowad\bin\Debug\net5.0\assignmentRowad
with code 0 (0x0).
Press any key to close this window . . .|
```

Value Types

- **Stored in the stack**
- **Contain actual value**
- **Assigning creates a copy**

int a = 5;

int b = a; // b gets a copy

Reference Types

- **Stored in the heap**
- **Variables store a reference (address)**
- **Assigning shares the same object**

Person p1 = new Person();

Person p2 = p1; // both point to same object

The screenshot shows the Microsoft Visual Studio IDE interface. The top bar displays "Program.cs" as the active file, "assignmentRowad" as the project name, and "Main(string[] args)" as the current method. The main window contains the C# code for the Program class:

```
1  using System;
2
3  namespace assignmentRowad
4  {
5      internal class Program
6      {
7          static void Main(string[] args)
8          {
9              int x = 15;
10             int y = 4;
11
12             int sum = x + y;
13             int difference = x - y;
14             int product = x * y;
15             double division = x / y;
16             int remainder = x % y;
17
18             Console.WriteLine("Sum: " + sum);
19             Console.WriteLine("Difference: " + difference);
20             Console.WriteLine("Product: " + product);
21             Console.WriteLine("Division result: " + division);
22             Console.WriteLine("Remainder: " + remainder);
23         }
24     }
25 }
26
```

The status bar at the bottom indicates "91 %", "No issues found", and "Ln: 15 Ch: 38 SPC CRLF".

The screenshot shows the Microsoft Visual Studio Debug window. It displays the output of the console application, which includes the calculated values for sum, difference, product, division, and remainder.

```
Sum: 19
Difference: 11
Product: 60
Division result: 3
Remainder: 3
```

Below the output, the full command prompt path is shown: "C:\assignmentRowad\assignmentRowadSolution\assignmentRowad\bin\Debug\net5.0\assignmentRowad". The message "Press any key to close this window . . ." is also visible at the bottom of the window.

A screenshot of a C# code editor window titled "Program.cs*". The code is as follows:

```
1 using System;
2
3 namespace assignmentRowad
4 {
5     internal class Program
6     {
7         static void Main(string[] args)
8         {
9             int a = 2;
10            int b = 7;
11            Console.WriteLine(a % b);
12            // Output: 2
13            // Explanation: a % b gives the remainder of 2 divided by 7.
14            // Since 2 is smaller than 7, it cannot be divided even once,
15            // so the remainder is 2.
16        }
17    }
18}
19
```

The line "Console.WriteLine(a % b);" is highlighted with a yellow selection bar. A tooltip or comment block is displayed below the line, explaining the output and the meaning of the modulus operator.

A screenshot of a C# code editor window titled "Program.cs*". The code is as follows:

```
1 using System;
2
3 namespace assignmentRowad
4 {
5     internal class Program
6     {
7         static void Main(string[] args)
8         {
9             int number = 16;
10
11             // Check if number is greater than 10 AND even
12             if (number > 10 && number % 2 == 0)
13             {
14                 Console.WriteLine("The number is greater than 10 and even.");
15             }
16             else
17             {
18                 Console.WriteLine("The number is NOT both greater than 10 and even.");
19             }
20
21         }
22     }
23}
```

The line "if (number > 10 && number % 2 == 0)" is highlighted with a yellow selection bar. A tooltip or comment block is displayed below the line, explaining the use of the logical AND operator (&&).

&& (Logical AND)

- **Used with boolean conditions**
 - **Short-circuit: if the first condition is false, the second is not evaluated**
 - **Commonly used in if statements**
if (a > 0 && b > 0)
-

& (Bitwise AND)

- **Used with numbers (bit by bit) or booleans**
- **No short-circuit: both sides are always evaluated**
 - **Operates on binary representation**

int result = 5 & 3; // 101 & 011 = 001 → 1

The screenshot shows the Microsoft Visual Studio IDE interface. The top window displays the code for `Program.cs` in the `assignmentRowad` project. The code defines a `Program` class with a `Main` method. Inside `Main`, it reads a double value from the console, performs explicit and implicit casting, and then prints the results back to the console. The bottom window shows the output of the `Debug` session, which includes the input prompt, the casted values, and the path to the executable.

```
1 using System;
2
3 namespace assignmentRowad
4 {
5     internal class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Enter a double value: ");
10            double number = double.Parse(Console.ReadLine());
11
12            // Explicit casting (double to int)
13            int explicitCast = (int)number;
14
15            // Implicit casting (int to double)
16            double implicitCast = explicitCast;
17
18            Console.WriteLine("After explicit casting to int: " + explicitCast);
19            Console.WriteLine("After implicit casting back to double: " + implicitCast);
20
21        }
22    }
23 }
```

Microsoft Visual Studio Debug

```
Enter a double value: 10.97
After explicit casting to int: 10
After implicit casting back to double: 10

C:\assignmentRowad\assignmentRowadSolution\assignmentRowad\bin\Debug\net5.0\assignmentRowad
Press any key to close this window . . .
```

Explicit casting is required when converting a double to an int because:

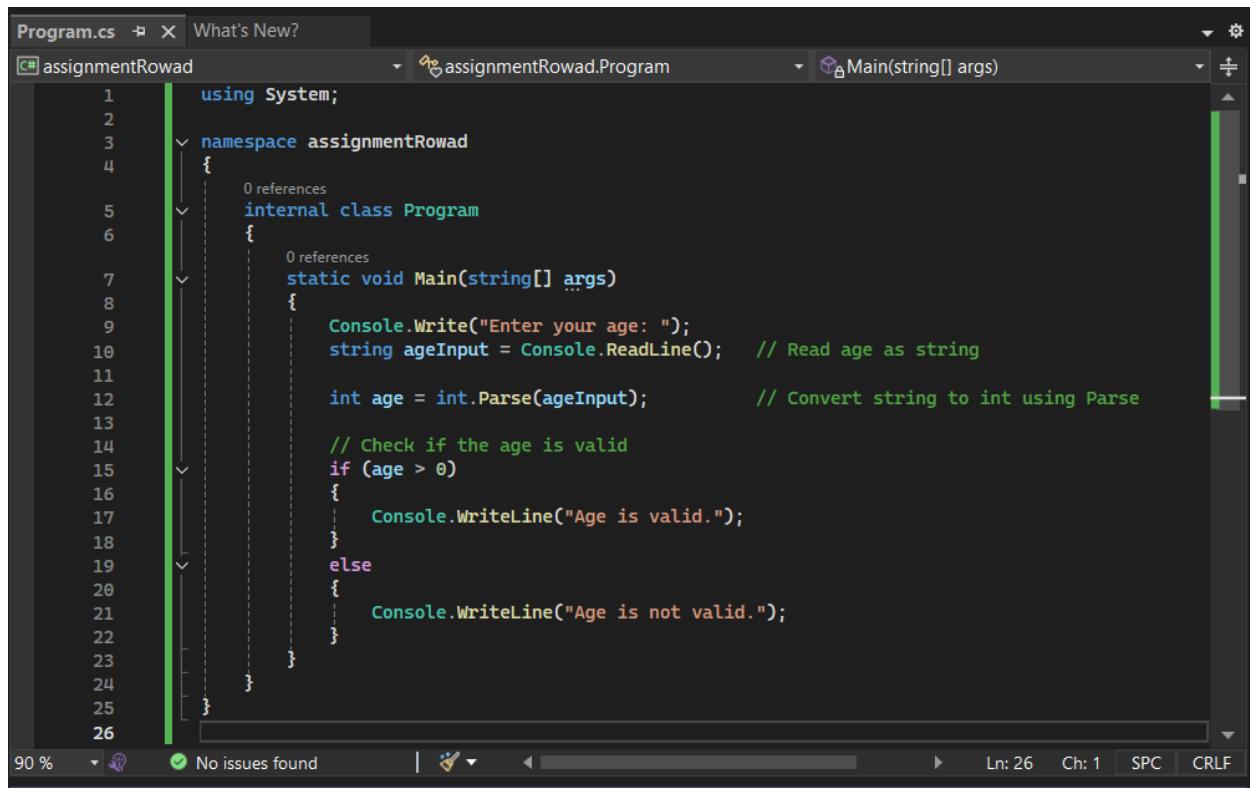
- A double can contain decimal values, while an int cannot.**
- Converting may cause data loss (the decimal part is removed).**

- **C# forces explicit casting to make sure the programmer is aware of this loss.**

Example:

(int)3.7 becomes 3

So, explicit casting prevents accidental loss of information.



```
1 using System;
2
3 namespace assignmentRowad
4 {
5     internal class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Enter your age: ");
10            string ageInput = Console.ReadLine(); // Read age as string
11
12            int age = int.Parse(ageInput); // Convert string to int using Parse
13
14            // Check if the age is valid
15            if (age > 0)
16            {
17                Console.WriteLine("Age is valid.");
18            }
19            else
20            {
21                Console.WriteLine("Age is not valid.");
22            }
23        }
24    }
25 }
26
```

The screenshot shows a Microsoft Visual Studio code editor window. The title bar says "Program.cs" and "assignmentRowad". The code itself is a simple C# program that prompts the user for their age, reads the input, converts it to an integer using `int.Parse`, and then prints a message indicating whether the age is valid (greater than 0). The code editor interface includes a status bar at the bottom showing "Ln: 26 Ch: 1 SPC CRLF".

- **FormatException: occurs when the input is not a valid number (for example, letters instead of digits).**

- **OverflowException: occurs when the input number is too large or too small to fit in an int.**

These exceptions can be handled using exception handling (try-catch) to prevent the program from crashing and to show a proper error message.

The screenshot shows the Microsoft Visual Studio interface. The top part is the code editor with the file 'Program.cs*' open. The code demonstrates integer incrementation:

```
1  using System;
2
3  namespace assignmentRowad
4  {
5      internal class Program
6      {
7          static void Main(string[] args)
8          {
9              int x = 5;
10
11             // Postfix increment
12             Console.WriteLine("Postfix (x++): " + (x++)); // prints 5, then x becomes 6
13
14             // Prefix increment
15             Console.WriteLine("Prefix (++x): " + (++x)); // x becomes 7, then prints 7
16
17         }
18     }
19 }
```

The bottom part is the terminal window titled 'Microsoft Visual Studio Debug'. It displays the output of the program:

```
Postfix (x++): 5
Prefix (++x): 7

C:\assignmentRowad\assignmentRowadSolution\assignmentRowad\bin\Debug\net5.0\assignment
th code 0 (0x0).
Press any key to close this window . . .|
```

- . Given the code below, what is the value of x after execution? Explain why `int x = 5; int y = ++x + x++;` :
`++x` increases x first → x becomes 6

- . **x++ uses the value first, then increases → x becomes 7**

So after execution, x = 7.

Part02:



Jana Ashraf
Software Engineering Student
@ACU| Software Developer | Full...
Giza, Al Jizah
Ahram Canadian University - ACU

Experience 3.7x more candidate replies
Try Recruiter Lite for EGPO

Profile viewers 90
Post impressions 30

Home 8 Jobs Messaging Notifications M

Stack vs Heap C# في حفلة الذاكرة (Memory):
خليج برماجك زي حفلة
أكل خفيف على الطاولة = Stack
سرع ومبشر.
أمثلة: int, double, bool
`int a = 5;`
كل واحد عنده نسخه الخاصة //
منش هيتغير a لو غيرت
Reference Types (Heap) بيترأ كبيرة في المطبخ
البيترأ في مكان واحد، وكل الناس يتشاركها.
أمثلة: class, string, array
`Person p1 = new Person();`
`Person p2 = p1; //` لو غيرت
هيتغير كمان p1 → p2 لو غيرت.
ملخص سرع
Stack = سرع خاص
Heap = مشترك + أكبر
ذاكرة زي حفلة 🎉
و هو المنظم 😊
#CSharp #DotNet #SoftwareEngineering #Programming #Coding #Tech
#Developer #ComputerScience

C#: Stack vs Heap

</ GWeaths > DEV

Compiled Languages

- **Code is converted into machine code before running.**
 - **The program runs fast.**
 - **Examples: C, C++**

What happens:

Source code → Compiler → Machine code → Run

Interpreted Languages

- **Code is executed line by line at runtime.**
- **Usually slower than compiled languages.**
 - **Examples: Python, JavaScript**

What happens:

Source code → Interpreter → Runs line by line

So what about C#?

C# is both compiled and interpreted.

How it works:

1. **Compiled into Intermediate Language (IL)**
(not machine code yet)

2. Executed by the .NET runtime (CLR) which interprets or JIT-compiles it into machine code

1. Implicit Casting (Automatic)

- No data loss**
- Automatic conversion**
- Usually from smaller type to larger type**

Example:

```
int a = 10;  
double b = a; // implicit casting
```

2. Explicit Casting (Manual)

- Possible data loss**
- You must write the cast**
- Used when converting from larger type to smaller type**

Example:

```
double a = 9.8;
```

```
int b = (int)a; // explicit casting
```

3. Convert Class

- **Converts between different data types**
- **Handles null values and returns default values**
 - **Safer than Parse**

Example:

```
string s = "123";
```

```
int n = Convert.ToInt32(s);
```

4. Parse Method

- **Converts string to a specific type**
- **Throws exception if string is invalid**
 - **Faster but less safe**

Example:

```
string s = "123";
```

```
int n = int.Parse(s);
```

Part03 Bonus:

Sure! Here is a clean and organized self-study summary for C# Session 2, covering all the topics you listed.

You can use this as a quick revision sheet or study notes.

C# Session 2 Self-Study Summary

1. Creating Objects (Reference Types)

- **Reference types (class, array, string, etc.) are stored in the heap.**
 - **When you create an object using new, the reference is stored in the stack.**
 - **Example:**
 - **Person p = new Person();**

2. Memory Leak

- **Occurs when unused objects are still referenced, so memory can't be freed.**
- **Common cause: event handlers, static lists, long-lived references.**

3. Garbage Collector (GC)

- **Automatically frees unreferenced objects from the heap.**
 - **Runs when needed (not predictable).**
 - **Helps prevent memory leaks.**

4. Customizing Garbage Collector

- **You can influence GC behavior using:**
 - **GC.Collect()**
 - **GC.SuppressFinalize()**
 - **GC.WaitForPendingFinalizers()**
- **Use carefully, not recommended in normal code.**

5. Unmanaged Resources

Examples:

- **Database connections**
 - **File streams**
 - **Network connections**

6. Scientific Notation / FPU

- **float and double use floating-point representation.**
 - **They can store very large or very small numbers.**
 - **Example:**
 - **float f = 1.23e-4f;**
 - **double d = 3.14e+6;**

7. When to Use Decimal?

Use decimal when precision matters, especially in:

- **Financial calculations**
 - **Currency values**

8. Difference Between float, double, and decimal

Type	Precision	Use Case	Memory
float	7 digits	Scientific	4 bytes
double	15 digits	Most calculations	8 bytes
decimal	28 digits	Money/finance	16 bytes

9. Checked & Unchecked Blocks

Used to handle overflow in numeric operations.

checked

```
{  
    int x = int.MaxValue;  
x++; // throws OverflowException  
}
```

unchecked

```
{  
    int x = int.MaxValue;  
x++; // no exception, wraps around  
}
```

10. Parsing Null Values

Example:

```
int x = int.Parse(null); // throws  
ArgumentNullException
```

11. Parse vs Convert (Performance)

- **Parse is faster but throws exception on invalid input.**
 - **Convert is safer, handles null, but slightly slower.**
-

12. Stack vs Heap

- **Stack: fast, stores value types and references.**
 - **Heap: slower, stores objects (reference types).**
-

13. Deallocation

- **Value types are deallocated automatically when the scope ends.**
 - **Reference types are deallocated by GC when no longer referenced.**
-

14. Bitwise Operators

Used for bit-level operations:

- **& (AND)**
- **| (OR)**
- **^ (XOR)**

- **~ (NOT)**
- **<< (left shift)**
- **>> (right shift)**

Example:

```
int a = 5; // 0101
```

```
int b = 3; // 0011
```

```
int c = a & b; // 0001 = 1
```

Great question! Here's a simple and clear explanation:

 **What does “C# is managed code” mean?**

When we say C# is managed code, it means:

 **The .NET runtime (CLR) controls your program**

The Common Language Runtime (CLR) handles:

- **Memory allocation**
- **Memory deallocation**
- **Garbage collection**
- **Type safety**
- **Security checks**

So you don't manually manage memory like in C++.

 **Why is this important?**

✓ It prevents memory errors

Examples:

- **No manual free() or delete**
- **Less chance of memory leaks (but still possible if references stay)**

✓ It makes code safer and more reliable

Because the CLR checks and manages resources.

 **What does “struct is considered like class before” mean?**

In C#, structs and classes are both used to create custom data types.

So when people say “struct is like class before”, they usually mean:

✓ Both can contain:

- **Fields (variables)**
- **Methods**

- Constructors

- Properties

Example:

```
struct Student
```

```
{
```

```
public string Name;
```

```
public int Age;
```

```
}
```

This looks very similar to a class.



The main difference (important!)

Even though they look similar, they behave differently:

- ◆ **Struct (Value Type)**

- **Stored in stack**

- **Copied when assigned**

- **Best for small data**

- ◆ **Class (Reference Type)**

- **Stored in heap**

- **References are copied**

- **Best for larger objects**