CodeBash · codebash.co.uk

FREE RESOURCE

20 C# Programming Challenges - Sample Pack

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 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 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 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 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.

Intermediate

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

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 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 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 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 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 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.

Applied

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

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 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 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 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.

Stretch Challenges

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

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 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 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 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 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.

CodeBash

The interactive coding platform for UK schools

codebash.co.uk
