01 Introduction to C# and Data Types

Understanding Data Types
Test your Knowledge
1. What type would you choose for the following "numbers"?
A person's telephone number
A person's height
A person's age
A person's gender (Male, Female, Prefer Not To Answer)
A person's salary
A book's ISBN
A book's price
A book's shipping weight
A country's population
The number of stars in the universe
The number of employees in each of the small or medium businesses in the United Kingdom (up to about 50,000 employees per business)

Scenario	Best Data Type	Reason
A person's telephone number	`string`	Phone numbers may have leading zeros, country codes (`+1`, `+44`), and special characters (`-`, `()`).
A person's height	`float` or `double`	Height may have decimal values (e.g., 5.9 feet, 175.5 cm). Use `float` for less precision, `double` for more.
A person's age	`int`	Age is always a whole number and does not require decimals.
A person's gender (Male, Female, Prefer Not To Answer)	`enum` or `string`	An `enum` is ideal for predefined categories, but a `string` allows flexibility for additional options.
A person's salary	`decimal`	`decimal` is preferred for monetary values due to its higher precision in financial calculations.
A book's ISBN	`string` .	ISBNs contain digits, hyphens, and sometimes letters (`978-3-16-148410-0`), so `string` is the best choice
A book's price	`decimal`	Prices involve currency and should use `decimal` for accuracy.

A book's shipping weight	`float` or `double`	Weight may include decimals (e.g., `1.25 kg`), so `float` or `double` is suitable.
A country's population	`long`	Population numbers can be very large (billions), so `long` is needed to handle large integers.
The number of stars in the universe	`ulong` (unsigned long) or `BigInteger`	The number is **extremely large**, and `BigInteger` (from `System.Numerics`) can handle unlimited digits
The number of employees in each of the small or medium businesses in the UK (up to about 50,000 employees per business)	`int`	`int` can store values up to ~2 billion, which is enough for this range.

Summary of Data Types:

- **Use `string`** for non-numeric values that may contain special characters (e.g., phone numbers, ISBNs).
- **Use `int` or `long`** for whole numbers, depending on their range.
- **Use `decimal`** for currency-related values to maintain precision.
- **Use `float` or `double`** for measurements (height, weight) where decimals are needed.
- **Use `enum`** for predefined categories like gender.
- **Use `BigInteger`** for extremely large numbers (e.g., stars in the universe).

2. What are the difference between value type and reference type variables? What is boxing and unboxing?

Feature	Value Type	Reference Type
Storage Location	Stored in **stack memory**	Stored in **heap memory**
Data Handling	Holds the **actual value**	Holds a **reference (memory address)** to the value
Copy Behavior	When assigned to another variable, a **copy** is created	When assigned, **both variables point to the same object**
Examples	`int`, `float`, `double`, `char`, `bool`, `struct`, `enum`	`string`, `class`, `object`, `array`, `interface`, `delegate`

Boxing and Unboxing

- **Boxing** → Converting a **value type** to an **object (reference type)**.
- **Unboxing** → Extracting the **value type** from an **object**.

3. What is meant by the terms managed resource and unmanaged resource in .NET?

Feature	**Managed Resource**	**Unmanaged Resource**
Definition	Resources managed automatically by the .NET runtime	Resources not handled by .NET, requiring manual cleanup
Examples	Objects, strings, arrays, `List<T>`	Files, database connections, network sockets, COM objects
Memory Management	Handled by the Garbage Collector	**Requires manual disposal** using `Dispose()` or `finalizer`
Cleanup Method	GC handles cleanup automatically	Implement `IDisposable` and call `Dispose()`

4. Whats the purpose of Garbage Collector in .NET?

The .NET Garbage Collector (GC) automatically manages memory, allowing memory to be allocated and de-allocated in a manner in which memory leaks are prevented and application performance is enhanced

Controlling Flow and Converting Types

Test your Knowledge

1. What happens when you divide an int variable by 0?

If you try to divide an int variable by 0 in C#, during runtime, it will throw a System.Divide By Zero Exception, and your application will get crashed. To prevent this, use error handling like try-catch or if checks.

Example:

Division by Zero with `int`

```
int a = 10;
int b = 0;
int result = a / b;
```

Output:

```
System. Divide By Zero Exception: Attempted to divide by zero.
```

2. What happens when you divide a double variable by 0?

When you divide a `double` variable by `0` in C#, it does **not throw an exception**. Instead, it returns **Infinity (∞)**, **Negative Infinity (-∞)**, or **NaN (Not a Number)** based on IEEE 754 rules.

Example:

```
double a = 10.0, b = 0.0;
 Console.WriteLine(a / b);
Console.WriteLine(b / b);
```

Output:

```
Infinity
NaN
```

3. What happens when you overflow an int variable, that is, set it to a value beyond its range?

When an `int` variable exceeds its **maximum (`2147483647`)** or **minimum (`-2147483648`)** value, it wraps around **unless checked mode is enabled**. In checked mode, an `OverflowException` occurs.

Example:

```
int max = int.MaxValue;
int overflow = max + 1;
Console.WriteLine(overflow);
```

Output:

```
-2147483648
```

4. What is the difference between x = y++; and x = ++y;?

- **`x = y++;`** → Assigns `y` to `x`, then increments `y` (**post-increment**).
- **`x = ++y;`** → Increments `y`, then assigns it to `x` (**pre-increment**).

Example:

```
int y = 5;
int x = y++; // x = 5, y = 6
 Console.WriteLine($"x: {x}, y: {y}");

y = 5;
 x = ++y; // x = 6, y = 6
 Console.WriteLine($"x: {x}, y: {y}");
```

Output:

```
x: 5, y: 6
x: 6, y: 6
```

5. What is the difference between break, continue, and return when used inside a loop statement?

- `break` → Exits the loop immediately.
- `continue` → Skips the current iteration and moves to the next.
- `return` → Exits the entire method.

Example:

```
for (int i = 1; i <= 5; i++)
 {
if (i == 3) continue;
 if (i == 5) break;
Console.Write(i + " ");
}
```

Output:

```
1 2 4
```

6. What are the three parts of a for statement and which of them are required?

1. **Initialization (`int i = 0`)** → Sets up the loop variable.
2. **Condition (`i < n`)** → Checks if the loop continues.
3. **Iteration (`i++`)** → Updates loop variable.

Only the condition is required, others can be omitted.

Example:

```
int i = 0;
for (; i < 3;)
{
   Console.WriteLine(i++);
}
```

Output:

```
0
1
2
```

7. What is the difference between the = and == operators?

- **= (Assignment Operator)** → Assigns a value to a variable.
- **== (Comparison Operator)** → Compares two values and returns `true` or `false`.

Example:

```
int a = 5;
bool is Equal = (a == 5);
Console. WriteLine(isEqual);
```

Output:

```
True
```

8. Does the following statement compile? for (; true;) ;

Yes, it compiles because the **condition is always `true`**, creating an **infinite loop**.

Example:

```
for (; true; ) Console.WriteLine("Running...");
```

9. What does the underscore _ represent in a switch expression?
In C#, the underscore _ in a **switch expression** acts as a **default case** or **discard pattern**, meaning it matches anything not explicitly handled by other cases. It ensures that all possible values are accounted for, preventing missing case errors.

10. What interface must an object implement to be used in foreach?
To be iterated using a foreach loop, an object must implement **IEnumerable** or **IEnumerable<T>**, which provides an **iterator** for traversing a collection. This enables iteration without explicitly managing an index, making it useful for lists, arrays, and collections.