

Part01

Q1. What is the difference between `int.Parse` and `Convert.ToInt32` when handling null inputs?

`Int.Parse(null)` = throws `ArgumentNullException`

`Convert.ToInt32(null)` = 0

Q2. Why is `TryParse` recommended over `Parse` in user-facing applications?

`TryParse` is handling bad arguments by returning 0 then complete the remaining program unlike the `Parse` which in this case throw an exception and the program will be crashed

Q3. Explain the real purpose of the `GetHashCode()` method.

It's a number to check if two objects points on the same reference or not (in case of referenced types)

If valued types it returns a value that represents its hash code in the stack

Q4. What is the significance of reference equality in .NET?

When we put `obj1 = obj2` that means the references of them will equal to other not only their values

Q5. Why string is immutable in C#?

It's the most efficient in case reading, no much modifications.

No a lot of modifications = No a lot of unreachable objects

Q6. How does `StringBuilder` address the inefficiencies of string concatenation?

String builder uses linked list (dynamic) so each char points to the other. This allows us to insert/remove into a specific position or appending with less space complexity. In addition to being mutable so editing any SB this means we will edit the same object and won't create a new obj

Unlike strings that are static (arrays of chars) then any operation will force us to create a new modified object (or destroy the old one then create a new one [GC do this after the memory has been filled])

Q7. Why is `StringBuilder` faster for large-scale string modifications?

Because of using linked lists in SB not just static arrays. This ease modifications to prevent memory wasting.

Q8. Which string formatting method is most used and why?

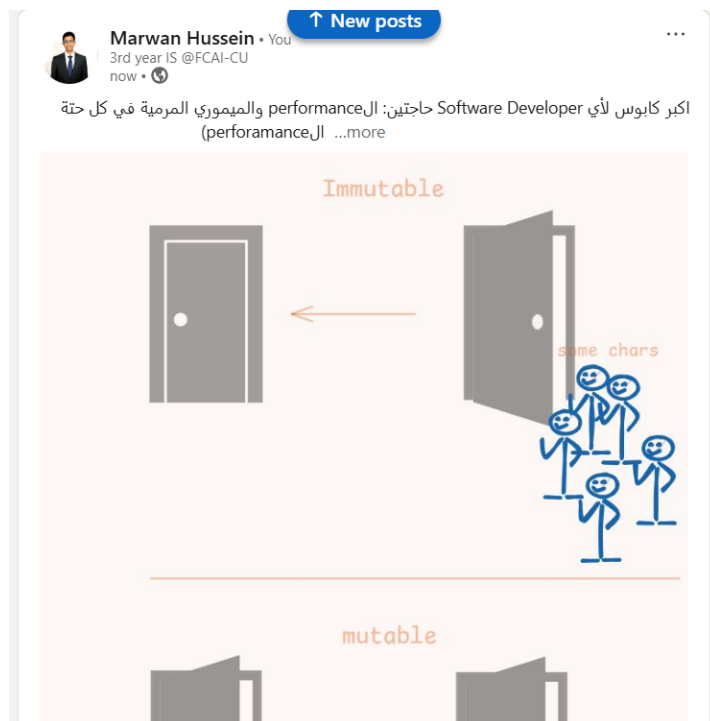
String Interpolation (using `$`) -> it's more readable and using less memory and we can use math and logic in it.

Q9. Explain how StringBuilder is designed to handle frequent modifications compared to strings.

	SB	String
type	Dynamic	Static
immutable	false	true
initially uses	Linked list	Arrays
best for	a lot of modification	read only data
shortcomings	high complexity (compared with strings)	memory wasting if we used it in a lot of modifications

Part02

1- LinkedIn article about string immutability (https://www.linkedin.com/posts/marwan-hussein-568373314_dotnet-softwareengineering-performance-activity-7422776756459368448-nSuZ?utm_source=share&utm_medium=member_desktop&rcm=ACoAAE_JsmABvTFVwhoNTi9LOJQCQO2JKvt0pM)



2- What's Enum data type, when is it used? And name three common built_in enums used frequently?

Enum = Enumeration: distinct value type -> allows us to create a set of named constants

When it used:

* using some magic keywords (instead of numbers) based on numbers like Red Black Tree (Enum elements are Red =0, Black=1)

* to make the code readable

Examples:

* ConsoleColor: to change text or background of the terminal like values (Red, Green, Blue,..)

* HttpStatusCode: (OK = 200, NotFound = 404, InternalServerError = 500)

* DayOfWeek (Sunday = 0, Monday = 1, ..etc)

3- what are scenarios to use string Vs StringBuilder?

String: static strings, using threads in a system

SB: a lot of modification in some string

Part03 (Bonus)

4- self-study report

1. Composite formatting

Takes a list of objects (string + format items). The arguments are indexed from 0 to n-1 {n: #of format items}

How works: parsing the string for curly brackets {}, then placeholder the suitable item in the suitable index

If the passed obj is a valueType then we Box it to be as referenced type (.ToString()) that moves the obj from the stack to the heap

2. Interpolation formatting

Syntactic sugar, when compiling, the compiler transforms an interpolated string into string.Format call anyway

3. .rdata section

Read-only data section

In general, the compiler sorts the data into different buckets based on how these data will be used. The rdata section is the bucket for strings that need to be seen by the program but shouldn't be changed while the program is running

e.g. constants, strings, import info

4. How the jump happens in switch

Using jump table:

Cases in the switch stored in rdata section

So if the value not in the range (min, max) -> goto the default scope

If cases are close to each other (like 1,2,3,4) then perform the case's index in the rdata

Else if they are far like (1,100,1e6): perform them as BST (binary search tree) or hash table to treat with string cases. This makes us avoid wasted memory

5. Evolution of if/switches in C# versions

6.

5- what meant by user defined constructor and its role in initialization

The role of initialization to initialize objects with a specific algorithm (may take parameters or something)

6- compare between Array and Linked List

	Array	Linked List
memory allocation	Contiguous cells	not must to be contiguous
type	static	dynamic
random access	yes	no
accessing speed	fast $[O(1)]$	slow $[O(n)]$
memory waste	can be if the array almost empty	minimal waste