Practice Lab 7: Arrays and Lists

The goal of this lab is to understand how you can store multiple variables of the same type together in an array. You will learn how to declare an array of variables, how to access and manipulate those variables using square brackets, and how to iterate over those variables using a loop.

Arrays

So far, you've learnt about how to store data in variables, manipulate that data with operations, make decisions with conditionals and repeat things using loops. The last big piece of the puzzle is storing several pieces of data in the same variable. That's not to say there is nothing left to learn – there's still a long way to go, but with just these few building blocks, you can write code to do pretty much anything. To store several pieces of data in the same variable, you can use an array. When you declare an array, you create a block of contiguous memory to store a specified number of variables of the same type. In this lab, you'll learn how to declare an array, access and manipulate individual elements of that array.

Initial Task - Before cloning the repository

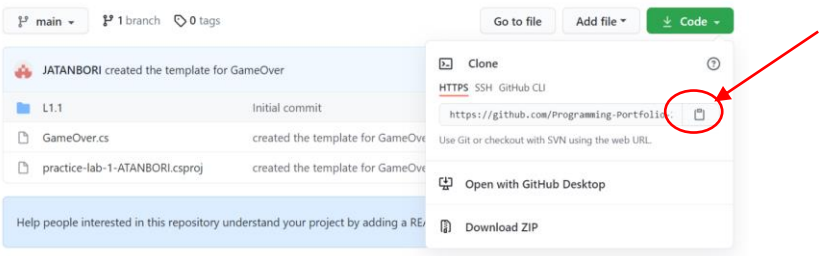
To complete the tasks for this Practice Lab, you should have created a GitHub account and joined the GitHub classroom specified above. We will use GitHub from within Visual Studio Code (VSCode), which will enable us to use some of your favourite parts of GitHub.

Cloning the repository

1. Create a Local folder to store these practice exercises.
2. Ensure all folders are closed in VSCode (File -> Close Folder)
3. From the activity, bar click the explorer icon (ctrl + shift + E)
4. Click to view your repository, then copy the link to your repository to the clipboard.

   https://classroom.github.com/a/NtkSHG3M

5. Click the Clone Repository button from the source control panel in VSCode.
6. When prompted to enter the repository URL, paste the data in the clipboard from step 4.



7. Select destination local copy folder and press the Select Repository Location button.

L7.1 Find Maximum Value

There is one more way to iterate over all the values in an array (and a few other C# collections). The for-each loop can be used to create a variable of the same type as the elements in the array, and then for each iteration of the loop the corresponding value in the array is assigned to the variable created by the loop.

```
foreach (uint value in values)
{

}
```

For example, in the code above a temporary variable of type uint (unsigned integer) with the identifier value is created, then the code block repeats for each element in the array of unsigned integers with the value of the corresponding uint in the array being assigned to the temporary variable – so if the values array contains elements with values of 1, 3, 5 and 7 the code block will be repeated four times, with value being assigned the value 1, 3, 5 and 7 respectively.

In the solution explorer, navigate to "L7.1 – FindMaximumValue." and open the .cs file. Look at the code in the project. An array of unsigned integers has been created with some values. You're going to write for-each loop to find the maximum value. We want to find the maximum value in the loop – so you should create a variable to hold the maximum value. Create a variable of type uint with the identifier maximumValue. Assign a value of zero to that variable. Then add the code from the screenshot.

```
foreach (uint value in values)
{

}
```

Inside the code block, write a conditional statement that checks to see if the value is greater than maximumValue. If it is, assign the value of the variable value to the variable maximumValue.
Finally, after the loop, output maximumValue to the console. You should find the maximum value is 3643574.

```
The Maximum value is 3643574
```

Test your code, and once you are satisfied that you understand how to declare an array, access elements within an array and modify an array then, push your changes to GitHub with an appropriate log message.

L7.2 Largest Difference

In the solution explorer, navigate to “L7.2 – LargestDifference.” and open the .cs file.

Look at the code. You will see an array of elements of the type byte, with the identifier costs. A byte is an integer type that can store values from 0 to 255. This is much smaller in memory than an int, which is 4 times as large – but can store a far wider range of numbers.

Create variables to store the Maximum and the Minimum values in the array. Then write a for-each-loop to iterate over the values in the costs array. For each value check if it is higher than the current maximum value you have found, or lower than the current minimum value. In each case, if you find a higher or lower value replace the current maximum or minimum value with the new value you have found.

At the end of the loop subtract the minimum value from the maximum and output the difference.

```
The Largest Difference is 231
```

Test your code and if the program works as expected, add the changes to the staging area. Next, commit the changes to the local repository with an appropriate message.

Finally, push the changes into GitHub.

L7.3 – TicTacToe

In the solution explorer, navigate to “L7.3-TicTacToe.” and open the .cs file.

Now you’re going to bring all the things you’ve learned together whilst also learning about a new concept, called multidimensional arrays. They allow you to declare arrays that seem to have additional structural elements, like being on a grid. For example, you can declare an array that represents a 2-by-3 grid, and it will contain a total of 6 elements, together with ways to access those elements using dimensions as if it had “rows” and “columns”. You can create multidimensional arrays that have any number of dimensions – 2D, 3D, 5D – whatever. It is rare to go above 2D and even more rare to go past 3D because of the efficiency of dealing with such structures. If you have a 5D array and you double each dimension the overall size of the array will increase by a factor of 32 (2^5) which is bad.

We’re going to make a game of Tic Tac Toe. In Tic Tac Toe, 2 players take turns to claim a space on a 3x3 grid, marking it with either "O" or "X". You cannot claim a space that has already been claimed. The winner is the first player to get three of their symbols in a row,

column or diagonal line. If neither player does this, and there are no more moves that can be made then the game is a draw.

The project is empty. First let's think about the steps we need to take.

1. Set up the board
2. Display the board to the players
3. Ask player 1 to space to play in (1-9)
4. Check that is a valid selection – if not ask them to go again (returning to 2 - this is a loop)
5. Check to see if the game has been won – if it has congratulate player 1
6. Change player to player 2
7. Ask player 2 to pick a row
8. Ask player 2 to pick a column
9. Check that is a valid selection – if not ask them to go again (returning to 2 - this is a loop)
10. Check to see if the game has been won – if it has congratulate player 2
11. Go back to 2

Note how steps 8-11 mirror steps 3-6. It would be better if we stored a variable to remember whose turn it was. Then we could do the following

1. Set up the board
2. Display the board to the players
3. Ask player 1 to space to play in (1-9)
4. Check that is a valid selection – if not ask them to go again (returning to 2 - this is a loop)
5. Check to see if the game has been won – if it has congratulate current player
6. Change current player to the other player
7. Go back to 2

Setting up the board.

Add the following code, which declares and 2D array of char (a character) that will be used to store the board state. The board array is initialised with values 1 – 9. The current player is also stored in a char variable with the identifier "currentPlayer" that will be either 'O' or 'X' – we assign 'O' to this indicating that the 'O' player will go first.

```
char[,] board = {{'1','2','3'}, {'4','5','6'}, {'7','8','9'}};
char currentPlayer = 'O';
```

Next we need a game loop – the loop that deals with steps 2 to 8. To start with create an empty loop.

```
while (true)
{

}
```

Display the board

Inside the loop first clear the Console of anything that has gone before. Then add a nested for loop which will step through each column on each row, and output the contents of the multidimensional array. The GetLength(0) and GetLength(1) methods will return the individual dimensions of the array, in a smilar way to GetLength. Similarly, the accessor for the array [rows, cols] will give us access to the row and column specified by the rows and cols variables, so if rows is 0 and cols is 0 then we will get the top left corner element. If rows is 1 and cols is 1 we get the middle element.

```
Console.Clear();

for (int rows = 0; rows < board.GetLength(0); rows++)
{
    for (int cols = 0; cols < board.GetLength(1); cols++)
    {
        Console.Write(board[rows, cols]);
    }
}
```

If you run the code the console will appear flickery, because it is updating so fast. To stop this add the following code at the end (but still inside) the while loop.

```
Console.WriteLine("Press any key to continue...");
Console.ReadKey();
```

You should see the following

```
123456789Press any key to continue...
```

This is not exactly what we want. Instead at the end of each row we want to drop down a row, to create a grid.

Change the code to add a Console.WriteLine() at the end of each row. Then test your code.

```
Console.Clear();

for (int rows = 0; rows < board.GetLength(0); rows = rows + 1)
{
    for (int cols = 0; cols < board.GetLength(1); cols = cols + 1)
    {
        Console.Write(board[rows, cols]);
    }
    Console.WriteLine(); // add this line
}

Console.WriteLine("Press any key to continue");
Console.ReadKey();
```

Think about how this works. The outer for loop will start each row, then the inner for loop will be looped three times, adding each column for the current row. Then the inner loop will end, and the Console.WriteLine will move us to the next line (or row). Then we will enter the inner loop again. The initialisation variable will be reset back to zero and well will iterate through each column on the second row. Then we move down another line, and repeat again for the third row.
You should end up with something like this.

```
123
456
789
```

Ask Player to select an available space

Next we need to ask the current player to select an available space. We will also have to validate that selection and keep asking the player to give input until the input is valid. Create a do while loop to control that process. To begin with assume that the input will not be invalid.

```
bool invalidInput;

do
{
    invalidInput = false;

} while (invalidInput);
```

Inside this loop add the following code. This will prompt the current player to pick a space.

```
Console.WriteLine("Player " + currentPlayer + " please pick a space.");
int inputInt = int.Parse(Console.ReadLine());
```

Check that is a valid selection

Next we need to check to see if the space the player picked is valid or not. Add the following code. This will set the invalidInput variable to true if the input is more than 9 or less than 1.

```
if (inputInt > 9 || inputInt < 1)
{
    invalidInput = true;
    continue;
}
```

After that, we also need to check whether the chosen space already contains an ‘X’ or an ‘O’. To do this we need to convert the number from a 1-9 to a row and column. We can also use the continue keyword to skip the rest of the loop.

We know that numbers 1-3 are row 0, numbers 4-6 are row 1 and numbers 7-9 are row 3, so if we take 1 away from inputInt and divide by 3 we will get the row number. Similarly, if we take 1 away from inputInt and modulus with 3 we will get the column number. Then we can check the element to see if it contains a ‘O’ or an ‘X’ – and if it does the choice is not valid. If it is not valid we set the invalidInput bool to false and use the continue keyword to skip the rest of the loop.

```
int row = (inputInt - 1) / 3;
int col = (inputInt - 1) % 3;

if (board[row, col] == 'O' || board[row, col] == 'X')
{
    invalidInput = true;
    continue;
}
```

If the choice is valid you can assign the character at the chosen row and column in the board to the currentPlayer character.

```
board[row, col] = currentPlayer;
```

Check to see if the game has been won

When we escape the input loop we know that the current player made a move. We now need to check to see if there is a winner, or if there is a draw. In either case we will report the result to the player and then break out of the game loop.

First we can check the rows, then the columns, then the two diagonals. If this conditional looks big and messy to you you're right! There are arguably better ways to write this code – especially if they allow us to change the dimensions of the board – but for now we'll go with messy!.

```
if( (board[0,0] == board[0,1] && board[0,0] == board[0,2]) ||
    (board[1,0] == board[1,1] && board[1,0] == board[1,2]) ||
    (board[2,0] == board[2,1] && board[2,0] == board[2,2]) ||

    (board[0,0] == board[1,0] && board[0,0] == board[2,0]) ||
    (board[1,0] == board[1,1] && board[0,1] == board[2,1]) ||
    (board[0,2] == board[1,2] && board[0,2] == board[2,2]) ||

    (board[0,0] == board[1,1] && board[0,0] == board[2,2]) ||
    (board[0,2] == board[1,1] && board[0,2] == board[2,0])

)
{
    Console.WriteLine("Player " + currentPlayer + " wins!");
    break;
}
```

If there is no winner yet, you still need to check to see whether there are any spaces left to go in. To do this we can iterate over every space until we find a space that does not contain a 'X' or an 'O'. First create a bool variable with the identifier "isDraw" and set it to true. Then when we find an 'X' or an 'O' we can set it to false.

```
bool isDraw = true;

for (int rows = 0; rows < board.GetLength(0); rows = rows + 1)
{
    for (int cols = 0; cols < board.GetLength(1); cols = cols + 1)
    {
        if(board[rows, cols] != 'X' && board[rows, cols] != 'O')
        {
            isDraw = false;
        }
    }
}

if(isDraw)
{
    Console.WriteLine("This is a draw!");
    break;
}
```

Change current player

Finally, if we reach the end of the while loop and the game has not ended we can switch the current player.

```
if(currentPlayer == 'O')
{
    currentPlayer = 'X';
}
else
{
    currentPlayer = 'O';
}
```

Test your code, and once you are satisfied that the code works, and that you understand how it works and how to declare a multidimensional array, accessing elements within a multidimensional array and modify a multidimensional array then push your changes to GitHub with an appropriate log message.

Assessment Block

L7.4 – Console-Based Shopping Basket and L7.5 – Hearts Mining Game are part of your portfolio submission and will be assessed towards the end of the trimester. For further details on the submission, see the assignment brief.

L7.4 – Console-Based Shopping Basket

We are going to write a console-based e-commerce product catalogue and shopping basket application. For simplification, we will assume that you will purchase items from a single category. Happy Days Games sell games to the general public in four categories: Playstation, Xbox, Nintendo and PC Games.

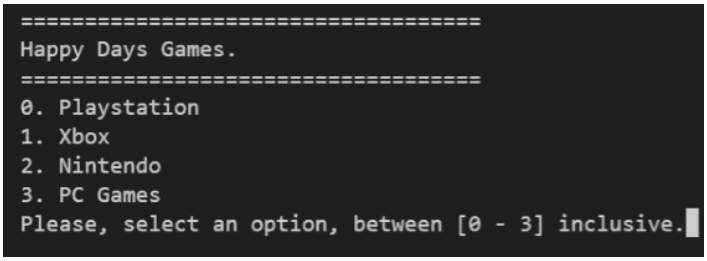
In the solution explorer, navigate to "L7.3-ConsoleShoppingBasket" and open the .cs file. You will also find two CSV files that contain the product categories and catalogue for Happy Days Games. Open the files and explore their contents.

We will develop a mini e-commerce console-based application that will display the product category pages to the customer. If the customer selects a category, the products catalogue for that category is displayed. The customer then selects and adds products to their shopping basket. If the entered option is outside the range of the displayed products, the application terminates and displays the content of their basket.

For this exercise, do not use struct and Enum types. You could make use of List Collection and Arrays. We will read the products and categories from the supplied CSV files (categories.csv and productCatalogue.csv). The only method in the program will be the "static void Main" method.

How the application will work.

1. When the application starts, the categories will be displayed for the customer to select one (see below).



2. When a customer selects a category, the program clears the console, and the products for the selected category is displayed.

```
============================================================
Product Catalogue.
============================================================
Option   | Category    | Product                                      |    Price
=======  | ========== | ============================================ | ========
      0. | Xbox        | Resident Evil Village                        |   £31.99
      1. | Xbox        | Mass Effect Legendary Edition                |   £29.99
      2. | Xbox        | Assassin's Creed Valhalla                    |   £35.99
      3. | Xbox        | Watch Dogs Legion                            |   £40.99
      4. | Xbox        | Cyberpunk 2077                               |   £12.99
      5. | Xbox        | F1 2020 Standard Edition                     |    £9.99
      6. | Xbox        | BIOMUTANT                                    |   £19.99
      7. | Xbox        | DOOM Eternal                                 |   £15.99
      8. | Xbox        | Planet Coaster                               |   £26.99
      9. | Xbox        | Monster Energy Supercross 4                  |   £18.99
Select a Product. Enter [0 - 9] inclusive.
```

3. The program prompts the customer to select a product by entering the option number (see below).

```
Select a Product. Enter [0 - 9] inclusive.0
Product Selected: Resident Evil Village
Select a Product. Enter [0 - 9] inclusive.4
Product Selected: Cyberpunk 2077
Select a Product. Enter [0 - 9] inclusive.8
Product Selected: Planet Coaster
Select a Product. Enter [0 - 9] inclusive.
```

4. The selected products are added to a shopping basket, and step 3 is repeated until the user enters an invalid product option.
5. When the user enters an invalid product option, the loop in step 3 terminates. The program clears the console, the products added to the shopping basket are displayed together with the total amount in the basket.

```
============================================================
Your Shooping basket.
============================================================
Resident Evil Village                          |   £31.99
Cyberpunk 2077                                 |   £12.99
Planet Coaster                                 |   £26.99
TOTAL AMOUNT:                                  |   £71.97
```

Test your program to ensure the categories, product catalogue, and shopping basket are displaying correctly, together with the correct total amount in the basket.

If the program works as expected, then add the changes to the staging area. Next, commit the changes to the local repository with an appropriate message.

Finally, push the changes into GitHub.

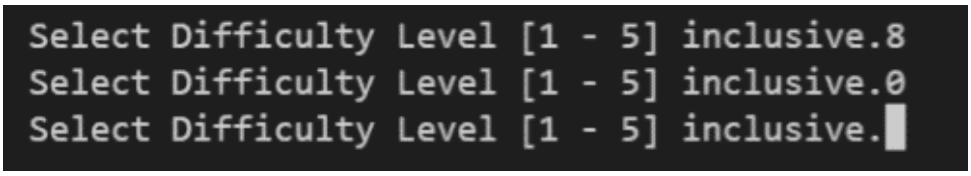
L7.5 – Hearts Mining Game

We have put together a hearts mining game for the programming portfolio module, which has a total of 24 spades and hearts buried in grid cells. There are four rows and six columns in the game grid. When a user selects an alphabet assigned to a grid cell, the item buried (spade or hearts) in that grid is revealed to them. The game aims to discover all the buried hearts with fewer grid cell selections. Your score increases when you get "hearts", while it decreases upon discovering "spades".

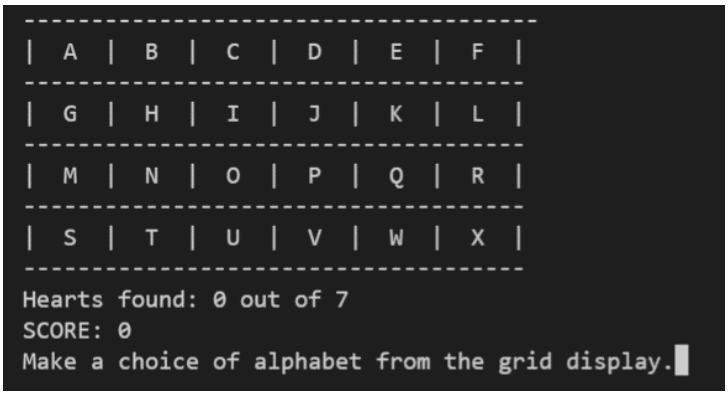
In the solution explorer, navigate to “L7.5-ConsoleHeartsMiningGame” and open the .cs file.

Now, write a program that will allow a user to play the game. Do not use struct and Enum types for this program. You could make use of Arrays and do not refactor your codes. The only method in the program will be the "static void Main" method. The program will have the following steps.

1. When the game starts, the player will have to select the difficulty level they want to play. The difficulty of the game should range from 1 to 5 inclusive. If a player selects a number outside of this range, the program should prompt them to enter the level again, which persists until they have entered the correct level (see below).



2. Your program should display the initial game grid, consisting of letters (alphabets) when the correct level is selected. The "total hearts" buried and the initial score of zero should also be displayed (see below). The program should then prompts the player to choose from the letters to reveal the treasure buried behind that letter (alphabet).



3. At the start, you should randomly bury the "hearts" and "spades" behind the alphabets. To implement this, you could use two arrays. The first array stores the alphabets displayed to the user, and the second stores Unicode characters of spades

and hearts, which is not visible to the player. The Unicode character for spades and hearts are '\u2660' and '\u2665', respectively.

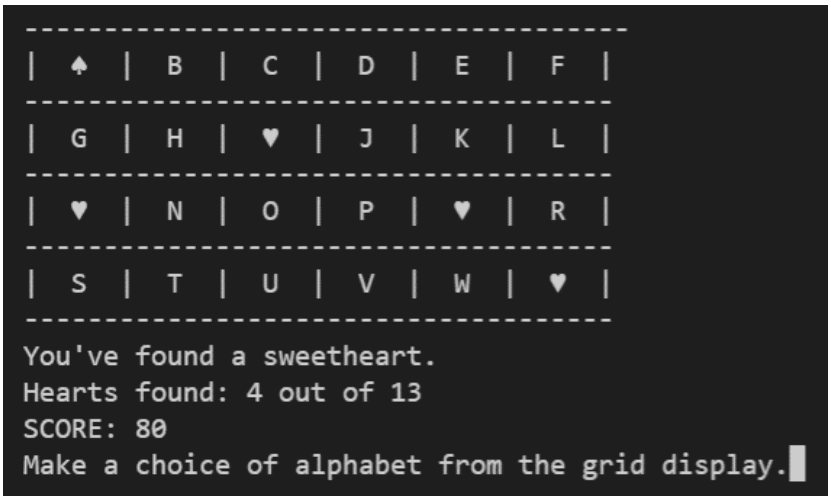
4. Difficulty Levels - A lower difficulty generates more "hearts" than a higher level. Below is a code snippet that shows a rudimentary generation of random hearts and spades. There are better ways to do this, but let's use this approach for this exercise. Note that we generate a random number between 0 and difficulty level plus 1, and if it is zero, then the grid cell is a heart otherwise, it is a spade.

```
if (Rng.Next(0, difficulty+1)==0)
{
    isHeart[i, j] = heart;
    totalHearts++;
}
else
{
    isHeart[i, j] = spade;
}
```

5. Start the game loop by doing the following:
    - Prompt the player to select a letter from the displayed alphabets.
    - If a player selects an alphabet not currently displayed, the program should prompt them to enter the alphabet again until they have entered a correct letter. To do this, search the alphabet grid to find the indices of the letter entered. If found, terminate the prompt, otherwise, ask the player to enter the letter again.

```
Select Difficulty Level [1 - 5] inclusive.2
-----------------------------------
|  A  |  B  |  C  |  D  |  E  |  F  |
-----------------------------------
|  G  |  H  |  I  |  J  |  K  |  L  |
-----------------------------------
|  M  |  N  |  O  |  P  |  Q  |  R  |
-----------------------------------
|  S  |  T  |  U  |  V  |  W  |  X  |
-----------------------------------
Hearts found: 0 out of 6
SCORE: 0
Make a choice of alphabet from the grid display.Z
Make a choice of alphabet from the grid display.Y
Make a choice of alphabet from the grid display.
```

- When the player's choice of the alphabet is found, replace the letter store at the found position of the alphabet array with the Unicode character of spade or hearts stored at the same location in the "isHeart" array.
- Then display the updated alphabet array grid together with the player's score.



6. Once the player has found all the buried hearts, the score is calculated as the number of hearts found divided by the number of alphabets selected by the user, then multiplied by 100. To do this, you must keep track of the number of hearts found, the total number of alphabets selected. In the above, there are four hearts and a spade:
    - Number of hearts = 4
    - Total number of alphabets = 4 + 1 = 5
    - SCORE = (4/5) * 100 = 80
7. The program writes the player game statistics to a file. It includes the score, difficulty level, total number of hearts generated for the game, and total number of selections made before finding all "hearts".
8. The program prompts the player if they want to play again. The game should continue if the player answers yes (Y) at this prompt and terminate otherwise.

Summary

In this lab you have had a swift introduction to the foreach loop, for iterating over collections, and declaring, accessing and modifying multidimensional arrays. Additionally, you have completed a more significant project from start to finish. If you are feeling confident, there is a bonus challenge below!

Extra Problem – The Connect Four

It should be stressed that developing a solution like in the previous example is not normal. Normally development is a slower process of implementing functionality and testing as you go.

Now, for a challenge can you make a connect4 game based upon what you have learnt?

In the solution explorer, navigate to “L7.6-ConnectFour” and open the .cs file.

Connect4 is a game for 2 players where players take turns to add their own coloured counters to columns of a grid. Counters fall to the lowest empty space in the column. The winner is the first player to create a straight line (row, column or diagonal) of at least 4 of their colour counters.

This is a challenging problem, but you should be able to attempt to make a start. Think about the list of things to do. Try to break the problem down into steps.

At various points through development, whenever the code has some piece of functionality implemented test your code, and once you are satisfied that it works as intended push the changes to GitHub with an appropriate log message.