CodeBash · codebash.co.uk

FREE RESOURCE

100 C# Programming Challenges - Teacher Solutions

Multiple solution approaches for each challenge, including manual implementations that do not rely on built-in functions. For teacher use only.

Created by Eoin Shannon

CodeBash

Get all 100 challenges (with solutions pack) free at **codebash.co.uk/resources**

How to Use This Pack

This document contains solution notes and example code for all 100 challenges. Multiple approaches are provided where relevant - a built-in approach, a manual approach (without built-in functions), and a recursive or advanced approach where applicable. These are for teacher reference only.

Which Tier for Which Class?

Year Group	Recommended Tiers
KS3 (Year 7-9)	Tier 1 (Foundation), selected Tier 2 (Intermediate)
Year 10-11 (Secondary)	Tiers 1-3 (Foundation, Intermediate, Applied)
A-Level / Further	Tiers 3-4 (Applied, Stretch Challenges)

Foundation: Basic syntax, input/output, selection, and loops. Suitable for KS3 and secondary foundation learners.

Intermediate: Functions, lists, strings, OOP basics, and classic algorithms. Core secondary and lower A-Level.

Applied: File I/O, error handling, data structures, recursion, and applied algorithms. Upper secondary and A-Level.

Stretch Challenges: Multi-part problems combining multiple skills. A-Level standard. Maps to NEA complexity requirements.

Foundation

Basic syntax, input/output, selection, and loops. Suitable for KS3 and secondary foundation learners.

Challenge 1 Hello, Name

PROBLEM

Write a program that asks the user for their name and displays a personalised greeting.

EXAMPLE

```
Enter your name: Alice
Hello, Alice! Welcome to C#.
```

MARK SCHEME

Correctly uses Console.ReadLine(). Outputs a greeting that includes the name entered.

EXTENSION

Ask for first and last name separately. Display "Hello, [first] [last]!"

TEACHER NOTES

> Works well as a first C# task. Students often forget to store the return value of Console.ReadLine() - and it returns a nullable string (string?), which is worth acknowledging early.

SOLUTIONS

Standard approach

```
Console.Write("Enter your name: ");
string name = Console.ReadLine()!;
Console.WriteLine($"Hello, {name}! Welcome to C#.");
```

First and last name (extension)

```
Console.Write("First name: ");
string first = Console.ReadLine()!;
Console.Write("Last name: ");
string last  = Console.ReadLine()!;
Console.WriteLine($"Hello, {first} {last}!");
```

Challenge 2

Age Calculator

PROBLEM

Ask the user for the year they were born. Calculate and display their age this year.

EXAMPLE

```
Enter your birth year: 2008
You are 17 years old.
```

MARK SCHEME

Correct use of int.Parse(). Correct subtraction. Output includes the calculated age.

EXTENSION

Ask for birth month and day. Give a more precise age in years and months.

TEACHER NOTES

> Introduces type parsing naturally. Students are often surprised that Console.ReadLine() always returns a string - setting this expectation before they see the error saves debugging time.

SOLUTIONS

Standard approach

```
Console.Write("Enter your birth year: ");
int birthYear   = int.Parse(Console.ReadLine()!);
int currentYear = 2025;
int age         = currentYear - birthYear;
Console.WriteLine($"You are approximately {age} years old.");
```

Using DateTime for exact current year

```
Console.Write("Enter your birth year: ");
int birthYear   = int.Parse(Console.ReadLine()!);
int currentYear = DateTime.Now.Year;
int age         = currentYear - birthYear;
Console.WriteLine($"You are approximately {age} years old.");
```

Challenge 3 — Area of a Rectangle

PROBLEM

Ask for the length and width of a rectangle. Display its area and perimeter.

EXAMPLE

```
Enter length: 8
Enter width: 5
Area: 40
Perimeter: 26
```

MARK SCHEME

Correct formulae for both area and perimeter. Both values displayed clearly.

EXTENSION

Also calculate the diagonal length using Math.Sqrt().

TEACHER NOTES

> A good consolidation task after covering variables and arithmetic operators. Encourage students to use descriptive variable names rather than single letters.

SOLUTIONS

Standard approach

```
Console.Write("Enter length: ");
double length = double.Parse(Console.ReadLine()!);
Console.Write("Enter width: ");
double width  = double.Parse(Console.ReadLine()!);

double area      = length * width;
double perimeter = 2 * (length + width);
double diagonal  = Math.Sqrt(length * length + width * width);

Console.WriteLine($"Area:      {area}");
Console.WriteLine($"Perimeter: {perimeter}");
Console.WriteLine($"Diagonal:  {diagonal:F2}");
```

With input validation (extension)

```
Console.Write("Enter length: ");
double length = double.Parse(Console.ReadLine()!);
Console.Write("Enter width: ");
double width  = double.Parse(Console.ReadLine()!);

if (length <= 0 || width <= 0) {
    Console.WriteLine("Dimensions must be positive.");
} else {
    Console.WriteLine($"Area:      {length * width:F2}");
    Console.WriteLine($"Perimeter: {2 * (length + width):F2}");
    Console.WriteLine($"Diagonal:  {Math.Sqrt(length * length + width * width):F2}");
}
```

Challenge 4 Odd or Even

PROBLEM

Ask the user for a whole number. Tell them whether it is odd or even.

EXAMPLE

```
Enter a number: 7
7 is odd.
```

MARK SCHEME

Correct use of modulo. Both cases handled. Output includes the original number.

EXTENSION

Also tell the user if the number is positive, negative, or zero.

TEACHER NOTES

> Introduces the modulo operator, which many students encounter for the first time here. Worth pausing to explain what remainder means before students start coding.

SOLUTIONS

Standard approach

```
Console.Write("Enter a number: ");
int number = int.Parse(Console.ReadLine()!);

if (number % 2 == 0) {
    Console.WriteLine($"{number} is even.");
} else {
    Console.WriteLine($"{number} is odd.");
}
```

With positive / negative / zero check (extension)

```
Console.Write("Enter a number: ");
int number = int.Parse(Console.ReadLine()!);

string parity = number % 2 == 0 ? "even" : "odd";
string sign;

if (number > 0) {
    sign = "positive";
} else if (number < 0) {
    sign = "negative";
} else {
    sign = "zero";
}

Console.WriteLine($"{number} is {parity} and {sign}.");
```

Challenge 5

Temperature Converter

PROBLEM

Write a program that converts a temperature from Celsius to Fahrenheit. Formula: F = (C x 9/5) + 32

EXAMPLE

```
Enter temperature in Celsius: 100
100 degrees C is 212.0 degrees F
```

MARK SCHEME

Correct formula. Output shows both values with units. Handles decimal input.

EXTENSION

Let the user choose the direction of conversion (C to F or F to C).

TEACHER NOTES

> Good for practising arithmetic and formatting. Students who finish quickly benefit from the extension, which introduces simple selection.

SOLUTIONS

Celsius to Fahrenheit

```
Console.Write("Enter temperature in Celsius: ");
double celsius    = double.Parse(Console.ReadLine()!);
double fahrenheit = celsius * 9.0 / 5.0 + 32;
Console.WriteLine($"{celsius} degrees C is {fahrenheit:F1} degrees F.");
```

Both directions (extension)

```
Console.Write("Convert (1) C to F or (2) F to C? ");
string choice = Console.ReadLine()!;

if (choice == "1") {
    Console.Write("Celsius: ");
    double c = double.Parse(Console.ReadLine()!);
    Console.WriteLine($"{c:F1}C = {c * 9.0 / 5.0 + 32:F1}F");
} else {
    Console.Write("Fahrenheit: ");
    double f = double.Parse(Console.ReadLine()!);
    Console.WriteLine($"{f:F1}F = {(f - 32) * 5.0 / 9.0:F1}C");
}
```

Challenge 6

Password Checker

PROBLEM

Ask the user to enter a password. Tell them if it is strong or weak. Rules: at least 8 characters, contains at least one number, contains at least one uppercase letter.

EXAMPLE

```
Enter a password: hello
Weak password. Must be at least 8 characters.
```

MARK SCHEME

All three rules checked independently. Specific feedback given for each failure. Passes when all three are met.

EXTENSION

Also require at least one special character (e.g. !, @, #).

TEACHER NOTES

> Students often try to use a single long if-statement instead of checking each rule separately. Encourage checking each condition independently so feedback is specific.

SOLUTIONS

Using LINQ helpers

```
Console.Write("Enter a password: ");
string password = Console.ReadLine()!;

if (password.Length < 8) {
    Console.WriteLine("Weak: must be at least 8 characters.");
} else if (!password.Any(char.IsDigit)) {
    Console.WriteLine("Weak: must contain at least one number.");
} else if (!password.Any(char.IsUpper)) {
    Console.WriteLine("Weak: must contain at least one uppercase letter.");
} else {
    Console.WriteLine("Strong password.");
}
```

Manual loops - without LINQ

```
Console.Write("Enter a password: ");
string password = Console.ReadLine()!;
bool hasDigit   = false;
bool hasUpper   = false;

foreach (char c in password) {
    if (char.IsDigit(c)) {
        hasDigit = true;
    }
    if (char.IsUpper(c)) {
        hasUpper = true;
    }
}

if (password.Length < 8) {
    Console.WriteLine("Weak: must be at least 8 characters.");
} else if (!hasDigit) {
    Console.WriteLine("Weak: must contain at least one number.");
} else if (!hasUpper) {
    Console.WriteLine("Weak: must contain at least one uppercase letter.");
} else {
    Console.WriteLine("Strong password.");
}
```

Challenge 7 Times Table

PROBLEM

Ask the user for a number. Display the times table for that number up to 12.

EXAMPLE

```
Enter a number: 6
6 x 1 = 6
6 x 2 = 12
...
6 x 12 = 72
```

MARK SCHEME

Correct loop range. Correct calculation. All 12 lines displayed in correct format.

EXTENSION

Ask the user how far they want the table to go (e.g. up to 20).

TEACHER NOTES

> A classic first for loop task. Students often use i <= 11 and miss the 12 times entry - worth discussing how loop bounds work before they start.

SOLUTIONS

Standard approach

```
Console.Write("Enter a number: ");
int n = int.Parse(Console.ReadLine()!);

for (int i = 1; i <= 12; i++) {
    Console.WriteLine($"{n} x {i} = {n * i}");
}
```

User-specified limit (extension)

```
Console.Write("Enter a number: ");
int n     = int.Parse(Console.ReadLine()!);
Console.Write("Show up to: ");
int limit = int.Parse(Console.ReadLine()!);

for (int i = 1; i <= limit; i++) {
    Console.WriteLine($"{n} x {i} = {n * i}");
}
```

Challenge 8

Number Guessing Game

PROBLEM

Generate a random number between 1 and 10. Ask the user to guess it. Tell them if they are correct, too high, or too low. Keep asking until they get it right.

EXAMPLE

```
Guess a number between 1 and 10: 5
Too low!
Guess a number between 1 and 10: 8
Too high!
Guess a number between 1 and 10: 7
Correct! You took 3 guesses.
```

MARK SCHEME

Random number generated. Loop continues until correct. High/low feedback given. Guess count displayed.

EXTENSION

Give the user a maximum of 5 guesses before revealing the answer.

TEACHER NOTES

> This task works well once students have covered loops and selection. Some students struggle to understand that the random number must be generated before the loop begins, not inside it.

SOLUTIONS

do-while loop — runs at least once

```
Random rng  = new();
int secret  = rng.Next(1, 11);
int guesses = 0;
int guess;

do {
    Console.Write("Guess a number between 1 and 10: ");
    guess = int.Parse(Console.ReadLine()!);
    guesses++;
    if (guess < secret) {
        Console.WriteLine("Too low!");
    } else if (guess > secret) {
        Console.WriteLine("Too high!");
    }
} while (guess != secret);

Console.WriteLine($"Correct! You took {guesses} guess{(guesses == 1 ? "" : "es")}.");
```

With maximum guess limit (extension)

```
Random rng      = new();
int secret      = rng.Next(1, 11);
int guesses     = 0;
int maxGuesses  = 5;
bool solved     = false;

while (guesses < maxGuesses && !solved) {
    Console.Write("Guess a number between 1 and 10: ");
    int guess = int.Parse(Console.ReadLine()!);
    guesses++;
    if (guess == secret) {
        Console.WriteLine($"Correct! You took {guesses} guess{(guesses == 1 ? "" : "es")}.");
        solved = true;
    } else if (guess < secret) {
        Console.WriteLine("Too low!");
    } else {
        Console.WriteLine("Too high!");
    }
}

if (!solved) {
    Console.WriteLine($"Out of guesses! The answer was {secret}.");
}
```

Challenge 9 FizzBuzz

PROBLEM

Print the numbers from 1 to 100. For multiples of 3 print Fizz, for multiples of 5 print Buzz, for multiples of both print FizzBuzz.

EXAMPLE

```
1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz...
```

MARK SCHEME

Correct output for all three cases. FizzBuzz case handled before the individual checks.

EXTENSION

Make the range and divisors configurable by user input.

TEACHER NOTES

> FizzBuzz is deliberately simple but tests whether students understand order of conditions in if/else if chains. Students who check 3 and 5 separately first will fail on 15 - worth discussing why as a class.

SOLUTIONS

Standard approach

```
for (int i = 1; i <= 100; i++) {
    if (i % 15 == 0) {
        Console.WriteLine("FizzBuzz");
    } else if (i % 3 == 0) {
        Console.WriteLine("Fizz");
    } else if (i % 5 == 0) {
        Console.WriteLine("Buzz");
    } else {
        Console.WriteLine(i);
    }
}
```

Configurable divisors (extension)

```
Console.Write("Upper limit: ");
int limit = int.Parse(Console.ReadLine()!);
Console.Write("First divisor: ");
int d1 = int.Parse(Console.ReadLine()!);
Console.Write("Second divisor: ");
int d2 = int.Parse(Console.ReadLine()!);

for (int i = 1; i <= limit; i++) {
    if (i % (d1 * d2) == 0) {
        Console.WriteLine("FizzBuzz");
    } else if (i % d1 == 0) {
        Console.WriteLine("Fizz");
    } else if (i % d2 == 0) {
        Console.WriteLine("Buzz");
    } else {
        Console.WriteLine(i);
    }
}
```

Challenge 10

Shopping Basket Total

PROBLEM

Ask the user to enter item prices one at a time. When they type 'done', display the total and the number of items.

EXAMPLE

```
Enter price (or 'done'): 3.99
Enter price (or 'done'): 1.50
Enter price (or 'done'): done
2 items. Total: 5.49
```

MARK SCHEME

Loop exits on 'done'. Correct total and item count. Handles at least one item entered.

EXTENSION

Apply a 10% discount if the total exceeds 20.

TEACHER NOTES

> A natural use of indefinite iteration. Students sometimes struggle with the sentinel value pattern - writing the loop condition and the string comparison in the same step can cause confusion.

SOLUTIONS

Sentinel-controlled loop

```
double total = 0;
int    count = 0;

Console.Write("Enter price (or 'done'): ");
string input = Console.ReadLine()!;

while (input.ToLower() != "done") {
    total += double.Parse(input);
    count++;
    Console.Write("Enter price (or 'done'): ");
    input = Console.ReadLine()!;
}

Console.WriteLine($"{count} items. Total: {total:F2}");
```

With 10% discount over £20 (extension)

```
double total = 0;
int    count = 0;

Console.Write("Enter price (or 'done'): ");
string input = Console.ReadLine()!;

while (input.ToLower() != "done") {
    total += double.Parse(input);
    count++;
    Console.Write("Enter price (or 'done'): ");
    input = Console.ReadLine()!;
}

if (total > 20) {
    double discount = total * 0.10;
    Console.WriteLine($"Discount applied: -£{discount:F2}");
    total -= discount;
}

Console.WriteLine($"{count} items. Total: £{total:F2}");
```

Challenge 11

Grade Classifier

PROBLEM

Ask for a student's exam score (0-100). Display the corresponding grade: 90-100=A*, 80-89=A, 70-79=B, 60-69=C, 50-59=D, below 50=U.

EXAMPLE

```
Enter score: 85
Grade: A
```

MARK SCHEME

All six grades correctly assigned. Invalid input handled with an error message.

EXTENSION

Ask for scores in five subjects. Display the average and overall grade.

TEACHER NOTES

> Students often write overlapping conditions without else if. Walk through what happens when a score of 85 hits each line - it helps them see why else if matters.

SOLUTIONS

Standard approach

```
Console.Write("Enter score: ");
int score = int.Parse(Console.ReadLine()!);

if (score < 0 || score > 100) {
    Console.WriteLine("Invalid score.");
} else if (score >= 90) {
    Console.WriteLine("Grade: A*");
} else if (score >= 80) {
    Console.WriteLine("Grade: A");
} else if (score >= 70) {
    Console.WriteLine("Grade: B");
} else if (score >= 60) {
    Console.WriteLine("Grade: C");
} else if (score >= 50) {
    Console.WriteLine("Grade: D");
} else {
    Console.WriteLine("Grade: U");
}
```

All students in a loop (extension)

```
string GetGrade(int s) {
    if (s < 0 || s > 100) { return "Invalid"; }
    if (s >= 90) { return "A*"; }
    if (s >= 80) { return "A";  }
    if (s >= 70) { return "B";  }
    if (s >= 60) { return "C";  }
    if (s >= 50) { return "D";  }
    return "U";
}

Console.Write("How many students? ");
int n = int.Parse(Console.ReadLine()!);

for (int i = 0; i < n; i++) {
    Console.Write($"Score {i + 1}: ");
    int score = int.Parse(Console.ReadLine()!);
    Console.WriteLine($"Grade: {GetGrade(score)}");
}
```

Challenge 12

Factorial

PROBLEM

Ask the user for a positive integer. Calculate and display its factorial.

EXAMPLE

```
Enter a number: 5
5! = 120
```

MARK SCHEME

Correct calculation. Handles 0! = 1. Rejects negative input with an error message.

EXTENSION

Write a recursive version of the function.

TEACHER NOTES

> A good introduction to accumulator patterns. Many students initialise their result variable at 0 instead of 1 - worth predicting what will happen before they run it.

SOLUTIONS

Iterative

```
Console.Write("Enter a number: ");
int n = int.Parse(Console.ReadLine()!);

if (n < 0) {
    Console.WriteLine("Please enter a positive integer.");
} else {
    long result = 1;
    for (int i = 2; i <= n; i++) {
        result *= i;
    }
    Console.WriteLine($"{n}! = {result}");
}
```

Recursive (extension)

```
Console.Write("Enter a number: ");
int n = int.Parse(Console.ReadLine()!);
Console.WriteLine($"{n}! = {Factorial(n)}");

long Factorial(int n) {
    if (n <= 1) {
        return 1;
    }
    return n * Factorial(n - 1);
}
```

Challenge 13 Countdown Timer

PROBLEM

Ask the user for a starting number. Count down to zero, displaying each number.

EXAMPLE

```
Start from: 5
5
4
3
2
1
Blast off!
```

MARK SCHEME

All numbers displayed in correct order. 'Blast off!' shown at end. Handles start of 0.

EXTENSION

Add a 1-second pause between each number using Thread.Sleep(1000).

TEACHER NOTES

> Useful for consolidating loops. Students using a for loop often forget to print 'Blast off!' after it ends - discuss the loop's exit condition explicitly.

SOLUTIONS

Using a while loop

```
Console.Write("Start from: ");
int i = int.Parse(Console.ReadLine()!);

while (i >= 1) {
    Console.WriteLine(i);
    i--;
}

Console.WriteLine("Blast off!");
```

Using a for loop

```
Console.Write("Start from: ");
int n = int.Parse(Console.ReadLine()!);

for (int i = n; i >= 1; i--) {
    Console.WriteLine(i);
}

Console.WriteLine("Blast off!");
```

Challenge 14 Sum of Digits

PROBLEM

Ask the user for a positive integer. Calculate the sum of its digits.

EXAMPLE

```
Enter a number: 1234
Sum of digits: 10
```

MARK SCHEME

Correct sum for any positive integer. Handles single-digit input correctly.

EXTENSION

Keep summing the digits until a single digit remains (digital root).

TEACHER NOTES

> This task introduces the idea of treating a number as a sequence of characters. Students who are comfortable with loops often find this satisfying because it combines two ideas they already know.

SOLUTIONS

Iterating over the string

```
Console.Write("Enter a number: ");
string input = Console.ReadLine()!;
int sum = 0;

foreach (char c in input) {
    sum += c - '0';
}

Console.WriteLine($"Sum of digits: {sum}");
```

Digital root — keep summing until single digit (extension)

```
Console.Write("Enter a number: ");
string input = Console.ReadLine()!;

int DigitSum(string s) {
    int total = 0;
    foreach (char c in s) {
        total += c - '0';
    }
    return total;
}

int result = int.Parse(input);
while (result >= 10) {
    result = DigitSum(result.ToString());
}
Console.WriteLine($"Digital root: {result}");
```

Challenge 15 Number to Words (1-9)

PROBLEM

Ask the user for a number between 1 and 9. Display it in words.

EXAMPLE

```
Enter a number (1-9): 7
seven
```

MARK SCHEME

Correct word for all nine numbers. Handles input outside 1-9 with an error message.

EXTENSION

Extend to cover 1-20.

TEACHER NOTES

> A gentle introduction to using arrays as lookup tables. Students who use if/else if chains for all nine cases have solved it correctly but should be shown the array approach as a cleaner alternative.

SOLUTIONS

Using an array lookup

```
string[] words = { "", "one", "two", "three", "four", "five",
                   "six", "seven", "eight", "nine" };

Console.Write("Enter a number (1-9): ");
int n = int.Parse(Console.ReadLine()!);

if (n >= 1 && n <= 9) {
    Console.WriteLine(words[n]);
} else {
    Console.WriteLine("Please enter a number between 1 and 9.");
}
```

Extended to 20 (extension)

```
string[] ones = { "", "one", "two", "three", "four", "five",
                  "six", "seven", "eight", "nine", "ten",
                  "eleven", "twelve", "thirteen", "fourteen", "fifteen",
                  "sixteen", "seventeen", "eighteen", "nineteen", "twenty" };

Console.Write("Enter a number (1-20): ");
int n = int.Parse(Console.ReadLine()!);

if (n >= 1 && n <= 20) {
    Console.WriteLine(ones[n]);
} else {
    Console.WriteLine("Out of range.");
}
```

Challenge 16

Average Calculator

PROBLEM

Ask the user to enter a series of numbers, one per line. When they type 'done', display the average.

EXAMPLE

```
Enter a number (or 'done'): 10
Enter a number (or 'done'): 20
Enter a number (or 'done'): done
Average: 15.0
```

MARK SCHEME

Correct average. Handles one number entered. Handles 'done' as first input gracefully.

EXTENSION

Also display the highest and lowest values entered.

TEACHER NOTES

> Common errors include dividing total by count before the loop ends, or failing to handle the case where no numbers are entered. Both make good discussion points.

SOLUTIONS

Sentinel-controlled loop

```
double total = 0;
int    count = 0;

Console.Write("Enter a number (or 'done'): ");
string input = Console.ReadLine()!;

while (input.ToLower() != "done") {
    total += double.Parse(input);
    count++;
    Console.Write("Enter a number (or 'done'): ");
    input = Console.ReadLine()!;
}

if (count > 0) {
    Console.WriteLine($"Average: {total / count:F1}");
} else {
    Console.WriteLine("No numbers entered.");
}
```

With highest and lowest (extension)

```
List<double> numbers = new();

Console.Write("Enter a number (or 'done'): ");
string input = Console.ReadLine()!;

while (input.ToLower() != "done") {
    numbers.Add(double.Parse(input));
    Console.Write("Enter a number (or 'done'): ");
    input = Console.ReadLine()!;
}

if (numbers.Count > 0) {
    double highest = numbers[0];
    double lowest  = numbers[0];
    double total   = 0;

    foreach (double n in numbers) {
        total += n;
        if (n > highest) { highest = n; }
        if (n < lowest)  { lowest  = n; }
    }

    Console.WriteLine($"Average: {total / numbers.Count:F2}");
    Console.WriteLine($"Highest: {highest}  Lowest: {lowest}");
} else {
    Console.WriteLine("No numbers entered.");
}
```

Challenge 17

Palindrome Checker

PROBLEM

Ask the user for a word. Tell them whether it is a palindrome (reads the same forwards and backwards).

EXAMPLE

```
Enter a word: racecar
racecar is a palindrome.
```

MARK SCHEME

Correct result for palindromes and non-palindromes. Case-insensitive comparison.

EXTENSION

Handle phrases (remove spaces and punctuation before checking). 'A man a plan a canal Panama' should return palindrome.

TEACHER NOTES

> Introduces string reversal in C#. Students may be surprised there is no built-in string.Reverse() - discussing why the Reverse().ToArray() pattern is needed is a useful moment.

SOLUTIONS

Using LINQ Reverse

```
Console.Write("Enter a word: ");
string word     = Console.ReadLine()!.ToLower();
string reversed = new string(word.Reverse().ToArray());

if (word == reversed) {
    Console.WriteLine($"{word} is a palindrome.");
} else {
    Console.WriteLine($"{word} is not a palindrome.");
}
```

Manual reversal — no LINQ

```
Console.Write("Enter a word: ");
string word = Console.ReadLine()!.ToLower();
string rev  = "";

for (int i = word.Length - 1; i >= 0; i--) {
    rev += word[i];
}

if (word == rev) {
    Console.WriteLine($"{word} is a palindrome.");
} else {
    Console.WriteLine($"{word} is not a palindrome.");
}
```

Phrase palindrome — ignores spaces and punctuation (extension)

```
Console.Write("Enter a phrase: ");
string phrase  = Console.ReadLine()!.ToLower();
string cleaned = new string(phrase.Where(char.IsLetter).ToArray());
string rev     = new string(cleaned.Reverse().ToArray());

if (cleaned == rev) {
    Console.WriteLine("Palindrome!");
} else {
    Console.WriteLine("Not a palindrome.");
}
```

Challenge 18

Coin Toss Simulator

PROBLEM

Simulate 100 coin tosses. Count and display the number of heads and tails.

EXAMPLE

```
Heads: 52
Tails: 48
```

MARK SCHEME

Exactly 100 tosses. Both results counted correctly. Totals add to 100.

EXTENSION

Ask the user how many tosses to simulate. Display results as a percentage.

TEACHER NOTES

> A fun entry point for the Random class. Students enjoy seeing that results vary each run and this can prompt a brief discussion about pseudorandom number generation.

SOLUTIONS

Standard approach

```
Random rng = new();
int heads  = 0;
int tails  = 0;

for (int i = 0; i < 100; i++) {
    if (rng.Next(0, 2) == 0) {
        heads++;
    } else {
        tails++;
    }
}

Console.WriteLine($"Heads: {heads}");
Console.WriteLine($"Tails: {tails}");
```

User-specified tosses with percentages (extension)

```
Console.Write("How many tosses? ");
int n      = int.Parse(Console.ReadLine()!);
Random rng = new();
int heads  = 0;
int tails  = 0;

for (int i = 0; i < n; i++) {
    if (rng.Next(0, 2) == 0) {
        heads++;
    } else {
        tails++;
    }
}

Console.WriteLine($"Heads: {heads} ({heads * 100.0 / n:F1}%)");
Console.WriteLine($"Tails: {tails} ({tails * 100.0 / n:F1}%)");
```

Challenge 19

BMI Calculator

PROBLEM

Ask for weight (kg) and height (m). Calculate and display BMI and the category. Formula: BMI = weight / (height x height) Categories: Below 18.5=Underweight, 18.5-24.9=Healthy, 25-29.9=Overweight, 30+=Obese.

EXAMPLE

```
Enter weight (kg): 70
Enter height (m): 1.75
BMI: 22.9 - Healthy
```

MARK SCHEME

Correct formula. All four categories correct. Output shows BMI value and category.

EXTENSION

Validate that height and weight are positive numbers.

TEACHER NOTES

> Consolidates float input and else if chains. Some students square the height using Math.Pow(height, 2) which is fine - worth acknowledging both approaches.

SOLUTIONS

Standard approach

```
Console.Write("Enter weight (kg): ");
double weight = double.Parse(Console.ReadLine()!);
Console.Write("Enter height (m): ");
double height = double.Parse(Console.ReadLine()!);

double bmi = Math.Round(weight / (height * height), 1);

string category;
if (bmi < 18.5) {
    category = "Underweight";
} else if (bmi < 25.0) {
    category = "Healthy";
} else if (bmi < 30.0) {
    category = "Overweight";
} else {
    category = "Obese";
}

Console.WriteLine($"BMI: {bmi} - {category}");
```

With validation (extension)

```
Console.Write("Enter weight (kg): ");
double weight = double.Parse(Console.ReadLine()!);
Console.Write("Enter height (m): ");
double height = double.Parse(Console.ReadLine()!);

if (weight <= 0 || height <= 0) {
    Console.WriteLine("Weight and height must be positive.");
} else {
    double bmi = Math.Round(weight / (height * height), 1);
    string category;

    if (bmi < 18.5) {
        category = "Underweight";
    } else if (bmi < 25.0) {
        category = "Healthy";
    } else if (bmi < 30.0) {
        category = "Overweight";
    } else {
        category = "Obese";
    }

    Console.WriteLine($"BMI: {bmi} - {category}");
}
```

Challenge 20

Number Pyramid

PROBLEM

Ask the user for a number n. Print a right-angled triangle of numbers n rows tall.

EXAMPLE

```
n=4:
1
1 2
1 2 3
1 2 3 4
```

MARK SCHEME

Correct shape and numbers. Works for any positive n.

EXTENSION

Print an inverted version (n rows down to 1 row).

TEACHER NOTES

> Introduces nested loops. Students need to understand that the inner loop bound depends on the outer loop variable - draw the structure on the board before they attempt it.

SOLUTIONS

Right-angled triangle

```
Console.Write("Enter n: ");
int n = int.Parse(Console.ReadLine()!);

for (int row = 1; row <= n; row++) {
    for (int col = 1; col <= row; col++) {
        Console.Write($"{col} ");
    }
    Console.WriteLine();
}
```

Inverted triangle (extension)

```
Console.Write("Enter n: ");
int n = int.Parse(Console.ReadLine()!);

for (int row = n; row >= 1; row--) {
    for (int col = 1; col <= row; col++) {
        Console.Write($"{col} ");
    }
    Console.WriteLine();
}
```

Challenge 21

Leap Year Checker

PROBLEM

Ask the user for a year. Tell them whether it is a leap year. Rules: divisible by 4, except centuries, unless divisible by 400.

EXAMPLE

```
Enter a year: 2000
2000 is a leap year.
```

MARK SCHEME

Correct result for all cases: 2000=leap, 1900=not, 2024=leap, 2023=not.

EXTENSION

Display all leap years between two years entered by the user.

TEACHER NOTES

> A good test of compound conditions. Test students with 1900 and 2000 specifically - those are the edge cases that catch out most solutions.

SOLUTIONS

Standard approach

```
Console.Write("Enter a year: ");
int year    = int.Parse(Console.ReadLine()!);
bool isLeap = (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);

if (isLeap) {
    Console.WriteLine($"{year} is a leap year.");
} else {
    Console.WriteLine($"{year} is not a leap year.");
}
```

All leap years in a range (extension)

```
Console.Write("Start year: ");
int start = int.Parse(Console.ReadLine()!);
Console.Write("End year: ");
int end   = int.Parse(Console.ReadLine()!);

for (int year = start; year <= end; year++) {
    bool isLeap = (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);
    if (isLeap) {
        Console.WriteLine(year);
    }
}
```

Challenge 22 Word Reverser

PROBLEM

Ask the user for a sentence. Display the sentence with the words in reverse order.

EXAMPLE

```
Enter a sentence: the quick brown fox
fox brown quick the
```

MARK SCHEME

Words in correct reverse order. Handles single words. Handles extra spaces.

EXTENSION

Also reverse the letters within each word.

TEACHER NOTES

> Introduces three important operations in one task. Students who use a loop to reverse the array are working correctly - show them Array.Reverse() as a cleaner alternative after.

SOLUTIONS

Using Split and Array.Reverse

```
Console.Write("Enter a sentence: ");
string   sentence = Console.ReadLine()!;
string[] words    = sentence.Split(' ');
Array.Reverse(words);
Console.WriteLine(string.Join(" ", words));
```

Manual reversal — no Array.Reverse

```
Console.Write("Enter a sentence: ");
string   sentence = Console.ReadLine()!;
string[] words    = sentence.Split(' ');
string   result   = "";

for (int i = words.Length - 1; i >= 0; i--) {
    if (result.Length > 0) {
        result += " ";
    }
    result += words[i];
}

Console.WriteLine(result);
```

Challenge 23

Multiplication Quiz

PROBLEM

Generate 10 random multiplication questions (numbers 1-12). Mark each answer right or wrong. Display the final score out of 10.

EXAMPLE

```
What is 7 x 8? 56
Correct!
What is 3 x 9? 25
Wrong! The answer was 27.
...
You scored 8 out of 10.
```

MARK SCHEME

10 questions generated. Correct/wrong feedback each time. Final score displayed.

EXTENSION

Time the quiz using a Stopwatch. Display how long it took.

TEACHER NOTES

> Students engage well with this task because the output is interactive. Watch out for students comparing a string input directly with an integer answer - int.Parse() conversion is a common omission.

SOLUTIONS

Standard approach

```
Random rng = new();
int    score = 0;

for (int i = 0; i < 10; i++) {
    int a = rng.Next(1, 13);
    int b = rng.Next(1, 13);
    Console.Write($"What is {a} x {b}? ");
    int answer = int.Parse(Console.ReadLine()!);
    if (answer == a * b) {
        Console.WriteLine("Correct!");
        score++;
    } else {
        Console.WriteLine($"Wrong! The answer was {a * b}.");
    }
}

Console.WriteLine($"You scored {score} out of 10.");
```

Timed quiz — records how long each answer takes (extension)

```
Random rng = new();
int score  = 0;

for (int i = 0; i < 5; i++) {
    int a = rng.Next(1, 13);
    int b = rng.Next(1, 13);
    Console.Write($"What is {a} x {b}? ");
    DateTime start  = DateTime.Now;
    int answer      = int.Parse(Console.ReadLine()!);
    double elapsed  = (DateTime.Now - start).TotalSeconds;

    if (answer == a * b) {
        Console.WriteLine($"Correct! ({elapsed:F1}s)");
        score++;
    } else {
        Console.WriteLine($"Wrong! The answer was {a * b}. ({elapsed:F1}s)");
    }
}

Console.WriteLine($"Final score: {score}/5");
```

Challenge 24

Currency Converter

PROBLEM

Ask the user for an amount in pounds. Display the equivalent in euros and US dollars using fixed conversion rates of your choice.

EXAMPLE

```
Enter amount in GBP: 50
58.50 euros
63.00 USD
```

MARK SCHEME

Correct calculations for both currencies. Symbols displayed. Handles decimal input.

EXTENSION

Let the user choose which currency to convert from and to.

TEACHER NOTES

A straightforward consolidation task. Encourage students to define the rates as named constants (const double) rather than hardcoding them inside the calculation.

SOLUTIONS

Standard approach

```
const double EUR_RATE = 1.17;
const double USD_RATE = 1.26;

Console.Write("Enter amount in GBP: ");
double amount = double.Parse(Console.ReadLine()!);

Console.WriteLine($"EUR: €{amount * EUR_RATE:F2}");
Console.WriteLine($"USD: ${amount * USD_RATE:F2}");
```

Choose currency interactively (extension)

```
var rates = new Dictionary<string, double> {
    ["EUR"] = 1.17,
    ["USD"] = 1.26,
    ["JPY"] = 195.3,
};

Console.Write("Enter amount in GBP: ");
double amount = double.Parse(Console.ReadLine()!);

foreach (var entry in rates) {
    Console.WriteLine($"{entry.Key}: {amount * entry.Value:F2}");
}
```

Challenge 25

Prime Number Checker

PROBLEM

Ask the user for a positive integer. Tell them whether it is prime.

EXAMPLE

```
Enter a number: 17
17 is prime.
```

MARK SCHEME

Correct result for primes and non-primes. Handles 1 (not prime) and 2 (prime) correctly.

EXTENSION

Display all prime numbers up to a user-specified limit.

TEACHER NOTES

> Students often check up to n-1, which works but is inefficient. Introduce the square root optimisation as an extension discussion for stronger students.

SOLUTIONS

Trial division

```
Console.Write("Enter a number: ");
int n = int.Parse(Console.ReadLine()!);

if (n < 2) {
    Console.WriteLine($"{n} is not prime.");
} else {
    bool isPrime = true;
    int i = 2;
    while (i <= (int)Math.Sqrt(n) && isPrime) {
        if (n % i == 0) {
            isPrime = false;
        }
        i++;
    }
    if (isPrime) {
        Console.WriteLine($"{n} is prime.");
    } else {
        Console.WriteLine($"{n} is not prime.");
    }
}
```

All primes up to limit — using a helper function (extension)

```
Console.Write("Show primes up to: ");
int limit = int.Parse(Console.ReadLine()!);

for (int n = 2; n <= limit; n++) {
    if (IsPrime(n)) {
        Console.Write($"{n} ");
    }
}
Console.WriteLine();

bool IsPrime(int n) {
    if (n < 2) { return false; }
    for (int i = 2; i <= (int)Math.Sqrt(n); i++) {
        if (n % i == 0) { return false; }
    }
    return true;
}
```

Intermediate

Functions, lists, strings, OOP basics, and classic algorithms. Core secondary and lower A-Level.

Challenge 26 Vowel Counter

PROBLEM

Ask the user for a sentence. Count and display the number of vowels.

EXAMPLE

```
Enter a sentence: Hello World
Number of vowels: 3
```

MARK SCHEME

All five vowels counted (case-insensitive). Correct count for any input.

EXTENSION

Display a breakdown by vowel (a: 1, e: 1, o: 1...).

TEACHER NOTES

> Good for consolidating iteration over strings. Students who call .ToLower() on the character before comparing against a single-case vowel string show good thinking.

SOLUTIONS

Standard approach

```
Console.Write("Enter a sentence: ");
string sentence = Console.ReadLine()!;
int count = 0;

foreach (char c in sentence.ToLower()) {
    if ("aeiou".Contains(c)) {
        count++;
    }
}

Console.WriteLine($"Number of vowels: {count}");
```

Breakdown by vowel (extension)

```
Console.Write("Enter a sentence: ");
string sentence = Console.ReadLine()!.ToLower();

var counts = new Dictionary<char, int> {
    ['a'] = 0, ['e'] = 0, ['i'] = 0, ['o'] = 0, ['u'] = 0
};

foreach (char c in sentence) {
    if (counts.ContainsKey(c)) {
        counts[c]++;
    }
}

foreach (var kv in counts) {
    Console.WriteLine($"{kv.Key}: {kv.Value}");
}
```

Challenge 27

List Maximum and Minimum

PROBLEM

Ask the user to enter 10 numbers. Display the largest and smallest without using built-in Max() or Min() methods.

EXAMPLE

```
After entering 10 numbers:
Largest: 97
Smallest: 3
```

MARK SCHEME

Does not use built-in Max/Min. Correct values identified for any 10 numbers.

EXTENSION

Sort the list without using built-in Sort() or OrderBy().

TEACHER NOTES

> The constraint on not using built-ins is important - it forces students to think about the algorithm. Students who initialise their tracker at 0 will fail if all numbers are negative.

SOLUTIONS

Manual max and min — no built-ins

```
List<double> numbers = new();

for (int i = 0; i < 10; i++) {
    Console.Write($"Enter number {i + 1}: ");
    numbers.Add(double.Parse(Console.ReadLine()!));
}

double largest  = numbers[0];
double smallest = numbers[0];

foreach (double n in numbers) {
    if (n > largest)  { largest  = n; }
    if (n < smallest) { smallest = n; }
}

Console.WriteLine($"Largest:  {largest}");
Console.WriteLine($"Smallest: {smallest}");
```

Manual bubble sort (extension)

```
List<double> nums = new();

for (int i = 0; i < 10; i++) {
    Console.Write($"Enter number {i + 1}: ");
    nums.Add(double.Parse(Console.ReadLine()!));
}

int n = nums.Count;
for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - 1 - i; j++) {
        if (nums[j] > nums[j + 1]) {
            (nums[j], nums[j + 1]) = (nums[j + 1], nums[j]);
        }
    }
}

Console.WriteLine("Sorted: " + string.Join(", ", nums));
```

Challenge 28

Student Grade Book

PROBLEM

Ask the teacher to enter student names and scores. When they type 'done', display each student with their grade, the class average, and the top scorer.

EXAMPLE

```
Enter name (or 'done'): Alice
Enter score: 78
...
Class average: 74.5
Top scorer: Alice (78)
```

MARK SCHEME

Names and grades stored and displayed. Average calculated correctly. Top scorer identified.

EXTENSION

Display results sorted by score (highest first).

TEACHER NOTES

> This task works well after students have learned lists and loops. Using two parallel lists versus a list of tuples is worth discussing as a design choice.

SOLUTIONS

Parallel lists — sentinel loop

```
List<string> names  = new();
List<int>    scores = new();

Console.Write("Enter name (or 'done'): ");
string name = Console.ReadLine()!;

while (name.ToLower() != "done") {
    Console.Write("Enter score: ");
    scores.Add(int.Parse(Console.ReadLine()!));
    names.Add(name);
    Console.Write("Enter name (or 'done'): ");
    name = Console.ReadLine()!;
}

if (names.Count > 0) {
    double average = 0;
    int    topIdx  = 0;
    for (int i = 0; i < scores.Count; i++) {
        average += scores[i];
        if (scores[i] > scores[topIdx]) { topIdx = i; }
    }
    average /= scores.Count;
    for (int i = 0; i < names.Count; i++) {
        Console.WriteLine($"{names[i]}: {scores[i]}");
    }
    Console.WriteLine($"Class average: {average:F1}");
    Console.WriteLine($"Top student: {names[topIdx]} ({scores[topIdx]})");
} else {
    Console.WriteLine("No students entered.");
}
```

Challenge 29

Caesar Cipher

PROBLEM

Encrypt a message using the Caesar cipher. Ask for the message and the shift value.

EXAMPLE

```
Message: hello
Shift: 3
Encrypted: khoor
```

MARK SCHEME

Correct encryption for all lowercase letters. Non-letter characters unchanged. Wrap-around handled.

EXTENSION

Add a decryption function. Allow uppercase letters.

TEACHER NOTES

> Links well with theory lessons on encryption. The wrap-around is the tricky part - students who get most letters right but fail on x, y, z have probably missed the modulo.

SOLUTIONS

Standard approach

```
Console.Write("Enter a sentence: ");
string sentence = Console.ReadLine()!;
var    wordFreq  = new Dictionary<string, int>();

foreach (string word in sentence.ToLower().Split(' ')) {
    string w = word.Trim('.', ',', '!', '?');
    if (w.Length > 0) {
        if (wordFreq.ContainsKey(w)) {
            wordFreq[w]++;
        } else {
            wordFreq[w] = 1;
        }
    }
}

foreach (var entry in wordFreq) {
    Console.WriteLine($"{entry.Key}: {entry.Value}");
}
```

Sorted by frequency (extension)

```
Console.Write("Enter a sentence: ");
string sentence = Console.ReadLine()!;
var    wordFreq  = new Dictionary<string, int>();

foreach (string word in sentence.ToLower().Split(' ')) {
    string w = word.Trim('.', ',', '!', '?');
    if (w.Length > 0) {
        wordFreq.TryGetValue(w, out int v);
        wordFreq[w] = v + 1;
    }
}

var sorted = wordFreq.OrderByDescending(kv => kv.Value);
foreach (var entry in sorted) {
    Console.WriteLine($"{entry.Key}: {entry.Value}");
}
```

Challenge 30 Word Frequency Counter

PROBLEM

Ask the user for a sentence. Display how many times each unique word appears.

EXAMPLE

```
Enter a sentence: the cat sat on the mat near the cat
the: 3
cat: 2
sat: 1
```

MARK SCHEME

Correct counts for all words. Case-insensitive. All unique words displayed.

EXTENSION

Sort output by frequency (most common first).

TEACHER NOTES

> An ideal first Dictionary task. Students who use a list and count occurrences are working harder than they need to - this is a good moment to motivate why dictionaries exist.

SOLUTIONS

Standard approach

```
Console.Write("Enter a string: ");
string input    = Console.ReadLine()!;
string reversed = new string(input.Reverse().ToArray());

Console.WriteLine($"Original: {input}");
Console.WriteLine($"Reversed: {reversed}");
Console.WriteLine($"Same:     {input == reversed}");
```

Manual reversal — no LINQ

```
Console.Write("Enter a string: ");
string input = Console.ReadLine()!;
string rev   = "";

for (int i = input.Length - 1; i >= 0; i--) {
    rev += input[i];
}

Console.WriteLine($"Original: {input}");
Console.WriteLine($"Reversed: {rev}");
Console.WriteLine($"Same:     {input == rev}");
```

Challenge 31 Rock Paper Scissors

PROBLEM

Implement a best-of-three Rock Paper Scissors game against the computer.

EXAMPLE

```
Enter Rock, Paper, or Scissors: Rock
Computer chose Paper.
Computer wins this round!
Score: You 0 - Computer 1
```

MARK SCHEME

All three outcomes correct. Best-of-three tracked. Final result displayed.

EXTENSION

Add Lizard and Spock (Big Bang Theory version).

TEACHER NOTES

> Students enjoy this task. The condition logic for all nine combinations is the main challenge - encourage them to map out all possible matchups before coding.

SOLUTIONS

Standard approach

```
Console.Write("Enter a number: ");
int n = int.Parse(Console.ReadLine()!);
int a = 0;
int b = 1;

Console.Write($"{a} {b}");
int count = 2;

while (count < n) {
    int next = a + b;
    Console.Write($" {next}");
    a = b;
    b = next;
    count++;
}

Console.WriteLine();
```

Recursive (extension)

```
Console.Write("How many terms? ");
int n = int.Parse(Console.ReadLine()!);

for (int i = 0; i < n; i++) {
    Console.Write($"{Fib(i)} ");
}
Console.WriteLine();

int Fib(int n) {
    if (n <= 1) { return n; }
    return Fib(n - 1) + Fib(n - 2);
}
```

Challenge 32

Function: Is Prime

PROBLEM

Write a local function IsPrime(int n) that returns true if n is prime, false otherwise. Use it to print all prime numbers up to 100.

EXAMPLE

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

MARK SCHEME

Function defined correctly. Returns bool. Correct primes up to 100 displayed.

EXTENSION

Use the Sieve of Eratosthenes algorithm instead.

TEACHER NOTES

> Works well as a first local function task because the function has a clear, testable purpose. Students who print inside the function rather than returning a value need guidance on the return concept.

SOLUTIONS

Caesar cipher — encrypt and decrypt

```
Console.Write("Enter text: ");
string text = Console.ReadLine()!;
Console.Write("Enter shift: ");
int shift = int.Parse(Console.ReadLine()!);

string encrypted = Caesar(text, shift);
string decrypted  = Caesar(encrypted, -shift);

Console.WriteLine($"Encrypted: {encrypted}");
Console.WriteLine($"Decrypted: {decrypted}");

string Caesar(string text, int shift) {
    string result = "";
    foreach (char c in text) {
        if (char.IsLetter(c)) {
            char baseChar = char.IsUpper(c) ? 'A' : 'a';
            result += (char)(((c - baseChar + shift % 26 + 26) % 26) + baseChar);
        } else {
            result += c;
        }
    }
    return result;
}
```

Brute-force all 26 shifts (extension)

```
Console.Write("Enter ciphertext: ");
string text = Console.ReadLine()!;

for (int shift = 1; shift <= 25; shift++) {
    string result = "";
    foreach (char c in text) {
        if (char.IsLetter(c)) {
            char baseChar = char.IsUpper(c) ? 'A' : 'a';
            result += (char)(((c - baseChar + shift) % 26) + baseChar);
        } else {
            result += c;
        }
    }
    Console.WriteLine($"Shift {shift,2}: {result}");
}
```

Challenge 33

Fibonacci Sequence

PROBLEM

Write a function that returns the first n Fibonacci numbers as a list.

EXAMPLE

```
How many? 8
0, 1, 1, 2, 3, 5, 8, 13
```

MARK SCHEME

Correct sequence. Function returns a List<int>. Handles n=1 and n=2 correctly.

EXTENSION

Write a recursive version. Compare outputs with the iterative version.

TEACHER NOTES

> Check that students handle n=1 (returns [0]) and n=2 (returns [0, 1]) as edge cases. The extension to recursion works well as a lead-in to later recursive challenges.

SOLUTIONS

Standard approach

```
int[] numbers = { 5, 3, 8, 1, 9, 2, 7 };
Console.WriteLine("Before: " + string.Join(", ", numbers));

SelectionSort(numbers);
Console.WriteLine("After:  " + string.Join(", ", numbers));

void SelectionSort(int[] lst) {
    int n = lst.Length;
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++) {
            if (lst[j] < lst[minIdx]) {
                minIdx = j;
            }
        }
        if (minIdx != i) {
            (lst[i], lst[minIdx]) = (lst[minIdx], lst[i]);
        }
    }
}
```

Counting comparisons and swaps (extension)

```
int[] numbers = { 5, 3, 8, 1, 9, 2, 7 };
var (comps, swaps) = SelectionSortCount(numbers);
Console.WriteLine("Sorted:      " + string.Join(", ", numbers));
Console.WriteLine($"Comparisons: {comps}   Swaps: {swaps}");

(int comps, int swaps) SelectionSortCount(int[] lst) {
    int n = lst.Length;
    int comparisons = 0;
    int swapCount   = 0;
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++) {
            comparisons++;
            if (lst[j] < lst[minIdx]) { minIdx = j; }
        }
        if (minIdx != i) {
            (lst[i], lst[minIdx]) = (lst[minIdx], lst[i]);
            swapCount++;
        }
    }
    return (comparisons, swapCount);
}
```

Challenge 34

ATM Simulator

PROBLEM

Simulate an ATM. Start with a balance of 500. Allow the user to: check balance, deposit, withdraw, or quit. Validate that withdrawals do not exceed the balance.

EXAMPLE

```
1. Check balance
2. Deposit
3. Withdraw
4. Quit
Choice: 3
Withdraw: 600
Insufficient funds.
```

MARK SCHEME

All four options work correctly. Validation prevents overdraft. Loop continues until quit.

EXTENSION

Set a daily withdrawal limit of 300. Track total withdrawn in the current session.

TEACHER NOTES

> A good structured task that requires combining loops, functions, and conditionals. Students who hard-code the menu into one block can be encouraged to break it into local functions.

SOLUTIONS

Stack using a List

```
var stack = new List<int>();

Push(10); Push(20); Push(30);
Console.WriteLine($"Peek: {Peek()}");
Console.WriteLine($"Pop:  {Pop()}");
Console.WriteLine($"Pop:  {Pop()}");
Console.WriteLine($"Size: {stack.Count}");

void  Push(int v)  => stack.Add(v);
int   Pop()        { int v = stack[^1]; stack.RemoveAt(stack.Count - 1); return v; }
int   Peek()       => stack[^1];
```

Using the built-in Stack<T> class

```
var stack = new Stack<int>();

stack.Push(10);
stack.Push(20);
stack.Push(30);

Console.WriteLine($"Peek: {stack.Peek()}");
Console.WriteLine($"Pop:  {stack.Pop()}");
Console.WriteLine($"Pop:  {stack.Pop()}");
Console.WriteLine($"Size: {stack.Count}");
```

Challenge 35 String Statistics

PROBLEM

Write a function that takes a string and returns: length, number of words, number of vowels, and number of uppercase letters.

EXAMPLE

```
Input: 'Hello World'
Length: 11, Words: 2, Vowels: 3, Uppercase: 2
```

MARK SCHEME

Correct values for all four statistics. Function takes a string parameter and returns all results.

EXTENSION

Also return the most common character.

TEACHER NOTES

> A useful consolidation of string methods in one task. In C#, returning multiple values using a tuple (int, int, int, int) is a clean approach worth discussing.

SOLUTIONS

Queue using a List

```
var queue = new List<string>();

Enqueue("Alice"); Enqueue("Bob"); Enqueue("Charlie");
Console.WriteLine($"Front: {Peek()!}");
Console.WriteLine($"Dequeued: {Dequeue()!}");
Console.WriteLine($"Dequeued: {Dequeue()!}");

void    Enqueue(string v)   => queue.Add(v);
string? Dequeue()           { if (queue.Count == 0) return null; var v = queue[0];
queue.RemoveAt(0); return v; }
string? Peek()              => queue.Count == 0 ? null : queue[0];
```

Using the built-in Queue<T> class

```
var queue = new Queue<string>();

queue.Enqueue("Alice");
queue.Enqueue("Bob");
queue.Enqueue("Charlie");

Console.WriteLine($"Front: {queue.Peek()}");
Console.WriteLine($"Dequeued: {queue.Dequeue()}");
Console.WriteLine($"Dequeued: {queue.Dequeue()}");
Console.WriteLine($"Remaining: {queue.Count}");
```

Challenge 36 Number to Roman Numerals

PROBLEM

Convert a number between 1 and 3999 to Roman numerals.

EXAMPLE

```
Enter a number: 2024
MMXXIV
```

MARK SCHEME

Correct conversion for all values 1-3999. Handles subtractive notation (IV, IX, XL, XC, CD, CM).

EXTENSION

Convert Roman numerals back to a decimal number.

TEACHER NOTES

> The subtractive notation cases (IV, IX etc.) trip up students who only map single symbols. Walk through IV=4 before they start and ask them to think about how to represent it.

SOLUTIONS

Simple hash table with chaining

```
const int SIZE = 10;
var table = new List<(string key, string value)>[SIZE];
for (int i = 0; i < SIZE; i++) {
    table[i] = new List<(string, string)>();
}

Insert("name",  "Alice");
Insert("grade", "A");
Insert("age",   "17");
Console.WriteLine(Lookup("name"));   // Alice
Console.WriteLine(Lookup("grade"));  // A

int Hash(string key) {
    int hash = 0;
    foreach (char c in key) { hash = (hash * 31 + c) % SIZE; }
    return hash;
}

void Insert(string key, string value) {
    int idx = Hash(key);
    var bucket = table[idx];
    for (int i = 0; i < bucket.Count; i++) {
        if (bucket[i].key == key) {
            bucket[i] = (key, value);
            return;
        }
    }
    bucket.Add((key, value));
}

string? Lookup(string key) {
    int idx = Hash(key);
    foreach (var (k, v) in table[idx]) {
        if (k == key) { return v; }
    }
    return null;
}
```

Challenge 37

Stack Implementation

PROBLEM

Implement a stack using a List<T>. Provide Push, Pop, Peek, and IsEmpty operations as local functions.

EXAMPLE

```
Push(5), Push(3), Push(9)
Peek() -> 9
Pop() -> 9
Peek() -> 3
```

MARK SCHEME

All four operations work correctly. Pop() and Peek() handle empty stack gracefully.

EXTENSION

Use your stack to check if brackets in a string are balanced, e.g. {[()]} is balanced.

TEACHER NOTES

> Best taught alongside theory content on stacks and queues. The Add/RemoveAt mapping to push/pop is worth making explicit.

SOLUTIONS

Recursive DFS and BFS

```
var graph = new Dictionary<string, List<string>> {
    ["A"] = new() { "B", "C" },
    ["B"] = new() { "A", "D", "E" },
    ["C"] = new() { "A", "F" },
    ["D"] = new() { "B" },
    ["E"] = new() { "B" },
    ["F"] = new() { "C" },
};

Console.Write("DFS: "); DFS("A", new HashSet<string>());
Console.WriteLine();

Console.Write("BFS: ");
var visited = new HashSet<string>();
var queue   = new Queue<string>();
queue.Enqueue("A");
visited.Add("A");
while (queue.Count > 0) {
    string node = queue.Dequeue();
    Console.Write(node + " ");
    foreach (string neighbour in graph[node]) {
        if (!visited.Contains(neighbour)) {
            visited.Add(neighbour);
            queue.Enqueue(neighbour);
        }
    }
}
Console.WriteLine();

void DFS(string node, HashSet<string> seen) {
    seen.Add(node);
    Console.Write(node + " ");
    foreach (string neighbour in graph[node]) {
        if (!seen.Contains(neighbour)) {
            DFS(neighbour, seen);
        }
    }
}
```

Challenge 38 Queue Implementation

PROBLEM

Implement a queue using a List<T>. Provide Enqueue, Dequeue, Peek, and IsEmpty operations.

EXAMPLE

```
Enqueue('A'), Enqueue('B'), Enqueue('C')
Dequeue() -> 'A'
Peek() -> 'B'
```

MARK SCHEME

Correct FIFO behaviour. Empty queue handled gracefully on Dequeue and Peek.

EXTENSION

Simulate a printer queue: add jobs, process them in order, display the status.

TEACHER NOTES

> Pair this with challenge 37 for a theory and programming double lesson. The printer queue extension maps the abstract structure to a real-world use case that students can reason about.

SOLUTIONS

Manual queue implementation

```
var queue = new List<string>();

Enqueue(queue, "A"); Enqueue(queue, "B"); Enqueue(queue, "C");
Console.WriteLine(Dequeue(queue)!);  // A
Console.WriteLine(Peek(queue)!);     // B
Console.WriteLine($"Empty: {IsEmpty(queue)}");

void    Enqueue(List<string> q, string v) => q.Add(v);
string? Dequeue(List<string> q)  { if (IsEmpty(q)) { return null; } var v = q[0]; q.RemoveAt(0);
return v; }
string? Peek(List<string> q)     => IsEmpty(q) ? null : q[0];
bool    IsEmpty(List<string> q)  => q.Count == 0;
```

Challenge 39 Anagram Checker

PROBLEM

Write a function that takes two words and returns true if they are anagrams of each other.

EXAMPLE

```
listen / silent -> true
hello / world -> false
```

MARK SCHEME

Correct result for anagrams and non-anagrams. Case-insensitive. Handles spaces.

EXTENSION

Find all anagrams of a given word from a provided word list.

TEACHER NOTES

> A neat task that demonstrates the power of sorting as a problem-solving technique. Students who compare character counts in a dictionary also get a working and arguably more efficient solution.

SOLUTIONS

Sort and compare

```
Console.WriteLine(IsAnagram("listen", "silent"));  // True
Console.WriteLine(IsAnagram("hello",  "world"));   // False

bool IsAnagram(string w1, string w2) {
    w1 = w1.ToLower().Replace(" ", "");
    w2 = w2.ToLower().Replace(" ", "");
    char[] a = w1.ToCharArray();
    char[] b = w2.ToCharArray();
    Array.Sort(a);
    Array.Sort(b);
    return new string(a) == new string(b);
}
```

Manual frequency count — without Sort

```
Console.WriteLine(IsAnagram("listen", "silent"));  // True
Console.WriteLine(IsAnagram("hello",  "world"));   // False

bool IsAnagram(string w1, string w2) {
    w1 = w1.ToLower().Replace(" ", "");
    w2 = w2.ToLower().Replace(" ", "");
    if (w1.Length != w2.Length) { return false; }

    var counts = new Dictionary<char, int>();
    foreach (char c in w1) {
        counts.TryGetValue(c, out int v);
        counts[c] = v + 1;
    }
    foreach (char c in w2) {
        if (!counts.ContainsKey(c) || counts[c] == 0) { return false; }
        counts[c]--;
    }
    return true;
}
```

Challenge 40

Number Base Converter

PROBLEM

Write a program that converts a decimal number to binary, octal, and hexadecimal - without using C#'s built-in Convert.ToString(n, base).

EXAMPLE

```
Enter a decimal number: 255
Binary: 11111111
Octal: 377
Hexadecimal: FF
```

MARK SCHEME

Correct conversion for all three bases using the division algorithm. Handles 0.

EXTENSION

Convert in the other direction: binary, octal, and hex inputs to decimal.

TEACHER NOTES

> Works very well alongside binary and hexadecimal theory content. The restriction on built-in conversion forces students to understand the algorithm.

SOLUTIONS

Manual conversion — no Convert.ToString

```
Console.Write("Enter a decimal number: ");
int n = int.Parse(Console.ReadLine()!);
Console.WriteLine($"Binary:      {ToBase(n, 2)}");
Console.WriteLine($"Octal:       {ToBase(n, 8)}");
Console.WriteLine($"Hexadecimal: {ToBase(n, 16)}");

string ToBase(int n, int base_) {
    if (n == 0) { return "0"; }
    const string digits = "0123456789ABCDEF";
    string result = "";
    while (n > 0) {
        result = digits[n % base_] + result;
        n /= base_;
    }
    return result;
}
```

Using Convert.ToString (extension)

```
Console.Write("Enter a decimal number: ");
int n = int.Parse(Console.ReadLine()!);
Console.WriteLine($"Binary:      {Convert.ToString(n, 2)}");
Console.WriteLine($"Octal:       {Convert.ToString(n, 8)}");
Console.WriteLine($"Hexadecimal: {Convert.ToString(n, 16).ToUpper()}");
```

Challenge 41

Highest Common Factor

PROBLEM

Write a function to find the highest common factor (HCF) of two numbers using the Euclidean algorithm.

EXAMPLE

```
HCF of 48 and 18: 6
```

MARK SCHEME

Correct HCF for all valid inputs. Implements the Euclidean algorithm rather than brute force.

EXTENSION

Also calculate the lowest common multiple using the HCF.

TEACHER NOTES

> A good introduction to algorithm implementation. Students who use brute force (checking all numbers from 1 to n) arrive at a correct answer but should be shown why the Euclidean approach is better.

SOLUTIONS

Euclidean algorithm — iterative

```
Console.Write("First number: ");
int a = int.Parse(Console.ReadLine()!);
Console.Write("Second number: ");
int b = int.Parse(Console.ReadLine()!);

int hcf = Hcf(a, b);
long lcm = (long)a * b / hcf;
Console.WriteLine($"HCF: {hcf}");
Console.WriteLine($"LCM: {lcm}");

int Hcf(int a, int b) {
    while (b != 0) {
        (a, b) = (b, a % b);
    }
    return a;
}
```

Euclidean algorithm — recursive

```
Console.WriteLine(Hcf(48, 18));  // 6

int Hcf(int a, int b) {
    if (b == 0) { return a; }
    return Hcf(b, a % b);
}
```

Challenge 42 Sentence Scrambler

PROBLEM

Take a sentence and scramble the letters within each word, keeping the first and last letters in place.

EXAMPLE

```
the quick brown fox -> the qicuk bwron fox
```

MARK SCHEME

First and last letters unchanged. Middle letters randomised. Words of 3 or fewer characters left unchanged.

EXTENSION

Ask the user if they can still read the scrambled sentence. Build a short quiz around it.

TEACHER NOTES

> An engaging task with a fun result. Students often forget to handle short words (3 characters or fewer) correctly. Testing with a single-letter word is a good edge case to discuss.

SOLUTIONS

Standard approach

```
Random rng = new();
Console.Write("Enter a sentence: ");
string sentence  = Console.ReadLine()!;
string[] words   = sentence.Split(' ');
string scrambled = string.Join(" ", words.Select(ScrambleWord));
Console.WriteLine(scrambled);

string ScrambleWord(string word) {
    if (word.Length <= 3) { return word; }
    var middle = word[1..^1].ToList().OrderBy(_ => rng.Next()).ToList();
    return word[0] + string.Concat(middle) + word[^1];
}
```

Manual shuffle — no LINQ OrderBy

```
Random rng = new();
Console.Write("Enter a sentence: ");
string sentence = Console.ReadLine()!;
string[] words  = sentence.Split(' ');

for (int w = 0; w < words.Length; w++) {
    string word = words[w];
    if (word.Length > 3) {
        char[] middle = word[1..^1].ToCharArray();
        for (int i = middle.Length - 1; i > 0; i--) {
            int j = rng.Next(i + 1);
            (middle[i], middle[j]) = (middle[j], middle[i]);
        }
        words[w] = word[0] + new string(middle) + word[^1];
    }
}

Console.WriteLine(string.Join(" ", words));
```

Challenge 43

Matrix Addition

PROBLEM

Create two 3x3 matrices as 2D arrays. Write a function to add them and display the result.

EXAMPLE

```
[[1,2,3],[4,5,6],[7,8,9]] + [[9,8,7],[6,5,4],[3,2,1]] = [[10,10,10],[10,10,10],
[10,10,10]]
```

MARK SCHEME

Correct element-wise addition. Result displayed in grid format. Function takes two matrices as parameters.

EXTENSION

Write functions for matrix subtraction and matrix multiplication.

TEACHER NOTES

> Use this task to introduce 2D arrays. Students benefit from seeing a 3x3 grid drawn on the board alongside the array representation before they start coding.

SOLUTIONS

Matrix addition with nested loops

```
int[,] A = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
int[,] B = { {9, 8, 7}, {6, 5, 4}, {3, 2, 1} };
int[,] C = AddMatrices(A, B);

for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        Console.Write($"{C[i, j]} ");
    }
    Console.WriteLine();
}

int[,] AddMatrices(int[,] a, int[,] b) {
    int rows   = a.GetLength(0);
    int cols   = a.GetLength(1);
    var result = new int[rows, cols];
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[i, j] = a[i, j] + b[i, j];
        }
    }
    return result;
}
```

Challenge 44

Bank Account Class

PROBLEM

Create a BankAccount class with fields for account holder name and balance. Add methods for Deposit, Withdraw, and Display. Validate that withdrawals do not exceed the balance.

EXAMPLE

```
var account = new BankAccount("Alice", 500);
account.Deposit(200);
account.Withdraw(100);
account.Display();  // Balance: 600
```

MARK SCHEME

Class defined with constructor. Correct Deposit/Withdraw methods. Validation prevents negative balance.

EXTENSION

Add a transaction history List<string>. Add a PrintStatement() method.

TEACHER NOTES

> A good first OOP task because the real-world analogy is strong. Students often confuse the class definition with creating an object - make sure they write and run the instantiation line in the same file.

SOLUTIONS

BankAccount class

```
var account = new BankAccount("Alice", 500);
account.Deposit(200);
account.Withdraw(100);
account.Display();

class BankAccount {
    string       holder;
    double       balance;
    List<string> history = new();

    public BankAccount(string holder, double balance = 0) {
        this.holder  = holder;
        this.balance = balance;
    }

    public void Deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            history.Add($"+£{amount:F2}");
        }
    }

    public void Withdraw(double amount) {
        if (amount > balance) {
            Console.WriteLine("Insufficient funds.");
        } else {
            balance -= amount;
            history.Add($"-£{amount:F2}");
        }
    }

    public void Display() {
        Console.WriteLine($"{holder}: Balance £{balance:F2}");
        Console.WriteLine("History: " + string.Join(", ", history));
    }
}
```

Challenge 45 Linear Search

PROBLEM

Write a function LinearSearch(List<string> lst, string target) that returns the index of the target or -1 if not found.

EXAMPLE

```
LinearSearch(["Alice", "Bob", "Charlie"], "Bob") -> 1
LinearSearch(["Alice", "Bob"], "Dave") -> -1
```

MARK SCHEME

Searches element by element. Returns correct index or -1. Does not use built-in IndexOf().

EXTENSION

Count the number of comparisons made and display it alongside the result.

TEACHER NOTES

> Pair with theory on searching algorithms. The comparison counter in the extension leads naturally into a discussion of efficiency and Big O notation at A-Level.

SOLUTIONS

Linear search — no IndexOf

```
var names = new List<string> { "Alice", "Bob", "Charlie", "Dave" };
Console.WriteLine(LinearSearch(names, "Bob"));   // 1
Console.WriteLine(LinearSearch(names, "Eve"));   // -1

int LinearSearch(List<string> lst, string target) {
    for (int i = 0; i < lst.Count; i++) {
        if (lst[i] == target) { return i; }
    }
    return -1;
}
```

With comparison count (extension)

```
var (idx, comps) = LinearSearch(new List<string> { "Alice", "Bob", "Charlie" }, "Charlie");
Console.WriteLine($"Found at index {idx} after {comps} comparisons.");

(int index, int comparisons) LinearSearch(List<string> lst, string target) {
    for (int i = 0; i < lst.Count; i++) {
        if (lst[i] == target) { return (i, i + 1); }
    }
    return (-1, lst.Count);
}
```

Challenge 46 Binary Search

PROBLEM

Write a function BinarySearch(int[] lst, int target) for a sorted array. Return the index or -1 if not found.

EXAMPLE

```
BinarySearch([1, 3, 5, 7, 9, 11], 7) -> 3
BinarySearch([1, 3, 5, 7, 9, 11], 6) -> -1
```

MARK SCHEME

Correct divide-and-conquer approach. Returns correct index or -1. Requires sorted input.

EXTENSION

Count comparisons. Compare with linear search on the same list.

TEACHER NOTES

> Best done after challenge 45 (linear search) so students can compare the two. Many students find the pointer logic confusing at first - tracing through a small example on paper before coding helps.

SOLUTIONS

Binary search — iterative

```
int[] lst = { 1, 3, 5, 7, 9, 11 };
Console.WriteLine(BinarySearch(lst, 7));   // 3
Console.WriteLine(BinarySearch(lst, 6));   // -1

int BinarySearch(int[] lst, int target) {
    int low  = 0;
    int high = lst.Length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        if (lst[mid] == target) {
            return mid;
        } else if (lst[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}
```

Recursive version

```
int[] lst = { 1, 3, 5, 7, 9, 11 };
Console.WriteLine(BinarySearch(lst, 7));   // 3
Console.WriteLine(BinarySearch(lst, 6));   // -1

int BinarySearch(int[] lst, int target, int low = 0, int high = -1) {
    if (high == -1) { high = lst.Length - 1; }
    if (low > high) { return -1; }
    int mid = (low + high) / 2;
    if (lst[mid] == target) {
        return mid;
    } else if (lst[mid] < target) {
        return BinarySearch(lst, target, mid + 1, high);
    } else {
        return BinarySearch(lst, target, low, mid - 1);
    }
}
```

Challenge 47

Bubble Sort

PROBLEM

Implement bubble sort to sort an array of numbers in ascending order. Do not use built-in Array.Sort() or LINQ's OrderBy().

EXAMPLE

```
Input: [64, 34, 25, 12, 22, 11, 90]
Output: [11, 12, 22, 25, 34, 64, 90]
```

MARK SCHEME

Correct implementation. Sorted in ascending order. Works for any length array.

EXTENSION

Add an optimisation flag to stop early if no swaps occurred in a pass.

TEACHER NOTES

> Students who understand the concept but write incorrect index logic benefit from tracing through one full pass manually before coding.

SOLUTIONS

Bubble sort — ascending

```
int[] data = { 64, 34, 25, 12, 22, 11, 90 };
BubbleSort(data);
Console.WriteLine(string.Join(", ", data));

void BubbleSort(int[] lst) {
    int n = lst.Length;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1 - i; j++) {
            if (lst[j] > lst[j + 1]) {
                (lst[j], lst[j + 1]) = (lst[j + 1], lst[j]);
            }
        }
    }
}
```

Optimised with early-exit flag

```
int[] data = { 64, 34, 25, 12, 22, 11, 90 };
BubbleSortOptimised(data);
Console.WriteLine(string.Join(", ", data));

void BubbleSortOptimised(int[] lst) {
    int n = lst.Length;
    for (int i = 0; i < n - 1; i++) {
        bool swapped = false;
        for (int j = 0; j < n - 1 - i; j++) {
            if (lst[j] > lst[j + 1]) {
                (lst[j], lst[j + 1]) = (lst[j + 1], lst[j]);
                swapped = true;
            }
        }
        if (!swapped) { return; }
    }
}
```

Challenge 48 Insertion Sort

PROBLEM

Implement insertion sort on a list of strings, sorting alphabetically.

EXAMPLE

```
Input: ["banana", "apple", "cherry", "date"]
Output: ["apple", "banana", "cherry", "date"]
```

MARK SCHEME

Correct algorithm. Alphabetical order produced. Works for any length list.

EXTENSION

Sort in descending order by adding a parameter flag.

TEACHER NOTES

> Using strings rather than numbers gives this a distinct flavour from bubble sort. Students discover that C# compares strings lexicographically by default, which is worth discussing.

SOLUTIONS

Insertion sort — alphabetical

```
string[] words = { "banana", "apple", "cherry", "date" };
InsertionSort(words);
Console.WriteLine(string.Join(", ", words));

void InsertionSort(string[] lst) {
    for (int i = 1; i < lst.Length; i++) {
        string key = lst[i];
        int j      = i - 1;
        while (j >= 0 && string.Compare(lst[j], key) > 0) {
            lst[j + 1] = lst[j];
            j--;
        }
        lst[j + 1] = key;
    }
}
```

Insertion sort — numeric (extension)

```
int[] data = { 5, 2, 9, 1, 7, 3 };
InsertionSort(data);
Console.WriteLine(string.Join(", ", data));

void InsertionSort(int[] lst) {
    for (int i = 1; i < lst.Length; i++) {
        int key = lst[i];
        int j   = i - 1;
        while (j >= 0 && lst[j] > key) {
            lst[j + 1] = lst[j];
            j--;
        }
        lst[j + 1] = key;
    }
}
```

Challenge 49

Merge Sort

PROBLEM

Implement merge sort. Demonstrate it on a list of 10 random integers.

EXAMPLE

```
Before: [38, 27, 43, 3, 9, 82, 10, 1, 64, 17]
After: [1, 3, 9, 10, 17, 27, 38, 43, 64, 82]
```

MARK SCHEME

Correct recursive split. Correct merge procedure. Output is sorted.

EXTENSION

Count the number of comparisons made. Compare with bubble sort on the same data.

TEACHER NOTES

> This is the first genuinely recursive task in this tier. Students who have not seen recursion before may need challenge 68 (Tower of Hanoi) introduced first.

SOLUTIONS

Merge sort

```
Random rng = new();
int[] data  = Enumerable.Range(0, 10).Select(_ => rng.Next(1, 101)).ToArray();
Console.WriteLine("Before: " + string.Join(", ", data));
Console.WriteLine("After:  " + string.Join(", ", MergeSort(data)));

int[] MergeSort(int[] lst) {
    if (lst.Length <= 1) { return lst; }
    int   mid   = lst.Length / 2;
    int[] left  = MergeSort(lst[..mid]);
    int[] right = MergeSort(lst[mid..]);
    return Merge(left, right);
}

int[] Merge(int[] left, int[] right) {
    var result = new List<int>();
    int i = 0;
    int j = 0;
    while (i < left.Length && j < right.Length) {
        if (left[i] <= right[j]) {
            result.Add(left[i++]);
        } else {
            result.Add(right[j++]);
        }
    }
    result.AddRange(left[i..]);
    result.AddRange(right[j..]);
    return result.ToArray();
}
```

Challenge 50

Username Generator

PROBLEM

Write a function that generates a username from a first name, last name, and birth year. Format: first letter of first name + full last name + last two digits of year, all lowercase.

EXAMPLE

```
Alice Smith, 2007 -> asmith07
```

MARK SCHEME

Correct format produced. All lowercase. Handles names with mixed capitalisation.

EXTENSION

Check a list of existing usernames and add a number suffix if the generated name is already taken.

TEACHER NOTES

> A nice applied string manipulation task. The extension gives faster students an interesting problem that introduces list membership testing.

SOLUTIONS

Standard approach

```
Console.Write("First name: ");
string first = Console.ReadLine()!;
Console.Write("Last name: ");
string last  = Console.ReadLine()!;
Console.Write("Birth year: ");
int year     = int.Parse(Console.ReadLine()!);
Console.WriteLine(GenerateUsername(first, last, year));

string GenerateUsername(string first, string last, int year) {
    return char.ToLower(first[0]) + last.ToLower() + (year % 100).ToString("D2");
}
```

With duplicate detection (extension)

```
var existing = new List<string> { "asmith07", "asmith071" };
Console.WriteLine(GenerateUsername("Alice", "Smith", 2007, existing));

string GenerateUsername(string first, string last, int year, List<string> existing) {
    string baseUsername = char.ToLower(first[0]) + last.ToLower() + (year % 100).ToString("D2");
    string username = baseUsername;
    int counter     = 1;

    while (existing.Contains(username)) {
        username = baseUsername + counter;
        counter++;
    }

    return username;
}
```

Challenge 51

Roman Numeral Validator

PROBLEM

Write a function that checks whether a string is a valid Roman numeral.

EXAMPLE

```
IsValidRoman("XIV") -> true
IsValidRoman("IIII") -> false
```

MARK SCHEME

Correctly validates valid Roman numerals. Rejects clearly invalid strings.

EXTENSION

Convert valid Roman numerals to their decimal equivalent.

TEACHER NOTES

A challenge that tests careful rule-following more than syntax knowledge. Students benefit from listing the rules explicitly before coding.

SOLUTIONS

Pattern-based validation

```
using System.Text.RegularExpressions;

Console.WriteLine(IsValidRoman("XIV"));    // True
Console.WriteLine(IsValidRoman("IIII"));   // False

bool IsValidRoman(string s) {
    if (string.IsNullOrEmpty(s)) { return false; }
    return Regex.IsMatch(s,
        @"^M{0,3}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$");
}
```

Roman to decimal (extension)

```
Console.WriteLine(RomanToDecimal("XIV"));   // 14
Console.WriteLine(RomanToDecimal("XLII"));  // 42
Console.WriteLine(RomanToDecimal("MCM"));   // 1900

int RomanToDecimal(string s) {
    var values = new Dictionary<char, int> {
        ['I'] = 1, ['V'] = 5, ['X'] = 10,
        ['L'] = 50, ['C'] = 100, ['D'] = 500, ['M'] = 1000
    };
    int result = 0;
    for (int i = 0; i < s.Length; i++) {
        int current = values[s[i]];
        int next    = i + 1 < s.Length ? values[s[i + 1]] : 0;
        if (current < next) {
            result -= current;
        } else {
            result += current;
        }
    }
    return result;
}
```

Challenge 52 Temperature Statistics

PROBLEM

Ask the user to enter daily temperatures for a week (7 values). Display the average, hottest day, coldest day, and number of days above average.

EXAMPLE

```
Temperatures: 15, 18, 22, 20, 17, 14, 19
Average: 17.9 degrees C
Hottest: Day 3 (22 degrees C)
Coldest: Day 6 (14 degrees C)
Days above average: 3
```

MARK SCHEME

Correct average. Correct min/max with day number identified. Correct above-average count.

EXTENSION

Display a simple text-based bar chart of the temperatures.

TEACHER NOTES

> Students often calculate the above-average count before calculating the average, which causes issues. Encourage them to plan the order of their calculations before writing code.

SOLUTIONS

Temperature statistics

```
double[] temps = new double[7];

for (int day = 0; day < 7; day++) {
    Console.Write($"Day {day + 1} temperature: ");
    temps[day] = double.Parse(Console.ReadLine()!);
}

double total   = 0;
int    hotIdx  = 0;
int    coldIdx = 0;

for (int i = 0; i < 7; i++) {
    total += temps[i];
    if (temps[i] > temps[hotIdx])  { hotIdx  = i; }
    if (temps[i] < temps[coldIdx]) { coldIdx = i; }
}

double average = total / 7;
int    above   = 0;
foreach (double t in temps) {
    if (t > average) { above++; }
}

Console.WriteLine($"Average:            {average:F1} degrees C");
Console.WriteLine($"Hottest: Day {hotIdx + 1} ({temps[hotIdx]}C)");
Console.WriteLine($"Coldest: Day {coldIdx + 1} ({temps[coldIdx]}C)");
Console.WriteLine($"Days above average: {above}");
```

Challenge 53 Postcode Validator

PROBLEM

Write a function that checks if a UK postcode is in a valid format (e.g. SW1A 2AA).

EXAMPLE

```
IsValidPostcode("SW1A 2AA") -> true
IsValidPostcode("HELLO") -> false
```

MARK SCHEME

Accepts valid UK postcodes. Rejects invalid formats. Handles uppercase and lowercase input.

EXTENSION

Extract and display the outward code (first part) from a valid postcode.

TEACHER NOTES

A good task for discussing the complexity of real-world data validation. UK postcodes have several valid formats - discuss with students how much complexity to handle and why full validation is hard.

SOLUTIONS

Manual pattern checking

```
Console.WriteLine(IsValidPostcode("SW1A 2AA"));  // True
Console.WriteLine(IsValidPostcode("HELLO"));     // False

bool IsValidPostcode(string pc) {
    pc = pc.Trim().ToUpper();
    string[] parts = pc.Split(' ');
    if (parts.Length != 2) { return false; }
    string inward = parts[1];
    if (inward.Length != 3)        { return false; }
    if (!char.IsDigit(inward[0]))  { return false; }
    if (!char.IsLetter(inward[1])) { return false; }
    if (!char.IsLetter(inward[2])) { return false; }
    return parts[0].Length >= 2;
}
```

Challenge 54

Hangman

PROBLEM

Implement a text-based Hangman game. Choose a random word from a predefined list. Give the player 6 lives.

EXAMPLE

```
Word: _ _ _ _ _
Guess a letter: e
Word: _ e _ _ _
Lives remaining: 6
```

MARK SCHEME

Word displayed as underscores. Letters revealed on correct guess. Lives decremented on wrong guess. Win/lose detected.

EXTENSION

Display an ASCII art gallows that builds progressively as lives are lost.

TEACHER NOTES

> Students enjoy this task and it combines several skills naturally. Common issues include not checking for repeated guesses and not detecting the win condition once the last letter is revealed.

SOLUTIONS

Hangman — 6 lives

```
Random rng      = new();
string[] wordList = { "csharp", "hangman", "keyboard", "monitor", "variable" };
string word     = wordList[rng.Next(wordList.Length)];
var guessed     = new List<char>();
int lives       = 6;
bool won        = false;

while (lives > 0 && !won) {
    string display = string.Join(" ", word.Select(c => guessed.Contains(c) ? c : '_'));
    Console.WriteLine($"Word: {display}  Lives: {lives}");

    if (!display.Contains('_')) {
        Console.WriteLine("You win!");
        won = true;
    } else {
        Console.Write("Guess a letter: ");
        char guess = Console.ReadLine()![0];

        if (guessed.Contains(guess)) {
            Console.WriteLine("Already guessed.");
        } else if (word.Contains(guess)) {
            guessed.Add(guess);
            Console.WriteLine("Correct!");
        } else {
            guessed.Add(guess);
            lives--;
            Console.WriteLine("Wrong!");
        }
    }
}

if (lives == 0) {
    Console.WriteLine($"Game over! The word was: {word}");
}
```

Challenge 55

Contact Book

PROBLEM

Build a contact book using a Dictionary<string, string>. Allow the user to add, search, update, and delete contacts (name and phone number).

EXAMPLE

```
1. Add contact
2. Search
3. Update
4. Delete
5. Quit
```

MARK SCHEME

All four operations work. Search returns result or 'not found'. Handles empty contact book.

EXTENSION

Save and load the contact book from a file.

TEACHER NOTES

> A good consolidation task for dictionaries. The file persistence extension bridges into the Applied tier and works well for more confident students who finish early.

SOLUTIONS

Contact book with Dictionary

```
var contacts = new Dictionary<string, string>();
string choice = "";

while (choice != "5") {
    Console.WriteLine("\n1. Add  2. Search  3. Update  4. Delete  5. Quit");
    Console.Write("Choice: ");
    choice = Console.ReadLine()!;

    if (choice == "1") {
        Console.Write("Name: ");
        string name  = Console.ReadLine()!;
        Console.Write("Phone: ");
        string phone = Console.ReadLine()!;
        contacts[name] = phone;
        Console.WriteLine("Contact added.");
    } else if (choice == "2") {
        Console.Write("Search name: ");
        string name = Console.ReadLine()!;
        if (contacts.TryGetValue(name, out string? p)) {
            Console.WriteLine(p);
        } else {
            Console.WriteLine("Not found.");
        }
    } else if (choice == "3") {
        Console.Write("Name to update: ");
        string name = Console.ReadLine()!;
        if (contacts.ContainsKey(name)) {
            Console.Write("New phone: ");
            contacts[name] = Console.ReadLine()!;
            Console.WriteLine("Updated.");
        } else {
            Console.WriteLine("Not found.");
        }
    } else if (choice == "4") {
        Console.Write("Name to delete: ");
        string name = Console.ReadLine()!;
        if (contacts.Remove(name)) {
            Console.WriteLine("Deleted.");
        } else {
            Console.WriteLine("Not found.");
        }
    }
}
```

Applied

File I/O, error handling, data structures, recursion, and applied algorithms. Upper secondary and A-Level.

Challenge 56 File Word Count

PROBLEM

Write a program that reads a text file and displays the total word count, line count, and the five most common words.

EXAMPLE

```
File: story.txt
Lines: 42
Words: 387
Top 5 words: the(28), a(21), and(19), he(15), was(12)
```

MARK SCHEME

Correct file read with error handling. Correct word and line counts. Top 5 words identified and sorted.

EXTENSION

Ignore common stop words (the, a, is, in, and...) from the top 5.

TEACHER NOTES

> Introduce file I/O carefully before this task. Provide a sample text file for students to test with.

SOLUTIONS

File word count with top 5

```
Console.Write("Enter filename: ");
string filename = Console.ReadLine()!;

try {
    string[] lines = File.ReadAllLines(filename);
    var counts     = new Dictionary<string, int>();
    int wordCount  = 0;

    foreach (string line in lines) {
        foreach (string word in line.ToLower().Split(' ', StringSplitOptions.RemoveEmptyEntries))
{
            string w = word.Trim('.', ',', '!', '?', ';', ':', '"', ''');
            wordCount++;
            counts.TryGetValue(w, out int v);
            counts[w] = v + 1;
        }
    }

    var top5 = counts.OrderByDescending(kv => kv.Value).Take(5);
    Console.WriteLine($"Lines: {lines.Length}");
    Console.WriteLine($"Words: {wordCount}");
    Console.WriteLine("Top 5: " + string.Join(", ", top5.Select(kv => $"{kv.Key}({kv.Value})")));
} catch (FileNotFoundException) {
    Console.WriteLine("File not found.");
}
```

Challenge 57

CSV Reader

PROBLEM

Read a CSV file of student names and marks. Display each student's name and grade. Calculate the class average.

EXAMPLE

```
Alice, 78 -> B
Bob, 91 -> A*
Class average: 84.5
```

MARK SCHEME

Correct file read. Grade assigned from mark. Class average calculated correctly.

EXTENSION

Write results (name, mark, grade) to a new CSV output file.

TEACHER NOTES

> Provide a sample CSV file. Students who encounter encoding issues with special characters should use File.ReadAllLines(path, Encoding.UTF8).

SOLUTIONS

CSV reader with grades

```
string GetGrade(int score) {
    if (score >= 90) { return "A*"; }
    if (score >= 80) { return "A";  }
    if (score >= 70) { return "B";  }
    if (score >= 60) { return "C";  }
    if (score >= 50) { return "D";  }
    return "U";
}

try {
    double total = 0;
    int    count = 0;

    foreach (string line in File.ReadAllLines("students.csv")) {
        string[] parts = line.Split(',');
        string name    = parts[0].Trim();
        int mark       = int.Parse(parts[1].Trim());
        Console.WriteLine($"{name}: {mark} - {GetGrade(mark)}");
        total += mark;
        count++;
    }

    if (count > 0) {
        Console.WriteLine($"Class average: {total / count:F1}");
    }
} catch (FileNotFoundException) {
    Console.WriteLine("students.csv not found.");
}
```

Challenge 58 High Score File

PROBLEM

Write a game score system. Save the player's name and score to a file. Each time the program runs, load existing scores and display the top 3.

EXAMPLE

```
Enter your name: Alice
Enter your score: 2450
Top 3 scores:
1. Bob - 3100
2. Alice - 2450
3. Charlie - 1800
```

MARK SCHEME

Correct file write (append). Correct file read. Top 3 identified and displayed. Handles missing file on first run.

EXTENSION

Store scores as a sorted leaderboard. Keep only the top 10.

TEACHER NOTES

> The missing file on first run is the most commonly missed edge case. Use File.Exists() to check, or catch the exception.

SOLUTIONS

High score file

```
const string SCORE_FILE = "scores.txt";

Console.Write("Enter your name: ");
string name  = Console.ReadLine()!;
Console.Write("Enter your score: ");
int    score = int.Parse(Console.ReadLine()!);

File.AppendAllText(SCORE_FILE, $"{name},{score}\n");

var scores = new List<(string name, int score)>();
if (File.Exists(SCORE_FILE)) {
    foreach (string line in File.ReadAllLines(SCORE_FILE)) {
        var parts = line.Split(',');
        scores.Add((parts[0], int.Parse(parts[1])));
    }
}

var top3 = scores.OrderByDescending(s => s.score).Take(3).ToList();
Console.WriteLine("\nTop 3 Scores:");
for (int i = 0; i < top3.Count; i++) {
    Console.WriteLine($"{i + 1}. {top3[i].name} - {top3[i].score}");
}
```

Challenge 59

Error-Handled Calculator

PROBLEM

Build a calculator that handles division by zero, invalid input, and unknown operations gracefully using try/catch.

EXAMPLE

```
Enter first number: 10
Enter operator (+, -, *, /): /
Enter second number: 0
Error: Cannot divide by zero.
```

MARK SCHEME

All four operations work. Division by zero handled. Non-numeric input handled. Unknown operator handled.

EXTENSION

Add support for ** (power) and % (modulo).

TEACHER NOTES

> Good for introducing exception handling. Students often catch all errors with a bare catch clause - discuss why catching specific exception types is better practice.

SOLUTIONS

Calculator with full error handling

```
bool keepGoing = true;

while (keepGoing) {
    try {
        Console.Write("First number: ");
        double a = double.Parse(Console.ReadLine()!);
        Console.Write("Operator (+, -, *, /): ");
        string op = Console.ReadLine()!;
        Console.Write("Second number: ");
        double b = double.Parse(Console.ReadLine()!);

        if (op == "+") {
            Console.WriteLine($"Result: {a + b}");
        } else if (op == "-") {
            Console.WriteLine($"Result: {a - b}");
        } else if (op == "*") {
            Console.WriteLine($"Result: {a * b}");
        } else if (op == "/") {
            if (b == 0) {
                Console.WriteLine("Error: Cannot divide by zero.");
            } else {
                Console.WriteLine($"Result: {a / b}");
            }
        } else {
            Console.WriteLine("Unknown operator.");
        }
    } catch (FormatException) {
        Console.WriteLine("Invalid input. Please enter numbers only.");
    }

    Console.Write("Calculate again? (y/n): ");
    keepGoing = Console.ReadLine()!.ToLower() == "y";
}
```

Challenge 60

Log File Analyser

PROBLEM

Given a log file where each line starts with [ERROR], [INFO], or [WARNING], count each type and display the three most recent errors.

EXAMPLE

```
INFO: 142
WARNING: 31
ERROR: 8
Most recent errors:
[ERROR] Database connection failed
```

MARK SCHEME

Correct counts for all three types. Last three errors identified. Handles missing file.

EXTENSION

Write a summary report to a new file.

TEACHER NOTES

> Provide a sample log file for students to use. This task links well with real-world discussions about how developers monitor software in production.

SOLUTIONS

Log file analyser

```
var counts = new Dictionary<string, int> {
    ["INFO"] = 0, ["WARNING"] = 0, ["ERROR"] = 0
};
var errors = new List<string>();

try {
    foreach (string line in File.ReadAllLines("server.log")) {
        foreach (string level in counts.Keys) {
            if (line.StartsWith($"[{level}]")) {
                counts[level]++;
                if (level == "ERROR") {
                    errors.Add(line);
                }
            }
        }
    }

    foreach (var kv in counts) {
        Console.WriteLine($"{kv.Key}: {kv.Value}");
    }

    Console.WriteLine("\nMost recent errors:");
    foreach (string e in errors.TakeLast(3)) {
        Console.WriteLine(e);
    }
} catch (FileNotFoundException) {
    Console.WriteLine("Log file not found.");
}
```

Challenge 61

Student Records (OOP)

PROBLEM

Create a Student class with name, age, and a List<int> of grades. Add methods to add a grade, calculate the average, and return the highest grade.

EXAMPLE

```
Alice (17): Grades [78, 85, 92] - Average: 85.0 - Highest: 92
```

MARK SCHEME

Class with constructor. Correct methods. Grades list maintained and updated correctly.

EXTENSION

Create a Classroom class that holds a List<Student>. Add a method to return the top student.

TEACHER NOTES

> Students who have done challenge 44 (BankAccount) will find this more comfortable. Encourage students to think about what data belongs in the class and what belongs outside it.

SOLUTIONS

Student class with OOP

```
var alice = new Student("Alice", 17);
foreach (int g in new[] { 78, 85, 92 }) {
    alice.AddGrade(g);
}
Console.WriteLine(alice);

class Student {
    string    name;
    int       age;
    List<int> grades = new();

    public Student(string name, int age) {
        this.name = name;
        this.age  = age;
    }

    public void AddGrade(int grade) {
        grades.Add(grade);
    }

    public double Average() {
        if (grades.Count == 0) { return 0; }
        double total = 0;
        foreach (int g in grades) { total += g; }
        return total / grades.Count;
    }

    public int Highest() {
        int best = grades[0];
        foreach (int g in grades) {
            if (g > best) { best = g; }
        }
        return best;
    }

    public override string ToString() {
        return $"{name} ({age}): Grades [{string.Join(", ", grades)}] - " +
               $"Average: {Average():F1} - Highest: {Highest()}";
    }
}
```

Challenge 62 Animal Hierarchy (Inheritance)

PROBLEM

Create an Animal base class with a Name field. Create Dog and Cat subclasses that override a virtual Speak() method. Create a GuideDog subclass of Dog that adds a Guide() method.

EXAMPLE

```
var dog = new Dog("Rex");
dog.Speak();  -> Rex says: Woof!
var guide = new GuideDog("Buddy");
guide.Guide();  -> Buddy is guiding their owner.
```

MARK SCHEME

Inheritance implemented correctly. Method overriding demonstrated. GuideDog inherits from Dog.

EXTENSION

Override ToString() in each class. Create a List<Animal> and call Speak() on each (polymorphism).

TEACHER NOTES

> The GuideDog inheriting from Dog (rather than directly from Animal) is the key concept here. Draw the inheritance hierarchy on the board and ask students to identify which methods each class has access to.

SOLUTIONS

Inheritance and polymorphism

```
var animals = new Animal[] {
    new Dog("Rex"),
    new Cat("Whiskers"),
    new GuideDog("Buddy")
};

foreach (var animal in animals) {
    Console.WriteLine(animal.Speak());
}

if (animals[2] is GuideDog gd) {
    Console.WriteLine(gd.Guide());
}

class Animal {
    protected string name;
    public Animal(string name) { this.name = name; }
    public virtual string Speak() { return $"{name} makes a sound."; }
}

class Dog : Animal {
    public Dog(string name) : base(name) {}
    public override string Speak() { return $"{name} says: Woof!"; }
}

class Cat : Animal {
    public Cat(string name) : base(name) {}
    public override string Speak() { return $"{name} says: Meow!"; }
}

class GuideDog : Dog {
    public GuideDog(string name) : base(name) {}
    public string Guide() { return $"{name} is guiding their owner."; }
}
```

Challenge 63

Linked List

PROBLEM

Implement a singly linked list with Node and LinkedList classes. Provide methods to Append, Delete by value, and Display.

EXAMPLE

```
ll.Append(1); ll.Append(2); ll.Append(3);
ll.Display() -> 1 -> 2 -> 3
ll.Delete(2);
ll.Display() -> 1 -> 3
```

MARK SCHEME

Node class with Value and Next pointer. Correct Append and Delete. Delete handles missing value. Display traverses correctly.

EXTENSION

Add a method to reverse the linked list in place.

TEACHER NOTES

> Teach this alongside theory on linked lists. Students benefit from drawing boxes and arrows on paper before coding. The edge case of deleting the head node catches many students out.

SOLUTIONS

Singly linked list

```
var ll = new LinkedList_();
ll.Append(1); ll.Append(2); ll.Append(3);
ll.Display();  // 1 -> 2 -> 3
ll.Delete(2);
ll.Display();  // 1 -> 3

class Node_ {
    public int    Value;
    public Node_? Next;
    public Node_(int value) { Value = value; }
}

class LinkedList_ {
    Node_? head;

    public void Append(int value) {
        var node = new Node_(value);
        if (head == null) {
            head = node;
            return;
        }
        var curr = head;
        while (curr.Next != null) { curr = curr.Next; }
        curr.Next = node;
    }

    public void Delete(int value) {
        if (head == null) { return; }
        if (head.Value == value) { head = head.Next; return; }
        var curr = head;
        while (curr.Next != null) {
            if (curr.Next.Value == value) {
                curr.Next = curr.Next.Next;
                return;
            }
            curr = curr.Next;
        }
    }

    public void Display() {
        var parts = new List<string>();
        var curr  = head;
        while (curr != null) {
            parts.Add(curr.Value.ToString());
            curr = curr.Next;
        }
        Console.WriteLine(string.Join(" -> ", parts));
    }
}
```

Challenge 64

Binary Search Tree

PROBLEM

Implement a binary search tree with Insert, Search, and in-order traversal.

EXAMPLE

```
Insert: 5, 3, 7, 1, 4
In-order: [1, 3, 4, 5, 7]
Search 4: Found
```

MARK SCHEME

Correct insertion maintaining BST property. Search returns true/false. In-order traversal produces sorted output.

EXTENSION

Add a method to find the height of the tree.

TEACHER NOTES

> In-order traversal is recursive and students who have not seen recursion before will struggle. Challenge 49 (merge sort) or challenge 68 (Tower of Hanoi) make useful prerequisites.

SOLUTIONS

Binary search tree

```
var bst = new BST();
foreach (int v in new[] { 5, 3, 7, 1, 4, 6, 8 }) {
    bst.Insert(v);
}
Console.Write("In-order: "); bst.InOrder(); Console.WriteLine();
Console.WriteLine($"Search 4: {bst.Search(4)}");
Console.WriteLine($"Search 9: {bst.Search(9)}");

class BSTNode {
    public int Value; public BSTNode? Left, Right;
    public BSTNode(int v) { Value = v; }
}

class BST {
    BSTNode? root;

    public void Insert(int value) { root = Insert(root, value); }

    BSTNode Insert(BSTNode? node, int value) {
        if (node == null) { return new BSTNode(value); }
        if (value < node.Value) {
            node.Left  = Insert(node.Left, value);
        } else if (value > node.Value) {
            node.Right = Insert(node.Right, value);
        }
        return node;
    }

    public bool Search(int value) { return Search(root, value); }

    bool Search(BSTNode? node, int value) {
        if (node == null) { return false; }
        if (value == node.Value) { return true; }
        if (value < node.Value) { return Search(node.Left, value); }
        return Search(node.Right, value);
    }

    public void InOrder() { InOrder(root); }

    void InOrder(BSTNode? node) {
        if (node == null) { return; }
        InOrder(node.Left);
        Console.Write(node.Value + " ");
        InOrder(node.Right);
    }
}
```

Challenge 65

Hash Table

PROBLEM

Implement a hash table with a simple hash function, Insert, and Lookup. Handle collisions using chaining.

EXAMPLE

```
table.Insert("name", "Alice")
table.Lookup("name") -> "Alice"
table.Lookup("age") -> null
```

MARK SCHEME

Correct hash function. Insert stores key-value pair. Lookup returns correct value. Collision handling implemented.

EXTENSION

Implement a Delete method. Display the table's load factor.

TEACHER NOTES

> Links well with theory on hashing and collision handling. Walk through what happens when two keys hash to the same index before students write the collision handling code.

SOLUTIONS

Recursive flood fill

```
char[,] grid = {
    { '.', '.', '#', '.', '.' },
    { '.', '.', '#', '.', '.' },
    { '.', '.', '.', '.', '.' },
    { '#', '.', '.', '#', '.' },
    { '.', '.', '.', '#', '.' },
};

FloodFill(grid, 0, 0, '*');
PrintGrid(grid);

void FloodFill(char[,] g, int r, int c, char fill) {
    if (r < 0 || r >= g.GetLength(0)) { return; }
    if (c < 0 || c >= g.GetLength(1)) { return; }
    if (g[r, c] != '.') { return; }
    g[r, c] = fill;
    FloodFill(g, r - 1, c, fill);
    FloodFill(g, r + 1, c, fill);
    FloodFill(g, r, c - 1, fill);
    FloodFill(g, r, c + 1, fill);
}

void PrintGrid(char[,] g) {
    for (int r = 0; r < g.GetLength(0); r++) {
        for (int c = 0; c < g.GetLength(1); c++) {
            Console.Write(g[r, c]);
        }
        Console.WriteLine();
    }
}
```

Challenge 66

Graph (Adjacency List)

PROBLEM

Represent a graph of cities and roads using an adjacency list (Dictionary<string, List<string>>). Add methods to add a node, add an edge, and display all connections.

EXAMPLE

```
graph.AddEdge("London", "Brighton")
graph.Display()
London: [Brighton]
Brighton: [London]
```

MARK SCHEME

Correct adjacency list structure. Bidirectional edges added. Display shows all connections clearly.

EXTENSION

Implement breadth-first search to find if two cities are connected.

TEACHER NOTES

> Pairs well with theory on graph data structures. Students often forget to add the reverse edge for an undirected graph.

SOLUTIONS

Caesar cipher brute force

```
Console.Write("Enter ciphertext: ");
string text = Console.ReadLine()!;

for (int shift = 1; shift <= 25; shift++) {
    string result = "";
    foreach (char c in text) {
        if (char.IsLetter(c)) {
            char baseChar = char.IsUpper(c) ? 'A' : 'a';
            result += (char)(((c - baseChar + shift) % 26) + baseChar);
        } else {
            result += c;
        }
    }
    Console.WriteLine($"Shift {shift,2}: {result}");
}
```

Challenge 67

Dijkstra's Shortest Path

PROBLEM

Using a weighted graph as a Dictionary<string, Dictionary<string, int>>, implement Dijkstra's algorithm to find the shortest path between two nodes.

EXAMPLE

```
Shortest path from A to D: A -> B -> D (cost: 7)
```

MARK SCHEME

Correct distances calculated. Shortest path reconstructed and displayed. Handles disconnected graphs.

EXTENSION

Display the full path, not just the total distance.

TEACHER NOTES

> This is a genuinely challenging task best reserved for A-Level students. Work through a small worked example on the board before students attempt to code it.

SOLUTIONS

N-Queens — backtracking

```
int n = 8;
var board = new int[n];
int solutions = 0;
Solve(0);
Console.WriteLine($"Found {solutions} solutions for {n}-Queens.");

void Solve(int row) {
    if (row == n) {
        solutions++;
        if (solutions == 1) {
            for (int r = 0; r < n; r++) {
                for (int c = 0; c < n; c++) {
                    Console.Write(board[r] == c ? "Q " : ". ");
                }
                Console.WriteLine();
            }
        }
        return;
    }
    for (int col = 0; col < n; col++) {
        if (IsSafe(row, col)) {
            board[row] = col;
            Solve(row + 1);
        }
    }
}

bool IsSafe(int row, int col) {
    for (int r = 0; r < row; r++) {
        if (board[r] == col) { return false; }
        if (Math.Abs(board[r] - col) == Math.Abs(r - row)) { return false; }
    }
    return true;
}
```

Challenge 68

Tower of Hanoi

PROBLEM

Write a recursive function to solve the Tower of Hanoi puzzle for n discs. Display each move and return the total count.

EXAMPLE

```
n=3:
Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
...
7 moves total.
```

MARK SCHEME

Correct recursive implementation. All moves displayed correctly. Total moves = 2^n - 1.

EXTENSION

Display the total number of moves alongside the solution.

TEACHER NOTES

> One of the classic recursion tasks. Students find it hard to write but satisfying when it works. Start with n=2 and trace through on the board - the pattern becomes clear and the code almost writes itself.

SOLUTIONS

Knapsack — dynamic programming

```
int[] weights = { 1, 3, 4, 5 };
int[] values  = { 1, 4, 5, 7 };
int capacity  = 7;

Console.WriteLine($"Max value: {Knapsack(weights, values, capacity)}");

int Knapsack(int[] w, int[] v, int cap) {
    int n     = w.Length;
    int[,] dp = new int[n + 1, cap + 1];

    for (int i = 1; i <= n; i++) {
        for (int j = 0; j <= cap; j++) {
            dp[i, j] = dp[i - 1, j];
            if (w[i - 1] <= j) {
                int withItem = dp[i - 1, j - w[i - 1]] + v[i - 1];
                if (withItem > dp[i, j]) {
                    dp[i, j] = withItem;
                }
            }
        }
    }
    return dp[n, cap];
}
```

Challenge 69

Maze Solver

PROBLEM

Given a 2D grid maze (0=open, 1=wall), write a recursive function to find a path from a start position to an end position.

EXAMPLE

```
Maze solved: path found in 12 steps.
```

MARK SCHEME

Correct recursive backtracking. Path found if one exists. Returns no solution when none exists.

EXTENSION

Display the solved path marked on the grid with a special character.

TEACHER NOTES

> Backtracking is a conceptually challenging step from basic recursion. Students need to understand that returning false on a dead end triggers the parent call to try another direction.

SOLUTIONS

Towers of Hanoi

```
Console.Write("Number of discs: ");
int n = int.Parse(Console.ReadLine()!);
int moves = 0;
Hanoi(n, "A", "C", "B");
Console.WriteLine($"Total moves: {moves}");

void Hanoi(int discs, string from, string to, string via) {
    if (discs == 1) {
        Console.WriteLine($"Move disc 1 from {from} to {to}");
        moves++;
        return;
    }
    Hanoi(discs - 1, from, via, to);
    Console.WriteLine($"Move disc {discs} from {from} to {to}");
    moves++;
    Hanoi(discs - 1, via, to, from);
}
```

Challenge 70

Vigenere Cipher

PROBLEM

Implement the Vigenere cipher. Ask for a message and a keyword. Encrypt and display the result.

EXAMPLE

```
Message: hello
Keyword: key
Encrypted: rijvs
```

MARK SCHEME

Correct encryption for all lowercase letters. Keyword repeats correctly. Non-letter characters unchanged.

EXTENSION

Add decryption. Demonstrate how frequency analysis can help crack it.

TEACHER NOTES

> Best done after challenge 29 (Caesar cipher). The key repeating correctly is the hardest part - students often forget to use modulo on the key index.

SOLUTIONS

Maze solver — BFS

```
char[,] maze = {
    { 'S', '.', '#', '.', '.' },
    { '#', '.', '#', '.', '#' },
    { '.', '.', '.', '.', '#' },
    { '.', '#', '#', '.', '.' },
    { '.', '.', '.', '#', 'E' },
};

Solve(maze);

void Solve(char[,] m) {
    int rows = m.GetLength(0);
    int cols = m.GetLength(1);
    (int r, int c) start = (0, 0);
    (int r, int c) end   = (rows - 1, cols - 1);

    var queue   = new Queue<List<(int, int)>>();
    var visited = new HashSet<(int, int)>();

    queue.Enqueue(new List<(int, int)> { start });
    visited.Add(start);

    int[][] dirs = { new[]{-1,0}, new[]{1,0}, new[]{0,-1}, new[]{0,1} };

    while (queue.Count > 0) {
        var path = queue.Dequeue();
        var (r, c) = path[^1];

        if ((r, c) == end) {
            Console.WriteLine($"Path found! Length: {path.Count}");
            return;
        }

        foreach (var dir in dirs) {
            int nr = r + dir[0];
            int nc = c + dir[1];
            if (nr >= 0 && nr < rows && nc >= 0 && nc < cols &&
                m[nr, nc] != '#' && !visited.Contains((nr, nc))) {
                visited.Add((nr, nc));
                var newPath = new List<(int, int)>(path) { (nr, nc) };
                queue.Enqueue(newPath);
            }
        }
    }

    Console.WriteLine("No path found.");
}
```

Challenge 71

API Simulation

PROBLEM

Simulate a simple REST API using a Dictionary<int, Dictionary<string, string>>. Write functions for GET (retrieve by ID), POST (add new record), PUT (update), and DELETE.

EXAMPLE

```
Post({"name": "Alice"})
Get(1) -> {"name": "Alice"}
Put(1, {"name": "Bob"})
Get(1) -> {"name": "Bob"}
```

MARK SCHEME

All four operations work correctly. Correct error handling for missing records.

EXTENSION

Simulate HTTP status codes (200, 201, 404, 400) in return values.

TEACHER NOTES

> Links well with theory on client-server architecture and HTTP methods. Students find it useful to see the parallel between the function names and real HTTP verbs.

SOLUTIONS

Luhn algorithm

```
Console.WriteLine(IsValidCard("4532015112830366"));  // True
Console.WriteLine(IsValidCard("1234567890123456"));  // False

bool IsValidCard(string number) {
    int sum      = 0;
    bool doubleIt = false;

    for (int i = number.Length - 1; i >= 0; i--) {
        int digit = number[i] - '0';
        if (doubleIt) {
            digit *= 2;
            if (digit > 9) { digit -= 9; }
        }
        sum += digit;
        doubleIt = !doubleIt;
    }
    return sum % 10 == 0;
}
```

Challenge 72

Inventory System

PROBLEM

Build an inventory system using a Dictionary. Each item has a name, quantity, and price. Allow the user to add items, update stock, sell items (reducing quantity), and display low-stock alerts (below 5 units).

EXAMPLE

```
Sell("apple", 3) -> Stock: 7
Sell("apple", 9) -> Error: insufficient stock
Low stock alert: banana (2 remaining)
```

MARK SCHEME

All operations work. Selling validates sufficient stock. Low-stock alert identifies all items below threshold.

EXTENSION

Save and load inventory from a CSV file. Display total inventory value.

TEACHER NOTES

> A good task for introducing value tuples or nested dictionaries. Students who have only used flat dictionaries may need a brief worked example.

SOLUTIONS

RLE compress and decompress

```
string original    = "aaabbbccddddee";
string compressed  = Compress(original);
string restored    = Decompress(compressed);

Console.WriteLine($"Original:   {original}");
Console.WriteLine($"Compressed: {compressed}");
Console.WriteLine($"Restored:   {restored}");

string Compress(string s) {
    if (s.Length == 0) { return ""; }
    string result = "";
    int    count  = 1;

    for (int i = 1; i < s.Length; i++) {
        if (s[i] == s[i - 1]) {
            count++;
        } else {
            result += (count == 1 ? "" : count.ToString()) + s[i - 1];
            count = 1;
        }
    }
    result += (count == 1 ? "" : count.ToString()) + s[^1];
    return result;
}

string Decompress(string s) {
    string result = "";
    int i = 0;

    while (i < s.Length) {
        string numStr = "";
        while (i < s.Length && char.IsDigit(s[i])) {
            numStr += s[i++];
        }
        char c      = s[i++];
        int  repeat = numStr.Length > 0 ? int.Parse(numStr) : 1;
        result += new string(c, repeat);
    }
    return result;
}
```

Challenge 73 String Compression

PROBLEM

Implement run-length encoding: compress a string by replacing repeated characters with a count and the character.

EXAMPLE

```
Compress("aaabbbccddddee") -> "3a3b2c4d2e"
Compress("abcd") -> "abcd"
```

MARK SCHEME

Correct compression. Single characters represented without a count. Handles empty string.

EXTENSION

Write a Decompress function. Verify Decompress(Compress(s)) == s.

TEACHER NOTES

> Links well with theory on compression. The edge case of a single-character run (should it print '1a' or just 'a'?) is worth discussing as a design decision before students start.

SOLUTIONS

Full RLE compress / decompress

```
Console.WriteLine(Compress("aaabbbccddddee"));  // 3a3b2c4d2e
Console.WriteLine(Decompress("3a3b2c4d2e"));    // aaabbbccddddee

string Compress(string s) {
    if (s.Length == 0) { return ""; }
    string result = "";
    int    count  = 1;

    for (int i = 1; i < s.Length; i++) {
        if (s[i] == s[i - 1]) {
            count++;
        } else {
            result += (count == 1 ? "" : count.ToString()) + s[i - 1];
            count = 1;
        }
    }
    result += (count == 1 ? "" : count.ToString()) + s[^1];
    return result;
}

string Decompress(string s) {
    string result = "";
    int    i      = 0;

    while (i < s.Length) {
        string numStr = "";
        while (i < s.Length && char.IsDigit(s[i])) {
            numStr += s[i++];
        }
        char c      = s[i++];
        int  repeat = numStr.Length > 0 ? int.Parse(numStr) : 1;
        result += new string(c, repeat);
    }
    return result;
}
```

Challenge 74

Full Number Base Converter

PROBLEM

Build a full converter between binary, decimal, octal, and hexadecimal in both directions, without using C#'s built-in Convert.ToString(n, base).

EXAMPLE

```
Convert 255 (decimal) to:
Binary: 11111111, Octal: 377, Hex: FF
```

MARK SCHEME

All conversion directions correct. Correct for any valid input.

EXTENSION

Add input validation for each base (e.g. binary input only contains 0 and 1).

TEACHER NOTES

> A thorough task that tests understanding of number bases. Best done after challenge 40.

SOLUTIONS

Full base converter

```
Console.Write("Enter a decimal number: ");
int n = int.Parse(Console.ReadLine()!);
Console.WriteLine($"Binary:      {FromDecimal(n, 2)}");
Console.WriteLine($"Octal:       {FromDecimal(n, 8)}");
Console.WriteLine($"Hexadecimal: {FromDecimal(n, 16)}");

Console.Write("Enter a binary number: ");
string b = Console.ReadLine()!;
Console.WriteLine($"Binary {b} = decimal {ToDecimal(b, 2)}");

string FromDecimal(int n, int base_) {
    if (n == 0) { return "0"; }
    const string digits = "0123456789ABCDEF";
    string result = "";
    while (n > 0) {
        result = digits[n % base_] + result;
        n /= base_;
    }
    return result;
}

int ToDecimal(string s, int base_) {
    const string digits = "0123456789ABCDEF";
    int result = 0;
    foreach (char c in s.ToUpper()) {
        result = result * base_ + digits.IndexOf(c);
    }
    return result;
}
```

Challenge 75

FCFS Scheduler

PROBLEM

Given a list of tasks with arrival times and durations, implement a First Come First Served (FCFS) scheduler. Display the processing order and average waiting time.

EXAMPLE

```
Task A (arrives 0, duration 4): starts 0, finishes 4, waits 0
Task B (arrives 1, duration 3): starts 4, finishes 7, waits 3
Average wait: 1.5
```

MARK SCHEME

Tasks processed in arrival order. Correct waiting times. Average calculated and displayed.

EXTENSION

Implement Shortest Job First (SJF) and compare average waiting times with FCFS.

TEACHER NOTES

> Links directly with the operating systems topic in GCSE and A-Level specifications.

SOLUTIONS

FCFS scheduler

```
var tasks = new List<(string name, int arrives, int duration)> {
    ("A", 0, 4), ("B", 1, 3), ("C", 3, 2)
};
tasks.Sort((a, b) => a.arrives.CompareTo(b.arrives));

int currentTime = 0;
int totalWait   = 0;

foreach (var (name, arrives, duration) in tasks) {
    int start  = Math.Max(currentTime, arrives);
    int finish = start + duration;
    int wait   = start - arrives;
    totalWait   += wait;
    currentTime  = finish;
    Console.WriteLine($"Task {name}: starts {start}, finishes {finish}, waits {wait}");
}

Console.WriteLine($"Average wait: {totalWait / (double)tasks.Count:F1}");
```

Challenge 76

Phone Number Formatter

PROBLEM

Write a function that takes a UK phone number string in any format (with or without spaces, hyphens, or +44 prefix) and returns it in standard format: 07XXX XXXXXX.

EXAMPLE

```
+447911123456 -> 07911 123456
07911-123-456 -> 07911 123456
```

MARK SCHEME

Strips non-digit characters. Handles +44 prefix. Validates length. Returns standardised format.

EXTENSION

Validate that the result starts with 07 for mobile numbers.

TEACHER NOTES

> A practical real-world task. The various possible input formats make it a good test of systematic thinking.

SOLUTIONS

UK phone number formatter

```
Console.WriteLine(FormatUKPhone("+447911123456"));  // 07911 123456
Console.WriteLine(FormatUKPhone("07911-123-456"));  // 07911 123456

string FormatUKPhone(string raw) {
    string digits = new string(raw.Where(char.IsDigit).ToArray());
    if (digits.StartsWith("44")) {
        digits = "0" + digits[2..];
    }
    if (digits.Length != 11 || !digits.StartsWith("0")) {
        return "Invalid number";
    }
    return digits[..5] + " " + digits[5..];
}
```

Challenge 77

Text Adventure Engine

PROBLEM

Build a text adventure with at least 5 rooms connected by directions (north/south/east/west). The player can move between rooms and pick up items.

EXAMPLE

```
You are in the entrance hall.
Exits: north, east
> north
You are in the library. You see a key.
> take key
```

MARK SCHEME

Room data stored as dictionary or class. Movement validated. Items tracked. Win condition implemented.

EXTENSION

Save and load game state to a JSON file using System.Text.Json.

TEACHER NOTES

> Students enjoy the creative aspect of designing their own rooms. Set a clear minimum (5 rooms, movement working, at least one item) so they do not spend the entire lesson on story design.

SOLUTIONS

Text adventure engine

```
var rooms = new Dictionary<string, (string desc, Dictionary<string,string> exits, List<string>
items)> {
    ["entrance"] = ("You are in the entrance hall.",
        new() { ["north"] = "library", ["east"] = "kitchen" }, new()),
    ["library"]  = ("You are in the library.",
        new() { ["south"] = "entrance" }, new() { "key" }),
    ["kitchen"]  = ("You are in the kitchen.",
        new() { ["west"] = "entrance", ["north"] = "garden" }, new() { "apple" }),
    ["garden"]   = ("You are in the garden.",
        new() { ["south"] = "kitchen" }, new()),
};

string current   = "entrance";
var inventory    = new List<string>();
bool running     = true;

while (running) {
    var (desc, exits, items) = rooms[current];
    Console.WriteLine($"\n{desc}");
    if (items.Count > 0) { Console.WriteLine($"You see: {string.Join(", ", items)}"); }
    Console.WriteLine($"Exits: {string.Join(", ", exits.Keys)}");
    Console.Write("> ");
    string[] cmd = Console.ReadLine()!.ToLower().Split(' ');

    if (cmd.Length == 0) {
        continue;
    } else if (exits.ContainsKey(cmd[0])) {
        current = exits[cmd[0]];
    } else if (cmd[0] == "take" && cmd.Length > 1 && items.Contains(cmd[1])) {
        inventory.Add(cmd[1]);
        items.Remove(cmd[1]);
        Console.WriteLine($"You pick up the {cmd[1]}.");
    } else if (cmd[0] == "inventory") {
        string carrying = inventory.Count > 0 ? string.Join(", ", inventory) : "nothing";
        Console.WriteLine($"Carrying: {carrying}");
    } else if (cmd[0] == "quit") {
        Console.WriteLine("Goodbye!");
        running = false;
    } else {
        Console.WriteLine("Unknown command.");
    }
}
```

Challenge 78 Sentiment Analyser

PROBLEM

Given lists of positive and negative words, analyse a user-entered sentence and classify it as positive, negative, or neutral based on word counts.

EXAMPLE

```
Sentence: this is a great and wonderful day
Positive words: 2, Negative words: 0
Sentiment: Positive
```

MARK SCHEME

Correct word matching (case-insensitive). Correct classification. Score displayed alongside result.

EXTENSION

Load word lists from a file. Allow the user to add new words.

TEACHER NOTES

> Accessible and engaging, especially linked to discussions about AI and natural language processing.

SOLUTIONS

Dictionary-based sentiment analysis

```
string[] positiveWords = { "great","wonderful","excellent","happy","love",
                           "fantastic","amazing","good","brilliant","perfect" };
string[] negativeWords = { "bad","terrible","awful","hate","horrible",
                           "dreadful","poor","sad","disappointing","useless" };

Console.Write("Enter a sentence: ");
string sentence = Console.ReadLine()!.ToLower();
string[] words  = sentence.Split(' ');

int posCount = words.Count(w => positiveWords.Contains(w));
int negCount = words.Count(w => negativeWords.Contains(w));

Console.WriteLine($"Positive words: {posCount}, Negative words: {negCount}");

if (posCount > negCount) {
    Console.WriteLine("Sentiment: Positive");
} else if (negCount > posCount) {
    Console.WriteLine("Sentiment: Negative");
} else {
    Console.WriteLine("Sentiment: Neutral");
}
```

Challenge 79

Timetable Clash Detector

PROBLEM

Given a list of lessons (room, day, period), write a function that detects any room that is double-booked (same room, day, and period).

EXAMPLE

```
CLASH: Room 101, Monday, Period 3 - booked by Maths and Science
```

MARK SCHEME

All clashes correctly detected. Each clash reported with details. No false positives.

EXTENSION

Also detect teacher clashes (same teacher, same day, same period).

TEACHER NOTES

> Introduces value tuples as dictionary keys, which is a useful C# technique. Students who use three separate nested dictionaries can be guided toward the tuple key approach as a cleaner solution.

SOLUTIONS

Timetable clash detector

```
var lessons = new List<(string room, string day, int period, string subject)> {
    ("101", "Monday",  3, "Maths"),
    ("101", "Monday",  3, "Science"),
    ("102", "Tuesday", 1, "English"),
    ("101", "Monday",  5, "PE"),
};

var booked = new Dictionary<(string, string, int), string>();
foreach (var (room, day, period, subject) in lessons) {
    var key = (room, day, period);
    if (booked.ContainsKey(key)) {
        Console.WriteLine($"CLASH: Room {room}, {day}, Period {period} - " +
                          $"{booked[key]} and {subject}");
    } else {
        booked[key] = subject;
    }
}
```

Challenge 80

Pattern Matcher

PROBLEM

Write a simple recursive pattern matching function that supports * (matches any sequence of characters) and ? (matches any single character).

EXAMPLE

```
Match("h?llo", "hello") -> true
Match("h*", "hello world") -> true
Match("h?llo", "hllo") -> false
```

MARK SCHEME

Correct matching for ? (single character). Correct matching for * (any sequence). Handles edge cases.

EXTENSION

Add + to match one or more characters.

TEACHER NOTES

> A challenging task that requires careful recursive thinking. Walk through the base cases on the board before students attempt it.

SOLUTIONS

Pattern matcher with * and ?

```
Console.WriteLine(Match("h?llo", "hello"));        // True
Console.WriteLine(Match("h*",    "hello world"));  // True
Console.WriteLine(Match("h?llo", "hllo"));         // False

bool Match(string pattern, string text) {
    if (pattern.Length == 0) { return text.Length == 0; }
    if (pattern[0] == '?') {
        return text.Length > 0 && Match(pattern[1..], text[1..]);
    }
    if (pattern[0] == '*') {
        return Match(pattern[1..], text) ||
               (text.Length > 0 && Match(pattern, text[1..]));
    }
    return text.Length > 0 && pattern[0] == text[0] && Match(pattern[1..], text[1..]);
}
```

Stretch Challenges

Multi-part problems combining multiple skills. A-Level standard. Maps to NEA complexity requirements.

Challenge 81

Library Management System

PROBLEM

A library stores books (title, author, ISBN, available: true/false). Write a program that: adds new books, searches by author or title, borrows a book (marks unavailable), returns a book, and lists all available books.

EXAMPLE

```
Borrow("978-0-06-112008-4")
'To Kill a Mockingbird' is now on loan.
SearchAuthor("Harper Lee")
To Kill a Mockingbird - On loan
```

MARK SCHEME

All five functions implemented. Borrow validates availability. Search returns all matches including unavailable books.

EXTENSION

Add a borrower name to borrowed books. Show who has each book currently.

TEACHER NOTES

> A good first multi-function project. Encourage students to run a full sequence of operations (add, borrow, search, return) before submitting.

SOLUTIONS

Library management system

```
var books = new List<Book> {
    new("978-0-06-112008-4", "To Kill a Mockingbird", "Harper Lee",    true),
    new("978-0-7432-7356-5", "1984",                  "George Orwell", true),
};

Borrow("978-0-06-112008-4");
foreach (var b in SearchAuthor("Harper Lee")) {
    string status = b.Available ? "Available" : "On loan";
    Console.WriteLine($"{b.Title} - {status}");
}

void Borrow(string isbn) {
    var b = books.Find(b => b.Isbn == isbn);
    if (b == null) {
        Console.WriteLine("ISBN not found.");
        return;
    }
    if (b.Available) {
        b.Available = false;
        Console.WriteLine($"'{b.Title}' is now on loan.");
    } else {
        Console.WriteLine("Book is already on loan.");
    }
}

List<Book> SearchAuthor(string name) {
    return books.FindAll(b => b.Author.ToLower().Contains(name.ToLower()));
}

class Book {
    public string Isbn, Title, Author;
    public bool   Available;
    public Book(string isbn, string title, string author, bool available) {
        Isbn = isbn; Title = title; Author = author; Available = available;
    }
}
```

Challenge 82

Gradebook with File Persistence

PROBLEM

Build a gradebook that stores students and marks for up to 5 subjects, calculates each student's average, saves to CSV on exit, loads from CSV on startup, and reports the top student per subject.

EXAMPLE

```
Loaded 12 students from gradebook.csv
Top in Maths: Alice (97)
Class average: 74.2
```

MARK SCHEME

File I/O with correct CSV format. Correct averages. Top per subject identified. Handles missing file on first run.

EXTENSION

Generate a summary report showing class average per subject.

TEACHER NOTES

> This task requires students to plan a data structure before coding. Ask them to sketch the CSV layout on paper first.

SOLUTIONS

Gradebook with CSV persistence

```
const string FILENAME = "gradebook.csv";
string[] SUBJECTS     = { "Maths", "English", "Science", "CS", "History" };

var students = Load();
students.Add(new Student("Alice", new[] { 90, 85, 88, 95, 78 }));
Save(students);

if (students.Count > 0) {
    double total = 0;
    foreach (var s in students) {
        double avg = 0;
        foreach (int m in s.Marks) { avg += m; }
        total += avg / s.Marks.Length;
    }
    Console.WriteLine($"Class average: {total / students.Count:F1}");
}

List<Student> Load() {
    var list = new List<Student>();
    if (!File.Exists(FILENAME)) { return list; }
    foreach (string line in File.ReadAllLines(FILENAME).Skip(1)) {
        var parts = line.Split(',');
        list.Add(new Student(parts[0], parts[1..].Select(int.Parse).ToArray()));
    }
    Console.WriteLine($"Loaded {list.Count} students.");
    return list;
}

void Save(List<Student> students) {
    using var w = new StreamWriter(FILENAME);
    w.WriteLine("name," + string.Join(",", SUBJECTS));
    foreach (var s in students) {
        w.WriteLine(s.Name + "," + string.Join(",", s.Marks));
    }
}

class Student {
    public string Name;
    public int[]  Marks;
    public Student(string n, int[] m) { Name = n; Marks = m; }
}
```

Challenge 83 Train Timetable

PROBLEM

Store a train timetable (departure station, arrival station, departure time, arrival time, price) in a list. Write functions to: search by departure and arrival station, find the cheapest route, find the fastest route, and display all trains after a given time.

EXAMPLE

```
Search("London", "Brighton")
Cheapest: 08:15 - £11.50
Fastest: 10:30 - 55 mins
```

MARK SCHEME

All four queries work correctly. Handles no results. Data stored as appropriate structure.

EXTENSION

Allow multi-leg journeys with one change.

TEACHER NOTES

> The time comparison is the trickiest part. Discuss with students whether to store times as strings or integers before they start.

SOLUTIONS

Train timetable search

```
var timetable = new List<(string from, string to, string depart, string arrive, double price)> {
    ("London","Brighton","08:15","09:10",11.50),
    ("London","Brighton","10:30","11:25",18.00),
    ("London","Brighton","12:00","12:55",14.00),
};

Search("London", "Brighton");

int ToMins(string t) {
    var p = t.Split(':');
    return int.Parse(p[0]) * 60 + int.Parse(p[1]);
}

void Search(string origin, string dest) {
    var results = timetable.Where(t => t.from == origin && t.to == dest).ToList();
    if (results.Count == 0) {
        Console.WriteLine("No trains found.");
        return;
    }
    var cheapest = results.MinBy(t => t.price)!;
    var fastest  = results.MinBy(t => ToMins(t.arrive) - ToMins(t.depart))!;
    Console.WriteLine($"Cheapest: {cheapest.depart} - £{cheapest.price:F2}");
    Console.WriteLine($"Fastest:  {fastest.depart} - {ToMins(fastest.arrive) -
ToMins(fastest.depart)} mins");
}
```

Challenge 84

Password Manager

PROBLEM

Build a password manager that stores website, username, and password entries, encrypts passwords using the Caesar cipher before storing, decrypts on retrieval, saves to a file, and searches by website.

EXAMPLE

```
Save("gmail.com", "alice@gmail.com", "mypassword")
Retrieve("gmail.com") -> alice@gmail.com / mypassword
```

MARK SCHEME

All operations work. Encryption/decryption correct. File I/O persists data between runs.

EXTENSION

Require a master password to access the manager. Store it as a hash using SHA256.

TEACHER NOTES

A good motivating task for discussing encryption in a real context. Acknowledge with students that Caesar cipher is not real security - the task is about building the system structure.

SOLUTIONS

Password manager with Caesar encryption

```
const int SHIFT   = 7;
const string FILE = "passwords.txt";

SaveEntry("gmail.com", "alice@gmail.com", "mypassword");
var result = GetEntry("gmail.com");
if (result.HasValue) {
    Console.WriteLine($"Username: {result.Value.user}  Password: {result.Value.pass}");
}

string Encrypt(string text) { return Transform(text, SHIFT);  }
string Decrypt(string text) { return Transform(text, -SHIFT); }

string Transform(string text, int shift) {
    string result = "";
    foreach (char c in text) {
        if (char.IsLetter(c)) {
            char base_ = char.IsUpper(c) ? 'A' : 'a';
            result += (char)(((c - base_ + shift % 26 + 26) % 26) + base_);
        } else {
            result += c;
        }
    }
    return result;
}

void SaveEntry(string site, string user, string pass) {
    File.AppendAllText(FILE, $"{site},{user},{Encrypt(pass)}\n");
}

(string user, string pass)? GetEntry(string site) {
    if (!File.Exists(FILE)) { return null; }
    foreach (string line in File.ReadAllLines(FILE)) {
        var parts = line.Split(',');
        if (parts[0] == site) { return (parts[1], Decrypt(parts[2])); }
    }
    return null;
}
```

Challenge 85

Noughts and Crosses

PROBLEM

Implement a two-player Noughts and Crosses game. Display the board after each move. Detect wins, draws, and invalid moves.

EXAMPLE

```
 X | O | X
---|---|---
 O | X |
---|---|---
   |   | O
X wins!
```

MARK SCHEME

Board displayed correctly after each move. All wins detected. Draw detected. Invalid moves rejected.

EXTENSION

Add a computer player that makes random moves. Then try to make it play optimally.

TEACHER NOTES

> Students are often familiar with the game, which helps. The win detection logic requires checking 8 combinations - encourage students to write this as a separate function.

SOLUTIONS

Noughts and Crosses

```
string[] board   = Enumerable.Range(1, 9).Select(i => i.ToString()).ToArray();
string   current = "X";
int      turn    = 0;
bool     playing = true;

while (turn < 9 && playing) {
    Display(board);
    Console.Write($"{current}'s turn (1-9): ");
    string input = Console.ReadLine()!;

    if (!int.TryParse(input, out int move) || move < 1 || move > 9) {
        Console.WriteLine("Invalid.");
        continue;
    }
    int idx = move - 1;
    if (board[idx] == "X" || board[idx] == "O") {
        Console.WriteLine("Taken.");
        continue;
    }
    board[idx] = current;
    turn++;

    if (CheckWinner(board, current)) {
        Display(board);
        Console.WriteLine($"{current} wins!");
        playing = false;
    } else {
        current = current == "X" ? "O" : "X";
    }
}

if (playing) {
    Display(board);
    Console.WriteLine("It's a draw!");
}

void Display(string[] b) {
    for (int r = 0; r < 3; r++) {
        Console.WriteLine($" {b[r*3]} | {b[r*3+1]} | {b[r*3+2]} ");
        if (r < 2) { Console.WriteLine("-----------"); }
    }
}

bool CheckWinner(string[] b, string p) {
    int[][] wins = {
        new[]{0,1,2}, new[]{3,4,5}, new[]{6,7,8},
        new[]{0,3,6}, new[]{1,4,7}, new[]{2,5,8},
        new[]{0,4,8}, new[]{2,4,6}
    };
    foreach (var w in wins) {
        if (b[w[0]] == p && b[w[1]] == p && b[w[2]] == p) { return true; }
    }
    return false;
}
```

Challenge 86

Hospital Priority Queue

PROBLEM

Patients have a name and priority level (1=urgent, 2=standard, 3=routine). Implement a priority queue to process them in priority order, with equal-priority patients processed in arrival order.

EXAMPLE

```
Add("Alice", 2), Add("Bob", 1), Add("Charlie", 2)
Process() -> Bob (priority 1)
Process() -> Alice (priority 2, first to arrive)
```

MARK SCHEME

Correct priority ordering. FIFO within same priority. Dequeue processes in correct order.

EXTENSION

Add appointment times. Detect and handle time clashes.

TEACHER NOTES

> Links well with the queues theory topic. The stable sort behaviour (preserving arrival order within a priority level) is the key concept.

SOLUTIONS

Priority queue — list based

```
var patients = new List<(string name, int priority, int arrival)>();
int arrival  = 0;

Add("Alice",   2);
Add("Bob",     1);
Add("Charlie", 2);
Process();
Process();
Process();

void Add(string name, int priority) {
    patients.Add((name, priority, arrival++));
}

void Process() {
    if (patients.Count == 0) {
        Console.WriteLine("No patients.");
        return;
    }
    patients.Sort((a, b) => a.priority != b.priority
        ? a.priority.CompareTo(b.priority)
        : a.arrival.CompareTo(b.arrival));
    var p = patients[0];
    patients.RemoveAt(0);
    Console.WriteLine($"Processing: {p.name} (priority {p.priority})");
}
```

Challenge 87 Spell Checker

PROBLEM

Load a dictionary of correctly spelled words. For each word in a user-entered sentence, flag any word not in the dictionary and suggest the three closest matches.

EXAMPLE

```
Input: 'the quikc brwon fox'
Unknown: quikc -> suggestions: quick, thick, quack
Unknown: brwon -> suggestions: brown, brow, crown
```

MARK SCHEME

Correctly flags unknown words. Three suggestions produced for each. Case-insensitive matching.

EXTENSION

Implement the full Levenshtein distance algorithm for more accurate suggestions.

TEACHER NOTES

> Provide a word list. The edit distance algorithm is the intellectual core of this task - give stronger students time to discover it and weaker students a worked example to implement.

SOLUTIONS

Spell checker with edit distance

```
string[] DICTIONARY = { "quick","brown","fox","jumps","over","the","lazy","dog",
                        "thick","quack","brick","crown","trick","frown" };

Console.Write("Enter sentence: ");
string sentence = Console.ReadLine()!.ToLower();

foreach (string word in sentence.Split(' ')) {
    if (!DICTIONARY.Contains(word)) {
        var scored = DICTIONARY.Select(d => (EditDistance(word, d), d)).OrderBy(x => x.Item1);
        string suggestions = string.Join(", ", scored.Take(3).Select(x => x.d));
        Console.WriteLine($"Unknown: {word} - suggestions: {suggestions}");
    }
}

int EditDistance(string a, string b) {
    int m     = a.Length;
    int n     = b.Length;
    int[,] dp = new int[m + 1, n + 1];

    for (int i = 0; i <= m; i++) { dp[i, 0] = i; }
    for (int j = 0; j <= n; j++) { dp[0, j] = j; }

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (a[i-1] == b[j-1]) {
                dp[i, j] = dp[i-1, j-1];
            } else {
                dp[i, j] = 1 + Math.Min(dp[i-1, j], Math.Min(dp[i, j-1], dp[i-1, j-1]));
            }
        }
    }
    return dp[m, n];
}
```

Challenge 88

Bank Transaction Processor

PROBLEM

Read a CSV file of bank transactions (date, description, amount). Categorise each transaction based on keywords in the description. Display total spending per category and flag transactions above 100.

EXAMPLE

```
Food: £234.50
Entertainment: £45.00
Large transactions flagged:
12/03 Netflix Premium £149.99
```

MARK SCHEME

Correct file read. Correct categorisation. Totals correct. Transactions above 100 flagged.

EXTENSION

Generate a monthly summary if the transactions span multiple months.

TEACHER NOTES

> Provide a sample transactions CSV file. Encourage students to define keyword lists as a data structure rather than hardcoding comparisons.

SOLUTIONS

Bank transaction categoriser

```
var CATEGORIES = new Dictionary<string, string[]> {
    ["Food"]          = new[] { "tesco","sainsbury","mcdonalds","greggs","costa" },
    ["Transport"]     = new[] { "trainline","uber","tfl","petrol","bp" },
    ["Entertainment"] = new[] { "netflix","spotify","cinema","amazon","steam" },
    ["Utilities"]     = new[] { "bt","virgin","octopus","bulb","thames" },
};

var transactions = new List<(string date, string desc, double amount)> {
    ("12/03", "Tesco Metro",     45.20),
    ("13/03", "Netflix Premium", 14.99),
    ("14/03", "TfL Oyster",      32.00),
    ("15/03", "Greggs",           3.50),
};

var totals = CATEGORIES.Keys.ToDictionary(k => k, _ => 0.0);
totals["Other"] = 0;

foreach (var (date, desc, amount) in transactions) {
    bool categorised = false;
    foreach (var (cat, keywords) in CATEGORIES) {
        if (keywords.Any(kw => desc.ToLower().Contains(kw))) {
            totals[cat] += amount;
            categorised  = true;
        }
    }
    if (!categorised) { totals["Other"] += amount; }
    if (amount > 100)  { Console.WriteLine($"Large transaction flagged: {date} {desc} £
{amount:F2}"); }
}

foreach (var (cat, total) in totals.Where(kv => kv.Value > 0)) {
    Console.WriteLine($"{cat}: £{total:F2}");
}
```

Challenge 89

Cryptarithmetic Solver

PROBLEM

Solve the puzzle SEND + MORE = MONEY by assigning a unique digit (0-9) to each letter.

EXAMPLE

```
S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
9567 + 1085 = 10652
```

MARK SCHEME

Correct solution found. Leading digit constraint applied. Solution displayed clearly.

EXTENSION

Generalise to solve any cryptarithmetic puzzle entered by the user.

TEACHER NOTES

> A good introduction to brute-force search. The program may take a few seconds to run - a natural lead-in to discussing why smarter constraint-based search is used in practice.

SOLUTIONS

Cryptarithmetic — SEND + MORE = MONEY

```
bool found = false;
Solve(new List<int>(), new bool[10]);

void Solve(List<int> assigned, bool[] used) {
    if (found) { return; }
    if (assigned.Count == 8) {
        int s = assigned[0], e = assigned[1], n = assigned[2], d = assigned[3];
        int m = assigned[4], o = assigned[5], r = assigned[6], y = assigned[7];
        if (s == 0 || m == 0) { return; }
        int send  = s*1000 + e*100 + n*10 + d;
        int more  = m*1000 + o*100 + r*10 + e;
        int money = m*10000 + o*1000 + n*100 + e*10 + y;
        if (send + more == money) {
            Console.WriteLine($"S={s} E={e} N={n} D={d} M={m} O={o} R={r} Y={y}");
            Console.WriteLine($"{send} + {more} = {money}");
            found = true;
        }
        return;
    }
    for (int digit = 0; digit <= 9 && !found; digit++) {
        if (!used[digit]) {
            used[digit] = true;
            assigned.Add(digit);
            Solve(assigned, used);
            assigned.RemoveAt(assigned.Count - 1);
            used[digit] = false;
        }
    }
}
```

Challenge 90 Recursive Directory Tree

PROBLEM

Simulate a file system as nested Dictionary<string, object?> (null = file, dict = folder). Write a recursive function to display the full directory tree with indentation showing depth.

EXAMPLE

```
root/
  documents/
    report.txt
    notes.txt
  images/
    photo.jpg
```

MARK SCHEME

Correct recursive traversal. Indentation reflects depth accurately. Files and folders distinguished.

EXTENSION

Add functions to add files, add folders, and delete entries.

TEACHER NOTES

> A satisfying recursion task with a very visual output. The key insight is distinguishing between dictionary values (folders/null for files) - walk through this distinction before students start.

SOLUTIONS

Recursive directory tree display

```
var filesystem = new Dictionary<string, object?> {
    ["documents"] = new Dictionary<string, object?> {
        ["report.txt"] = null,
        ["notes.txt"]  = null,
    },
    ["images"] = new Dictionary<string, object?> {
        ["photo.jpg"] = null,
    },
    ["readme.txt"] = null,
};

DisplayTree(filesystem, "root");

void DisplayTree(object? node, string name, int indent = 0) {
    string prefix = new string(' ', indent * 2);
    if (node is Dictionary<string, object?> dir) {
        Console.WriteLine($"{prefix}{name}/");
        foreach (var (childName, child) in dir) {
            DisplayTree(child, childName, indent + 1);
        }
    } else {
        Console.WriteLine($"{prefix}{name}");
    }
}
```

Challenge 91

Compression Analysis

PROBLEM

Implement Run-Length Encoding and a simple dictionary-based compression on several test strings. Compare compression ratios and display results as a text-based bar chart.

EXAMPLE

```
RLE  |########## | 71%
Dict |############ | 85%
```

MARK SCHEME

Both algorithms correctly implemented. Compression ratio calculated accurately. Bar chart proportional and labelled.

EXTENSION

Test on a paragraph of real text from a file. Which algorithm performs better on natural language?

TEACHER NOTES

> Links directly with theory on compression. Students are often surprised that RLE performs poorly on natural language.

SOLUTIONS

Compression comparison

```
string[] tests = { "aaabbbccddddee", "abcabcabc", "hello world" };

foreach (string t in tests) {
    string rle = RleCompress(t);
    int r1     = (int)Math.Round((double)rle.Length / t.Length * 100);
    Console.WriteLine($"'{t}'");
    Console.WriteLine($"  RLE:  {rle}  ({r1}% of original)");
}

string RleCompress(string s) {
    if (s.Length == 0) { return ""; }
    string result = "";
    int    count  = 1;
    for (int i = 1; i < s.Length; i++) {
        if (s[i] == s[i-1]) {
            count++;
        } else {
            result += (count == 1 ? "" : count.ToString()) + s[i-1];
            count = 1;
        }
    }
    return result + (count == 1 ? "" : count.ToString()) + s[^1];
}
```

Challenge 92

Supermarket Queue Simulation

PROBLEM

Simulate a supermarket checkout queue. Customers arrive at random intervals. Each takes a random service time (1-5 mins). Run for 60 minutes. Display average waiting time and peak queue length.

EXAMPLE

```
Simulation complete.
Customers served: 23
Average wait: 2.4 minutes
Peak queue length: 7
```

MARK SCHEME

Simulation runs for exactly 60 minutes. Randomised arrivals and service. Both statistics calculated correctly.

EXTENSION

Add a second checkout that opens when the queue exceeds 5 people. Compare wait times.

TEACHER NOTES

> An accessible introduction to simulation as a problem-solving technique. The extension directly demonstrates why supermarkets open extra checkouts at busy times.

SOLUTIONS

Supermarket queue simulation

```
Random rng       = new();
var    queue     = new Queue<int>();
int    serverFreeAt = 0;
int    totalWait    = 0;
int    served       = 0;
int    peakLength   = 0;

for (int time = 0; time < 60; time++) {
    if (rng.NextDouble() < 0.5) {
        queue.Enqueue(time);
    }
    if (serverFreeAt <= time && queue.Count > 0) {
        int arrival  = queue.Dequeue();
        totalWait   += time - arrival;
        served++;
        serverFreeAt = time + rng.Next(1, 6);
    }
    if (queue.Count > peakLength) {
        peakLength = queue.Count;
    }
}

Console.WriteLine($"Customers served:  {served}");
if (served > 0) {
    Console.WriteLine($"Average wait:      {totalWait / (double)served:F1} mins");
}
Console.WriteLine($"Peak queue length: {peakLength}");
```

Challenge 93

Polynomial Calculator

PROBLEM

Represent a polynomial as a double[] (index = power). Write functions to evaluate at x, add two polynomials, multiply two polynomials, and calculate the derivative.

EXAMPLE

```
[1, 2, 3] represents 3x^2 + 2x + 1
Evaluate at x=2: 17
Derivative: [2, 6] -> 2 + 6x
```

MARK SCHEME

All four operations produce correct results. Derivative follows the power rule. Edge cases handled.

EXTENSION

Implement Newton's method to find an approximate root numerically.

TEACHER NOTES

> Most accessible to students with some Maths A-Level background. The array representation of polynomials is clever and worth spending time on before students attempt the operations.

SOLUTIONS

Polynomial operations

```
double[] p = { 1, 2, 3 };  // 3x^2 + 2x + 1
Console.WriteLine($"p(2)       = {Evaluate(p, 2)}");
Console.WriteLine($"p'(x)      = [{string.Join(", ", Derivative(p))}]");
Console.WriteLine($"p + p      = [{string.Join(", ", Add(p, p))}]");
Console.WriteLine($"p * [1,1]  = [{string.Join(", ", Multiply(p, new double[] { 1, 1 }))}]");

double Evaluate(double[] poly, double x) {
    double result = 0;
    for (int i = 0; i < poly.Length; i++) {
        result += poly[i] * Math.Pow(x, i);
    }
    return result;
}

double[] Derivative(double[] poly) {
    if (poly.Length <= 1) { return new double[] { 0 }; }
    double[] d = new double[poly.Length - 1];
    for (int i = 1; i < poly.Length; i++) {
        d[i - 1] = poly[i] * i;
    }
    return d;
}

double[] Add(double[] a, double[] b) {
    int      len = Math.Max(a.Length, b.Length);
    double[] r   = new double[len];
    for (int i = 0; i < len; i++) {
        r[i] = (i < a.Length ? a[i] : 0) + (i < b.Length ? b[i] : 0);
    }
    return r;
}

double[] Multiply(double[] a, double[] b) {
    double[] r = new double[a.Length + b.Length - 1];
    for (int i = 0; i < a.Length; i++) {
        for (int j = 0; j < b.Length; j++) {
            r[i + j] += a[i] * b[j];
        }
    }
    return r;
}
```

Challenge 94

Social Network

PROBLEM

Build a social network as an undirected graph using a Dictionary<string, HashSet<string>>. Implement: add user, add friendship, suggest friends (friends-of-friends not already connected), find shortest connection path, and identify the most connected user.

EXAMPLE

```
SuggestFriends("Alice") -> ["Charlie", "Dave"]
ConnectionPath("Alice", "Eve") -> Alice -> Bob -> Eve
```

MARK SCHEME

Graph correctly maintained as bidirectional. Friend suggestions correct. BFS for shortest path. Most connected user identified.

EXTENSION

Detect cliques - groups of three or more users who are all friends with each other.

TEACHER NOTES

> One of the more open-ended tasks. Students who implement BFS for the shortest path are doing A-Level standard work.

SOLUTIONS

Social network graph

```
var network = new Dictionary<string, HashSet<string>>();

AddFriendship("Alice", "Bob");
AddFriendship("Bob",   "Charlie");
AddFriendship("Alice", "Dave");

Console.WriteLine("Suggestions for Alice: " + string.Join(", ", SuggestFriends("Alice")));
Console.WriteLine("Path: " + string.Join(" -> ", ConnectionPath("Alice", "Charlie") ?? new()));

void AddUser(string name) {
    if (!network.ContainsKey(name)) { network[name] = new(); }
}

void AddFriendship(string a, string b) {
    AddUser(a); AddUser(b);
    network[a].Add(b);
    network[b].Add(a);
}

List<string> SuggestFriends(string name) {
    var friends     = network.GetValueOrDefault(name, new());
    var suggestions = new HashSet<string>();
    foreach (string friend in friends) {
        foreach (string fof in network[friend]) {
            if (fof != name && !friends.Contains(fof)) {
                suggestions.Add(fof);
            }
        }
    }
    return suggestions.ToList();
}

List<string>? ConnectionPath(string start, string end) {
    var visited = new HashSet<string>();
    var queue   = new Queue<List<string>>();
    queue.Enqueue(new() { start });
    visited.Add(start);

    while (queue.Count > 0) {
        var path = queue.Dequeue();
        string current = path[^1];
        if (current == end) { return path; }
        foreach (string neighbour in network.GetValueOrDefault(current, new())) {
            if (!visited.Contains(neighbour)) {
                visited.Add(neighbour);
                var newPath = new List<string>(path) { neighbour };
                queue.Enqueue(newPath);
            }
        }
    }
    return null;
}
```

Challenge 95

Text RPG with Save/Load

PROBLEM

Build a role-playing game using classes. Player has health, attack, defence, and level. Include at least three enemies. Implement turn-based combat, experience points, level-up logic, and save/load game state to a JSON file.

EXAMPLE

```
You encounter a Goblin (HP: 20, ATK: 5)
You attack for 8 damage. Goblin HP: 12
Goblin attacks for 3 damage. Your HP: 47
Goblin defeated! +50 XP
```

MARK SCHEME

Classes defined correctly. Combat system works. Level-up triggers at correct XP threshold. Save/load preserves all player attributes.

EXTENSION

Add an inventory system. Add multiple rooms connected by directions.

TEACHER NOTES

> Students enjoy this task but often spend too long on content rather than structure. Set clear time checkpoints: class design first, then combat, then persistence.

SOLUTIONS

Dijkstra's shortest path

```
var graph = new Dictionary<string, List<(string to, int weight)>> {
    ["A"] = new() { ("B", 4), ("C", 2) },
    ["B"] = new() { ("A", 4), ("D", 3), ("C", 1) },
    ["C"] = new() { ("A", 2), ("B", 1), ("D", 5) },
    ["D"] = new() { ("B", 3), ("C", 5) },
};

var dist = Dijkstra("A");
foreach (var (node, d) in dist) {
    Console.WriteLine($"A -> {node}: {d}");
}

Dictionary<string, int> Dijkstra(string start) {
    var dist    = graph.Keys.ToDictionary(k => k, _ => int.MaxValue);
    var visited = new HashSet<string>();
    dist[start] = 0;

    while (visited.Count < graph.Count) {
        string? current = null;
        int     minDist = int.MaxValue;

        foreach (var (node, d) in dist) {
            if (!visited.Contains(node) && d < minDist) {
                minDist = d;
                current = node;
            }
        }

        if (current == null) { break; }
        visited.Add(current);

        foreach (var (neighbour, weight) in graph[current]) {
            int newDist = dist[current] + weight;
            if (newDist < dist[neighbour]) {
                dist[neighbour] = newDist;
            }
        }
    }
    return dist;
}
```

Challenge 96 Web Server Log Analyser

PROBLEM

Parse a web server access log file. Display: total requests, top 5 most frequent IPs, top 5 most requested pages, count of each HTTP status code, and requests per hour.

EXAMPLE

```
Total requests: 8,432
Top IP: 192.168.1.1 (342 requests)
Top page: /index.html (1,204 hits)
Status 200: 7,891  Status 404: 312
```

MARK SCHEME

Correct parsing for all five statistics. Handles malformed lines without crashing. Top 5 lists sorted correctly.

EXTENSION

Flag any IP making more than 100 requests as a potential bot. Write a summary to a new file.

TEACHER NOTES

> Provide a sample log file in standard Apache or Nginx format. Students learn a great deal about real-world data from working with actual log formats.

SOLUTIONS

Interpreter for mini language

```
var variables = new Dictionary<string, int>();
string[] program = {
    "SET x 10",
    "SET y 5",
    "ADD x y",
    "PRINT x",
    "IF x 15 PRINT found",
};

foreach (string line in program) {
    string[] parts = line.Split(' ');
    string   op    = parts[0];

    if (op == "SET") {
        variables[parts[1]] = int.Parse(parts[2]);
    } else if (op == "ADD") {
        variables[parts[1]] += variables[parts[2]];
    } else if (op == "PRINT") {
        string val = parts[1];
        Console.WriteLine(variables.TryGetValue(val, out int v) ? v.ToString() : val);
    } else if (op == "IF") {
        int lhs = variables.TryGetValue(parts[1], out int l) ? l : int.Parse(parts[1]);
        int rhs = variables.TryGetValue(parts[2], out int r) ? r : int.Parse(parts[2]);
        if (lhs == rhs) {
            string remaining = string.Join(" ", parts[3..]);
            // Execute the then-clause as a nested instruction
            string[] inner = remaining.Split(' ');
            if (inner[0] == "PRINT") {
                Console.WriteLine(inner[1]);
            }
        }
    }
}
```

Challenge 97

Constraint-Based Timetabler

PROBLEM

Given teachers (each assigned one subject) and rooms, generate a one-day timetable across 6 periods. Constraints: no teacher double-booked; no room double-booked. Display the timetable as a grid.

EXAMPLE

```
Period 1: Room 101 - Maths (Mr Smith)
Period 1: Room 102 - English (Ms Jones)
...
```

MARK SCHEME

No constraint violations in output. All teachers scheduled. Grid display clear and readable.

EXTENSION

Use backtracking to find a valid schedule when greedy assignment produces a conflict.

TEACHER NOTES

> Links well with the operating systems topic on scheduling. Students who reach the backtracking extension are working well above GCSE level.

SOLUTIONS

Recursive descent expression parser

```
Console.WriteLine(Evaluate("3 + 4 * 2"));   // 11
Console.WriteLine(Evaluate("(3 + 4) * 2")); // 14
Console.WriteLine(Evaluate("10 - 3 - 2"));  // 5

double Evaluate(string expr) {
    var tokens = Tokenise(expr);
    int pos    = 0;
    return ParseExpr(tokens, ref pos);
}

double ParseExpr(string[] tokens, ref int pos) {
    double left = ParseTerm(tokens, ref pos);
    while (pos < tokens.Length && (tokens[pos] == "+" || tokens[pos] == "-")) {
        string op = tokens[pos++];
        double right = ParseTerm(tokens, ref pos);
        left = op == "+" ? left + right : left - right;
    }
    return left;
}

double ParseTerm(string[] tokens, ref int pos) {
    double left = ParseFactor(tokens, ref pos);
    while (pos < tokens.Length && (tokens[pos] == "*" || tokens[pos] == "/")) {
        string op    = tokens[pos++];
        double right = ParseFactor(tokens, ref pos);
        left = op == "*" ? left * right : left / right;
    }
    return left;
}

double ParseFactor(string[] tokens, ref int pos) {
    if (tokens[pos] == "(") {
        pos++;
        double val = ParseExpr(tokens, ref pos);
        pos++;  // consume ')'
        return val;
    }
    return double.Parse(tokens[pos++]);
}

string[] Tokenise(string expr) {
    return expr.Replace("(", " ( ").Replace(")", " ) ")
               .Split(' ', StringSplitOptions.RemoveEmptyEntries);
}
```

Challenge 98

Multi-File OOP School System

PROBLEM

Design a school management system across four files: Models.cs (Student, Teacher, ClassGroup classes), Database.cs (save/load from CSV), Interface.cs (menu-driven UI), and Program.cs (entry point).

EXAMPLE

```
File structure:
Models.cs -> class definitions
Database.cs -> file I/O
Interface.cs -> menus and display
Program.cs -> initialises and launches
```

MARK SCHEME

Correct modular structure. Classes have constructors and meaningful methods. File I/O persists between runs. Files compile and reference each other correctly.

EXTENSION

Add comprehensive input validation. Add a statistics function showing average class sizes.

TEACHER NOTES

> This task mirrors the kind of modular design expected in A-Level NEA projects. Discuss the separation of concerns principle before students start.

SOLUTIONS

Pathfinding — A* algorithm

```
int[,] grid = {
    {0,0,0,0,0},
    {0,1,1,1,0},
    {0,0,0,1,0},
    {0,1,0,0,0},
    {0,0,0,0,0},
};

var path = AStar(grid, (0,0), (4,4));
if (path != null) {
    Console.WriteLine($"Path length: {path.Count}");
    foreach (var (r,c) in path) { Console.Write($"({r},{c}) "); }
    Console.WriteLine();
} else {
    Console.WriteLine("No path found.");
}

List<(int,int)>? AStar(int[,] g, (int r,int c) start, (int r,int c) goal) {
    int rows = g.GetLength(0), cols = g.GetLength(1);
    int H((int r,int c) p) => Math.Abs(p.r - goal.r) + Math.Abs(p.c - goal.c);

    var open    = new SortedSet<(int f, int r, int c)>(Comparer<(int,int,int)>.Create((a,b) =>
        a.Item1 != b.Item1 ? a.Item1.CompareTo(b.Item1) :
        a.Item2 != b.Item2 ? a.Item2.CompareTo(b.Item2) :
        a.Item3.CompareTo(b.Item3)));
    var gScore  = new Dictionary<(int,int), int>();
    var cameFrom = new Dictionary<(int,int), (int,int)>();

    gScore[start] = 0;
    open.Add((H(start), start.r, start.c));

    int[][] dirs = { new[]{0,1}, new[]{0,-1}, new[]{1,0}, new[]{-1,0} };

    while (open.Count > 0) {
        var (_, cr, cc) = open.Min;
        open.Remove(open.Min);
        (int,int) current = (cr, cc);

        if (current == goal) {
            var path = new List<(int,int)>();
            var node = goal;
            while (node != start) { path.Add(node); node = cameFrom[node]; }
            path.Add(start);
            path.Reverse();
            return path;
        }

        foreach (var d in dirs) {
            int nr = cr + d[0], nc = cc + d[1];
            if (nr < 0 || nr >= rows || nc < 0 || nc >= cols || g[nr,nc] == 1) { continue; }
            (int,int) nb = (nr, nc);
            int tentative = gScore[current] + 1;
            if (tentative < gScore.GetValueOrDefault(nb, int.MaxValue)) {
                cameFrom[nb] = current;
                gScore[nb]   = tentative;
                open.Add((tentative + H(nb), nr, nc));
            }
        }
    }
    return null;
}
```

Challenge 99 k-Nearest Neighbours

PROBLEM

Implement the k-NN classification algorithm from scratch (no external ML libraries). Use the Iris dataset as a CSV. Split into training and test sets. Classify each test point and report overall accuracy.

EXAMPLE

```
k=3
Test set accuracy: 96.7%
Predicted: setosa, Actual: setosa
```

MARK SCHEME

Correct Euclidean distance. Correct k nearest identified. Majority vote produces prediction. Accuracy calculated over full test set.

EXTENSION

Test with k = 1, 3, 5, 7, 9. Display accuracy for each value and identify the best k.

TEACHER NOTES

> Works well alongside an introduction to machine learning concepts. Provide the Iris dataset as a CSV file.

SOLUTIONS

Cellular automaton — Conway's Game of Life

```
int rows = 10, cols = 10;
bool[,] grid = new bool[rows, cols];

// Glider
grid[1,2]=true; grid[2,3]=true; grid[3,1]=true; grid[3,2]=true; grid[3,3]=true;

for (int gen = 0; gen < 5; gen++) {
    Console.WriteLine($"--- Generation {gen} ---");
    Print(grid, rows, cols);
    grid = NextGen(grid, rows, cols);
}

bool[,] NextGen(bool[,] g, int r, int c) {
    bool[,] next = new bool[r, c];
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            int neighbours = CountNeighbours(g, i, j, r, c);
            if (g[i, j]) {
                next[i, j] = neighbours == 2 || neighbours == 3;
            } else {
                next[i, j] = neighbours == 3;
            }
        }
    }
    return next;
}

int CountNeighbours(bool[,] g, int row, int col, int rows, int cols) {
    int count = 0;
    for (int dr = -1; dr <= 1; dr++) {
        for (int dc = -1; dc <= 1; dc++) {
            if (dr == 0 && dc == 0) { continue; }
            int r = row + dr, c = col + dc;
            if (r >= 0 && r < rows && c >= 0 && c < cols && g[r, c]) {
                count++;
            }
        }
    }
    return count;
}

void Print(bool[,] g, int r, int c) {
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            Console.Write(g[i, j] ? "O" : ".");
        }
        Console.WriteLine();
    }
}
```

Challenge 100

Project Scoping Exercise

PROBLEM

This challenge has no code to write. Choose a real problem to solve with C# and produce: (1) a problem statement, (2) a named stakeholder and their requirements, (3) six measurable success criteria, (4) a list of C# techniques required, (5) pseudocode for the most complex part, (6) a risk assessment with two risks and mitigations.

EXAMPLE

```
This is a planning and analysis exercise that mirrors the A-Level NEA Analysis and
Design sections.
```

MARK SCHEME

Problem is specific and real. Stakeholder is genuine (not invented). All six criteria are measurable. Techniques are appropriate. Algorithm is detailed enough to code from. Risks are plausible with realistic mitigations.

EXTENSION

Build it.

TEACHER NOTES

> This task is best used as a transition exercise before starting an NEA or coursework project. Students who struggle to identify a real problem benefit from a short class discussion about problems they encounter in daily life.

SOLUTIONS

Mini relational database

```
var db = new SimpleDB();
db.CreateTable("students");
db.CreateTable("courses");

db.Insert("students", new() { ["id"]="1", ["name"]="Alice", ["age"]="17" });
db.Insert("students", new() { ["id"]="2", ["name"]="Bob",   ["age"]="16" });
db.Insert("courses",  new() { ["id"]="1", ["title"]="CS",   ["student_id"]="1" });
db.Insert("courses",  new() { ["id"]="2", ["title"]="Maths",["student_id"]="2" });

Console.WriteLine("All students:");
db.Select("students").ForEach(r => Console.WriteLine(string.Join(", ", r.Select(kv => $"{kv.Key}
={kv.Value}"))));

Console.WriteLine("\nStudents aged 17:");
db.Where("students", "age", "17").ForEach(r => Console.WriteLine(r["name"]));

Console.WriteLine("\nJoin students with courses:");
foreach (var s in db.Select("students")) {
    var course = db.Where("courses", "student_id", s["id"]).FirstOrDefault();
    if (course != null) {
        Console.WriteLine($"{s["name"]} -> {course["title"]}");
    }
}

class SimpleDB {
    Dictionary<string, List<Dictionary<string,string>>> tables = new();

    public void CreateTable(string name) {
        tables[name] = new();
    }

    public void Insert(string table, Dictionary<string,string> row) {
        tables[table].Add(row);
    }

    public List<Dictionary<string,string>> Select(string table) {
        return tables[table];
    }

    public List<Dictionary<string,string>> Where(string table, string key, string value) {
        var result = new List<Dictionary<string,string>>();
        foreach (var row in tables[table]) {
            if (row.TryGetValue(key, out string? v) && v == value) {
                result.Add(row);
            }
        }
        return result;
    }
}
```

CodeBash

The interactive coding platform for UK schools

codebash.co.uk