CodeBash · codebash.co.uk

FREE RESOURCE

100 C# Programming Challenges

A taster selection from the full 100-challenge pack - free for CS teachers

Created by Eoin Shannon

CodeBash

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

How to Use This Pack

Each challenge includes a problem statement (written in plain language), an example showing expected input and output, hints for differentiation, mark scheme notes, and an extension task for students who finish early.

The hints column can be withheld from more confident students. Mark scheme notes describe what a correct solution must achieve - they are intentionally brief so this document remains usable as a classroom handout.

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#.
```

HINTS

Use Console.ReadLine() to collect the name. Use string interpolation ($"Hello, {name}!") to build the message.

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.

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.
```

HINTS

Use int.Parse(Console.ReadLine()!) to convert the input. Subtract birth year from the current year (2025).

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.

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
```

HINTS

Area = length x width. Perimeter = 2 x (length + width). Use double.Parse() for inputs.

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.

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.
```

HINTS

Use the modulo operator %. If number % 2 == 0 it is even.

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.

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
```

HINTS

Use double.Parse() for input. Apply the formula. Display to 1 decimal place using :F1 in the format string.

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.

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.
```

HINTS

Use .Length for length. Use password.Any(char.IsDigit) to check for a digit. Use password.Any(char.IsUpper) for uppercase.

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.

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
```

HINTS

Use a for loop: for (int i = 1; i <= 12; i++).

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.

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.
```

HINTS

Use Random rng = new(); and rng.Next(1, 11) (upper bound is exclusive). Use a while loop. Count guesses with a variable.

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.

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...
```

HINTS

Check for multiples of both 3 and 5 first. Use the % operator. Use if / else if / else.

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.

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
```

HINTS

Use a while loop. Use double.Parse() to convert prices. Accumulate total with +=.

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.

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
```

HINTS

Use else if for each range. Handle invalid input (below 0 or above 100).

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.

Challenge 12 Factorial

PROBLEM

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

EXAMPLE

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

HINTS

Factorial of n = n x (n-1) x ... x 1. Start with result = 1 and multiply in a loop.

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.

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!
```

HINTS

Use a for loop counting down: for (int i = n; i >= 1; i--). Print 'Blast off!' after the loop.

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.

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
```

HINTS

Convert to a string and iterate through characters. Use int.Parse(c.ToString()) to convert each character back to int.

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.

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
```

HINTS

Use a string array indexed from 0. Index 0 can store an empty string. Access with words[n].

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.

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
```

HINTS

Accumulate the total and count separately. Divide at the end. Use double.Parse() for conversion.

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.

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.
```

HINTS

Reverse a string using new string(word.Reverse().ToArray()). Convert to lowercase with .ToLower() before comparing.

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.

Challenge 18 Coin Toss Simulator

PROBLEM

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

EXAMPLE

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

HINTS

Use Random rng = new(); and rng.Next(0, 2) (returns 0 or 1). Use a loop and two counters.

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.

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
```

HINTS

Use double.Parse() for inputs. Round BMI to 1 decimal place with Math.Round(bmi, 1).

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.

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
```

HINTS

Outer loop for rows, inner loop for numbers in each row. Use Console.Write() (without newline) for numbers, Console.WriteLine() at end of each row.

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.

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.
```

HINTS

Check: (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0).

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.

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
```

HINTS

Use .Split(' ') to get an array of words. Use Array.Reverse() to reverse. Use string.Join(' ', words) to reassemble.

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.

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.
```

HINTS

Use Random rng = new(); and rng.Next(1, 13). Use a loop. Compare user input (converted with int.Parse()) against the correct answer.

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.

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
```

HINTS

Define conversion rates as const double variables at the top. Multiply the amount by each rate.

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.

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.
```

HINTS

A prime has no factors other than 1 and itself. Check divisibility from 2 up to (int)Math.Sqrt(n).

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.

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
```

HINTS

Check each character against "aeiouAEIOU" or use .ToLower() first. Use a counter variable.

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.

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
```

HINTS

Store in a List<double>. Set initial highest/lowest to the first element. Compare each subsequent element.

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.

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)
```

HINTS

Use two parallel List<string> and List<int>, or a List of tuples. Calculate average from accumulated total.

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.

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
```

HINTS

Cast chars to int with (int)c. Handle wrap-around using % 26. Cast back with (char).

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.

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
```

HINTS

Use a Dictionary<string, int>. Use .Split(' ') to get words. Use ContainsKey() to check, then increment.

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.

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
```

HINTS

Use new Random() and a string array of choices. Use rng.Next(0, 3) to pick the computer's move.

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.

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
```

HINTS

Check divisibility from 2 up to (int)Math.Sqrt(n) + 1. Return false immediately if a divisor is found.

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.

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
```

HINTS

Start with a List<int> containing 0 and 1. Each subsequent number is the sum of the two before it. Use list[^1] and list[^2] for the last two elements.

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.

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.
```

HINTS

Use a while loop with a menu. Convert amounts with double.Parse(). Check balance before allowing withdrawal.

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.

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
```

HINTS

Use .Length for length. Use .Split(' ').Length for word count. Use .Count(char.IsUpper) from LINQ for uppercase.

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.

Challenge 36 Number to Roman Numerals

PROBLEM

Convert a number between 1 and 3999 to Roman numerals.

EXAMPLE

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

HINTS

Use two arrays: one for values (int[]), one for symbols (string[]). Work from the largest value down.

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.

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
```

HINTS

A stack is Last In, First Out. Use list.Add() for push and list[^1] with list.RemoveAt() for pop.

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.

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'
```

HINTS

A queue is First In, First Out. Enqueue adds to the back (Add), Dequeue removes from the front (RemoveAt(0)).

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.

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
```

HINTS

Sort the letters of both words and compare using string.Concat(). Use .ToLower() before sorting.

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.

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
```

HINTS

Use repeated division by the target base. Collect remainders in reverse order. Use a digit string "0123456789ABCDEF" for hex.

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.

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
```

HINTS

Euclidean algorithm: Hcf(a, b) = Hcf(b, a % b) until the remainder is 0.

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.

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
```

HINTS

For each word, keep index 0 and the last index in place. Shuffle the middle by converting to a List<char> and sorting with OrderBy(x => rng.Next()).

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.

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]]
```

HINTS

Use int[,] for a 2D array, or int[][] for a jagged array. Use nested loops to iterate rows and columns.

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.

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
```

HINTS

Define the class with a constructor. Methods should update the balance field. Use this. to refer to instance fields.

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.

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
```

HINTS

Loop through each element with its index. Return the index as soon as a match is found.

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.

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
```

HINTS

Use low, high, and mid pointers. Compare target with the middle element each iteration.

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.

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]
```

HINTS

Use nested loops. In each pass, compare adjacent elements and swap using a tuple swap: (arr[j], arr[j+1]) = (arr[j+1], arr[j]).

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.

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"]
```

HINTS

For each element, find its correct position in the already-sorted left portion and insert it. Use string.Compare() or the < operator for alphabetical comparison.

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.

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]
```

HINTS

Split the list in half recursively until single elements remain. Merge sorted halves back together.

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.

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
```

HINTS

Use first[0] for the first character. Use .ToLower(). Use (year % 100).ToString("D2") for the two-digit year.

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.

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
```

HINTS

Valid Roman numerals follow specific ordering and repetition rules. Use Regex.IsMatch() with a pattern that encodes the rules.

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.

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
```

HINTS

Store temperatures in a double[7] array. Calculate average first, then use loops for the other stats.

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.

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
```

HINTS

Split on space to get two parts. Check the inward code (second part) is 3 characters: digit then two letters. Use char.IsDigit() and char.IsLetter().

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.

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
```

HINTS

Store guessed letters in a List<char>. Build the display string using a foreach loop - show the letter if guessed, otherwise '_'.

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.

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
```

HINTS

Use a Dictionary<string, string> where the key is the name. Show a menu with a while loop. Use ContainsKey() to check for existing entries.

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.

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)
```

HINTS

Use File.ReadAllLines(). Split each line with .Split() to get words. Use Dictionary<string, int> to count frequencies.

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.

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
```

HINTS

Use File.ReadAllLines() and .Split(',') on each line. Use .Trim() to remove whitespace.

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).

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
```

HINTS

Use File.AppendAllText() to add scores. Use File.ReadAllLines() to read them back. Sort using LINQ's OrderByDescending().

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.

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.
```

HINTS

Use try/catch (FormatException) around double.Parse(). Check for division by zero separately with an if statement.

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.

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
```

HINTS

Use File.ReadAllLines() and check the start of each line with .StartsWith(). Store error lines in a List<string>.

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.

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
```

HINTS

Define the class with a constructor. Add methods that operate on the grades list. Override ToString() to display details.

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.

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.
```

HINTS

Use class Dog : Animal to inherit. Mark the base method as virtual and override it in each subclass.

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.

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
```

HINTS

Node class has a Value field and a Next? field (nullable). LinkedList has a Head? field. Traverse using a while loop.

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.

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
```

HINTS

Each node has Value, Left?, and Right?. Insert by comparing with the current node value. Use recursion for traversal.

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.

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
```

HINTS

Use a List<List<(string, string)>> for the table. Hash function: sum(ASCII values) % table_size.

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.

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]
```

HINTS

Use a Dictionary<string, List<string>> where keys are node names and values are lists of connected nodes.

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.

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)
```

HINTS

Use a Dictionary<string, int> for distances (initialised to int.MaxValue). Always process the unvisited node with the smallest known distance.

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.

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.
```

HINTS

Base case: move 1 disc directly, return 1. Recursive case: move n-1 discs to spare, move largest, move n-1 back. Return count + 1.

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.

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.
```

HINTS

Try moving in each direction recursively. Use a HashSet<(int,int)> to mark cells as visited. Return false to backtrack.

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.

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
```

HINTS

Each letter is shifted by the corresponding key letter ((key[i % key.Length] - 'a')). Use (int) casts. Handle modulo for wrap-around.

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.

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"}
```

HINTS

Store records in a Dictionary keyed by int ID. Return a (result, statusCode) tuple from each function.

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.

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)
```

HINTS

Use Dictionary<string, (int qty, double price)> or a nested Dictionary. Check quantity before selling.

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.

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"
```

HINTS

Loop through characters. Keep a count of consecutive identical characters. Build the result string as you go.

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.

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
```

HINTS

For decimal to other: use repeated division. For other to decimal: multiply each digit by its positional value (base^position).

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.

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
```

HINTS

Sort tasks by arrival time. Track current time. Waiting time = start time - arrival time.

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.

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
```

HINTS

Strip all non-digit characters using Where(char.IsDigit). Handle +44 prefix. Check length is 11.

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.

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
```

HINTS

Store room data in a Dictionary<string, Room> where Room is a class or struct with description, exits, and items.

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.

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
```

HINTS

Convert to lowercase and split on spaces. Count how many words appear in each list using .Contains().

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.

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
```

HINTS

Use a Dictionary<(string room, string day, int period), string> keyed by a value tuple. Check ContainsKey() before adding.

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.

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
```

HINTS

Use recursion. Handle base cases: both empty = true, pattern empty but string not = 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.

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
```

HINTS

Use a List<Book> where Book is a class with title, author, ISBN, and available fields.

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.

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
```

HINTS

Use a List of record types or classes with a string[] marks array. Write CSV with a header row. Use File.Exists() on startup.

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.

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
```

HINTS

*Convert times to minutes since midnight for comparison: int.Parse(parts[0]) * 60 + int.Parse(parts[1]).*

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.

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
```

HINTS

Encrypt the password field only. Use a fixed shift stored as a const in the program.

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.

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!
```

HINTS

Use a char[9] or string[9] for the board. Check all 8 win conditions (3 rows, 3 columns, 2 diagonals) in a helper function.

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.

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)
```

HINTS

Use a List<Patient> where Patient is a class/struct. Sort by priority then by arrival index.

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.

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
```

HINTS

Edit distance: count insertions, deletions, and substitutions needed to transform one word to another using a 2D dp array.

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.

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
```

HINTS

Use a Dictionary<string, string[]> for categories and their keyword arrays. Check each keyword with .Contains() on the lowercased description.

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.

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
```

HINTS

Write a recursive permutation generator over digits 0-9. Assign in order S,E,N,D,M,O,R,Y. Skip if S==0 or M==0.

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.

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
```

HINTS

Check if the value is a Dictionary to identify a folder. Use a string of spaces (string(' ', indent)) for indentation.

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.

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%
```

HINTS

Compression ratio = compressed_length / original_length. Lower is better. Build the bar chart using string repetition.

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.

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
```

HINTS

Use a Queue<int> for arrival times. Advance time in one-minute increments. Track when the server becomes free.

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.

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
```

HINTS

Derivative: multiply each coefficient by its power and reduce all powers by 1 (remove index 0). Multiplication: use a result array of length len(a)+len(b)-1.

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.

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
```

HINTS

Use a Dictionary<string, HashSet<string>> for bidirectional edges. Use BFS (Queue<List<string>>) for the shortest path.

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.

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
```

HINTS

Use Player and Enemy classes. Damage formula: attacker.Attack - defender.Defence (minimum 1). Use System.Text.Json.JsonSerializer for save/load.

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.

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
```

HINTS

Split each log line by spaces to extract IP (index 0), request (index 6), and status code (index 8). Use Dictionary<string, int> for counting.

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.

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)
...
```

HINTS

Use a Dictionary<(int period, string room), string> for the timetable. Check both dictionaries before each assignment.

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.

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
```

HINTS

Use a namespace to group your classes. Avoid circular dependencies by keeping Models.cs free of other class imports.

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.

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
```

HINTS

Euclidean distance: Math.Sqrt(sum of squared differences between features). Take majority vote from k nearest using a Dictionary<string, int>.

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.

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.
```

HINTS

Every success criterion must be testable with a clear pass/fail outcome. Avoid vague objectives like 'the system should be easy to use'.

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.

CodeBash

The interactive coding platform for UK schools

codebash.co.uk