1. Stack (LIFO - Last In, First Out)

The Stack<T> is your go-to for "undo" mechanics or reversing data.

- **Question 1: The String Reverser** Create a function string ReverseString(string input) that uses a Stack<char> to reverse a given string.
- **Question 2: Balanced Parentheses** Write a method bool IsBalanced(string input) that checks if a string of parentheses (e.g., "(())" or "()()") is balanced. Push opening brackets onto the stack and pop them when you hit a closing bracket.
- **Question 3: The Back Button** Simulate a browser's "Back" functionality. Create a class BrowserHistory with methods Visit(string url) and Back(). Use a stack to keep track of the pages visited.

---

2. Queue (FIFO - First In, First Out)

Use Queue<T> when you need to process items in the exact order they arrived.

- **Question 1: The Print Spooler** Create a simple PrintJob queue. Add five strings representing document names to a Queue<string>. Use a while loop to "print" (console log) and remove them one by one until the queue is empty.
- **Question 2: Circular Buffer Simulation** Write a program that accepts a stream of integers. Keep only the last 3 integers entered. Every time a 4th integer is added, the oldest one must be removed from the queue.
- **Question 3: Level-Order Sequence** Given an array representing a flat binary tree, use a Queue to print the elements level by level. (Basic logic: Enqueue the root, then repeatedly dequeue an item and enqueue its children).

---

3. HashSet (Unique Elements)

HashSet<T> is optimized for high-performance set operations and ensuring no duplicates exist.

- **Question 1: Duplicate Finder** Write a method List<int> GetDuplicates(int[] numbers). Use a HashSet to identify which numbers in the array appear more than once, and return a list of those duplicates.
- **Question 2: Unique Word Counter** Take a long paragraph of text, strip the punctuation, and convert it to lowercase. Use a HashSet<string> to count how many *unique* words are in the paragraph.
- **Question 3: Set Intersection** Create two HashSet<int> objects with some overlapping numbers. Use the built-in .IntersectWith() method to find common elements, and .SymmetricExceptWith() to find elements that are unique to each set.

Tips for Implementation

- **Stack:** Remember that Peek() lets you look at the top item without removing it.
- **Queue:** Use TryDequeue() if you are worried about running operations on an empty collection.
- **HashSet:** Use Add()—it returns false if the item already exists, which is a great shortcut for logic checks.