Assignment 1 Introduction to C# and Data Types

**Understanding Data Types**

1. What type would you choose for the following “numbers”?

* A person’s telephone number : string
* A person’s height: floating
* A person’s age: int
* A person’s gender (Male, Female, Prefer Not To Answer): enum
* A person’s salary: decimal
* A book’s ISBN: string
* A book’s price: decimal
* A book’s shipping weight: float
* A country’s population: long
* The number of stars in the universe: ulong
* The number of employees in each of the small or medium businesses in the United Kingdom (up to about 50,000 employees per business): int

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

A **value type** in C# stores the actual data directly in stack memory. When you assign one value type variable to another, a copy of the data is made, so changing one variable does not affect the other. Common value types include int, float, bool, char, struct, and enum. Value types are typically faster because they are stack-allocated and automatically cleaned up when they go out of scope.

A **reference** **type** stores a reference (memory address) to the actual object, which resides in heap memory. Assigning one reference type variable to another copies the reference, not the object itself, so multiple variables can point to the same object. This means that modifying the object through one reference also affects all other references pointing to it. Common reference types include class, string, array, and object.

**Boxing** is the process of converting a value type (like int, float, bool) into a reference type (usually object). When a value type is boxed, the runtime creates a new object on the heap and copies the value into it. boxing has a performance cost because it involves memory allocation and copying the value.

**Unboxing** is the reverse of boxing. It extracts the value type from the boxed object and copies it back to a stack variable. Unboxing requires an explicit cast to the original value type, and if the cast is incorrect, it throws an exception. Like boxing, unboxing also has a performance cost because it involves type checking and copying from the heap to the stack.

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

A **managed resource** is any resource that is handled automatically by the .NET runtime’s garbage collector (GC). This includes objects like strings, arrays, classes, and most .NET objects. The garbage collector automatically frees the memory when the object is no longer referenced, so you usually don’t need to manually release it.

An **unmanaged** **resource** is any resource not handled by the .NET garbage collector. These are typically OS-level resources such as file handles, database connections, network sockets, windows handles, or unmanaged memory. Since the GC cannot clean these up automatically, you must release them manually, usually by implementing the IDisposable interface and calling Dispose() or using a using statement.

4. what’s the purpose of Garbage Collector in .NET?

The **Garbage Collector** is a memory management feature of the .NET runtime that automatically reclaims memory occupied by objects that are no longer in use. Its main purpose is to simplify memory management and prevent memory leaks.

**Controlling Flow and Converting Types**

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

Dividing an int by 0 in C# throws a DivideByZeroException at runtime.

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

Dividing a double by 0 in C# does not throw an exception; it returns Infinity (for nonzero numerator) or NaN (if numerator is also 0).

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

Overflowing an int wraps around by default, but in a checked context it throws an OverflowException.

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

y++ increments after use (post-increment), ++y increments before use (pre-increment).

Example of y++:

int y = 5;

int x = y++;

// x = 5, y = 6

Example of ++y;

int y = 5;

int x = ++y;

// x = 6, y = 6

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

* Break: exits the current loop immediately.
* Continue: skips the rest of the current iteration and moves to the next.
* Return: exits the entire method, stopping the loop and any remaining code.

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

for (initializer; condition; iterator){};

* Initializer: runs once before the loop starts (e.g., int index = 0).
* Condition: checked before each iteration; loop runs while true (e.g., index < 10).
* Iterator: runs after each iteration (e.g., index++).

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

* “=” operator is used for assignment, meaning it stores the value on the right-hand side into the variable on the left-hand side.
* “==” operator is used for comparison; it checks whether two values are equal and returns a Boolean result (true or false).

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

* It compiles because all three parts of a for loop are optional. Here, the initializer and iterator are omitted, the condition is always true, and the loop body is an empty statement. This creates an infinite loop that does nothing.

9. What does the underscore \_ represent in a switch expression?

* “\_” in a switch expression catches all remaining cases that aren’t explicitly handled.

10. What interface must an object implement to be enumerated over by using the foreach statement?

* Any object that implements IEnumerable or IEnumerable<T> can be used with foreach.