1. nameof Exercises

The nameof operator returns the string name of a variable, type, or member. It's a lifesaver for refactoring.

1. **Argument Validation:** Create a method SetAge(int age). If age is less than 0, throw an ArgumentOutOfRangeException. Use nameof to pass the parameter name to the exception constructor.
2. **Property Change Notification:** Create a class User with a property Username. Write a method NotifyChange(string propertyName) and call it within the setter of Username using nameof so that if you rename the property later, the string stays updated.
3. **Logging Constants:** Write a script that logs the name of a specific method and a specific class to the console using nameof without hardcoding strings like "MyMethod".

---

2. typeof Exercises

The typeof operator is used to obtain the System.Type object for a type. This is resolved at **compile-time**.

1. **Type Checking:** Create an interface IShape and a class Circle. Write a program that uses typeof(Circle) to check if a specific object's type matches exactly, then print the full namespace and name of the type.
2. **Generic Metadata:** Write a generic method GetTypeName<T>(). Inside, use typeof(T) to return the name of the type passed as a generic argument.
3. **Attribute Inspection:** (Advanced) Create a custom attribute HelpAttribute. Apply it to a class DataProcessor. Use typeof(DataProcessor).GetCustomAttributes() to verify if the attribute is present.

---

3. sizeof Exercises

The sizeof operator returns the size (in bytes) of a type. Note that for user-defined structs, this usually requires an unsafe context.

1. **Primitive Mapping:** Write a program that prints the sizeof for int, long, double, and bool. Observe the difference between a 32-bit integer and a 64-bit double.
2. **Custom Struct Size:** Create a struct named Point containing two int fields (x and y). Use sizeof inside an unsafe block (or System.Runtime.CompilerServices.Unsafe.SizeOf<T>) to find out how many bytes the struct occupies.
3. **Memory Math:** If an array contains 1,000 long integers, calculate the total memory consumption in bytes by multiplying the array length by sizeof(long).

---

4. default Exercises

The default operator produces the default value of a type (e.g., 0 for numbers, null for references).

1. **Array Initialization:** Create an array of 5 DateTime objects. Use a loop or a direct assignment to set the third element to default. Print the result to see what the "zero-state" of a DateTime looks like.
2. **Generic Defaults:** Write a generic class Box<T>. Include a method ResetValue() that sets the internal value of type T back to its default. Test this with both an int (should become 0) and a string (should become null).
3. **The Default Literal:** In C# 7.1+, you can use the literal default without the type name. Create a method DoSomething(int count = default, string message = default) and demonstrate how calling it without arguments utilizes these default values.