Programming Languages Theory to Practice

25/11/2020 | 24678805@edgehill.ac.uk

Coursework one

Callum McLaughlin

CW1

Contents

[**Portfolio 1** 4](#_Toc57246314)

[**Exercise 1** 4](#_Toc57246315)

[**Reflection** 4](#_Toc57246316)

[Fig. 1 4](#_Toc57246317)

[Fig. 1.1 4](#_Toc57246318)

[**Exercise 2** 4](#_Toc57246319)

[**Reflection** 5](#_Toc57246320)

[Fig. 2 5](#_Toc57246321)

[Fig. 2.1 5](#_Toc57246322)

[**Exercise 3** 5](#_Toc57246323)

[**Reflection** 6](#_Toc57246324)

[Fig. 3 6](#_Toc57246325)

[Fig. 3.1 6](#_Toc57246326)

[**Exercise 4** 6](#_Toc57246327)

[**Reflection** 7](#_Toc57246328)

[Fig. 4 7](#_Toc57246329)

[Fig. 4.1 7](#_Toc57246330)

[**Exercise 5** 8](#_Toc57246331)

[**Reflection** 8](#_Toc57246332)

[Fig. 5 8](#_Toc57246333)

[Fig. 5.1 8](#_Toc57246334)

[**Portfolio 2** 9](#_Toc57246335)

[**Exercise 1** 9](#_Toc57246336)

[**Reflection** 9](#_Toc57246337)

[Fig. 6 9](#_Toc57246338)

[Fig. 6.1 9](#_Toc57246339)

[**Exercise 2** 9](#_Toc57246340)

[**Reflection** 10](#_Toc57246341)

[Fig. 7 10](#_Toc57246342)

[Fig. 7.1 10](#_Toc57246343)

[**Exercise 3** 10](#_Toc57246344)

[**Reflection** 11](#_Toc57246345)

[Fig. 8 11](#_Toc57246346)

[Fig. 8.1 11](#_Toc57246347)

[**Exercise 4** 11](#_Toc57246348)

[**Reflection** 11](#_Toc57246349)

[Fig. 9 12](#_Toc57246350)

[Fig. 9.1 12](#_Toc57246351)

[**Exercise 5** 12](#_Toc57246352)

[**Reflection** 13](#_Toc57246353)

[Fig. 10. 13](#_Toc57246354)

[Fig. 10.1 13](#_Toc57246355)

[Fig. 10.2 14](#_Toc57246356)

[**Exercise 6** 14](#_Toc57246357)

[**Reflection** 14](#_Toc57246358)

[Fig. 11 15](#_Toc57246359)

[Fig. 11.1 16](#_Toc57246360)

[**Portfolio 3** 17](#_Toc57246361)

[**Exercise 1** 17](#_Toc57246362)

[**Reflection** 17](#_Toc57246363)

[Fig. 12 17](#_Toc57246364)

[**Exercise 2** 18](#_Toc57246365)

[**Reflection** 18](#_Toc57246366)

[Fig. 13 18](#_Toc57246367)

[Fig. 13.1 19](#_Toc57246368)

[**Exercise 3** 19](#_Toc57246369)

[**Reflection** 19](#_Toc57246370)

[Fig. 14 20](#_Toc57246371)

[Fig. 14.1 21](#_Toc57246372)

[**Exercise 4** 22](#_Toc57246373)

[**Reflection** 22](#_Toc57246374)

[Fig. 15 22](#_Toc57246375)

[Fig. 15.1 23](#_Toc57246376)

[Exercise 5 24](#_Toc57246377)

[**Reflection** 24](#_Toc57246378)

[Fig. 16 24](#_Toc57246379)

[Exercise 6 25](#_Toc57246380)

[**Reflection** 25](#_Toc57246381)

[Fig. 17 25](#_Toc57246382)

[Fig. 17.1 26](#_Toc57246383)

[Bibliography 27](#_Toc57246384)

[How to find the difference between two integers C# 27](#_Toc57246385)

[Formatting a Float to 2 Decimal Places 27](#_Toc57246386)

[General Extra Reading Materials 27](#_Toc57246387)

# **Portfolio 1**

## **Exercise 1**

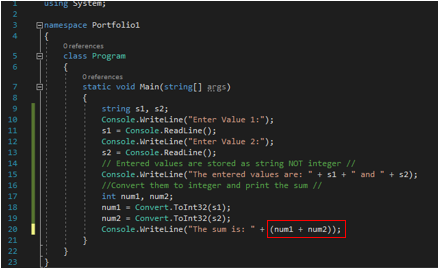
The first exercise asked me to modify the given code by removing the parentheses around “i1+i2” and explain the reason for the given output.

When the parenthesis around “num1 + num2” are removed (Fig.1), instead of adding the values of num1 and num2 together to create a sum total, each value is printed instead. E.g. If the values entered are 4 & 7 with an expected total of 11, the programme will print “47” instead as shown below in Fig.2.

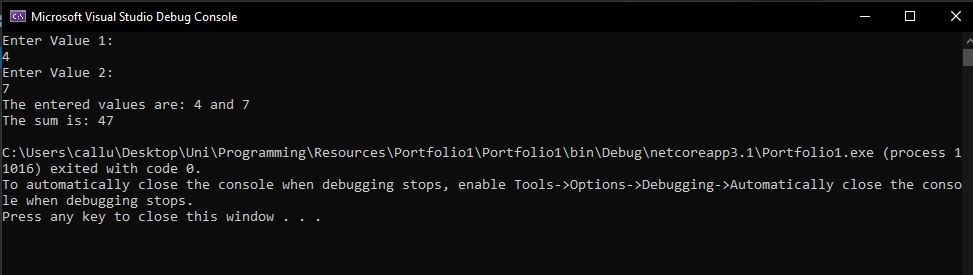
## **Reflection**

This was a simple programme that I was able to complete without any errors or problems. I had to do some quick research on converting the strings to integers as I had previously used this method before but could not remember the syntax.

### Fig. 1



### Fig. 1.1



## **Exercise 2**

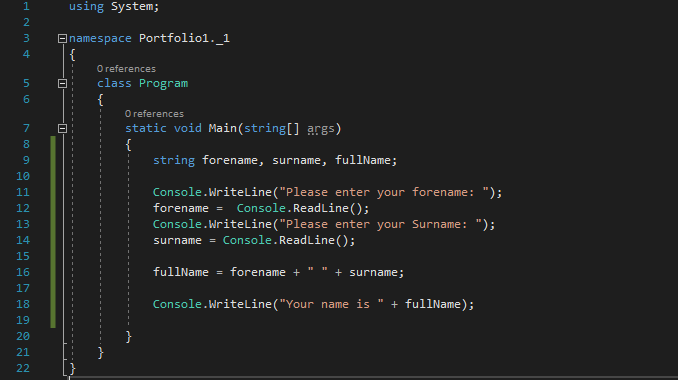
The next task was to create a programme that would allow a user to enter their forename and surname and subsequently print out both names on a single line. The user should be prompted to enter each piece of data separately.

This task was straightforward, first the user was asked to enter their forename which gets stored in a string variable called “forename”, the same is done with their surname, stored in a “surname” string variable. After both the forename and surname are entered and stored, the “fullName” variable is created by appending both names together with a space in between. The fullName variable is then printed on a new line after the text “Your name is: “, as seen in Fig 2.1.

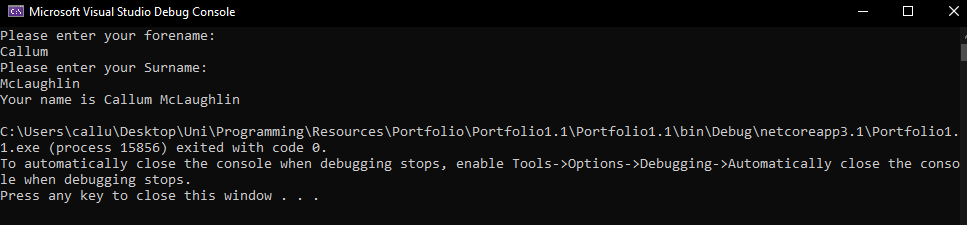
## **Reflection**

The next programme was also quite simple to complete, all I had to do was take two strings and put them together, I could have concatenated the two strings and made it into one string but I decided to go with a simpler programme and created a full name variable and simple equate that to be the forename and surname together.

### Fig. 2



### Fig. 2.1



## **Exercise 3**

The third exercise asked that the previous programme be extended for printing the full name of more than one person. Firstly, the number of people whose names were to be added was entered and then each forename and surname should be entered individually with all names shown on a single line per Full Name at the end.

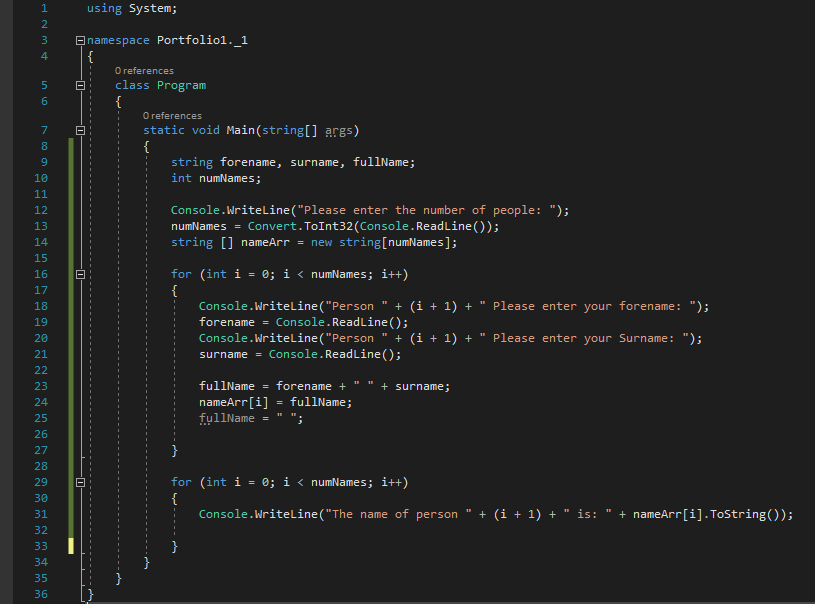
This was achieved by creating an array to store the Full Names of each person entered. The size of the array was determined by the number of people entered at the start. If a user entered that there were to be 5 peoples full names entered, that would set the array size to 5. As shown on line 14 of Fig. 3.

Next, a for loop was used and ran to the number of people entered, so if 5 was entered, the loop would run five times to allow 5 peoples forenames and surnames to be entered and stored into the array. After the for loop had run its course, the array would then be printed out using another for loop, running to the array length and printing each name on a new line with the position/number of each person entered beside their name, as seen in Fig. 3.1.

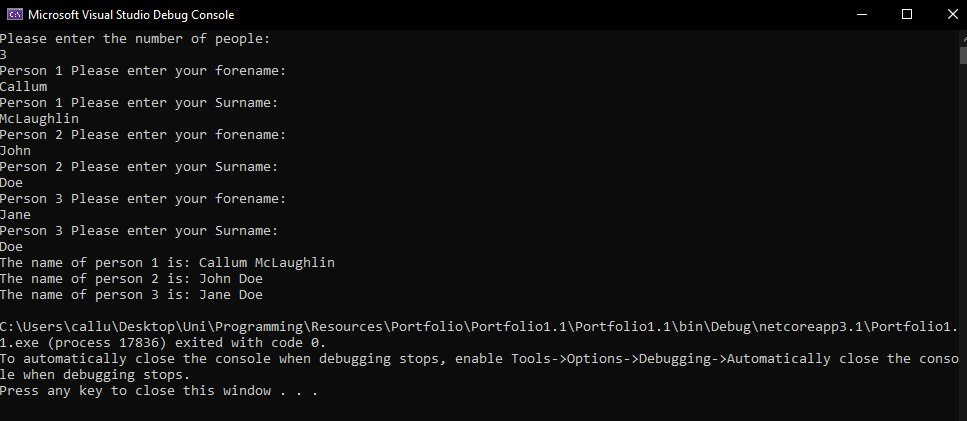
## **Reflection**

It was easy to extend the second programme to allow for multiple people to be entered. I simply asked the user to enter the number of people they wanted to enter and then created a dynamic array using that number. I did not encounter any problems with creating this programme.

### Fig. 3



### Fig. 3.1



## **Exercise 4**

Exercise 4 gave the task of creating a programme that calculates the area and perimeter or a circle after the user enters the radius of the circle. A constant for π was to be used (3.142). The task was completed by asking the user to enter the circle radius, which would then be stored in a variable “circRadius”.

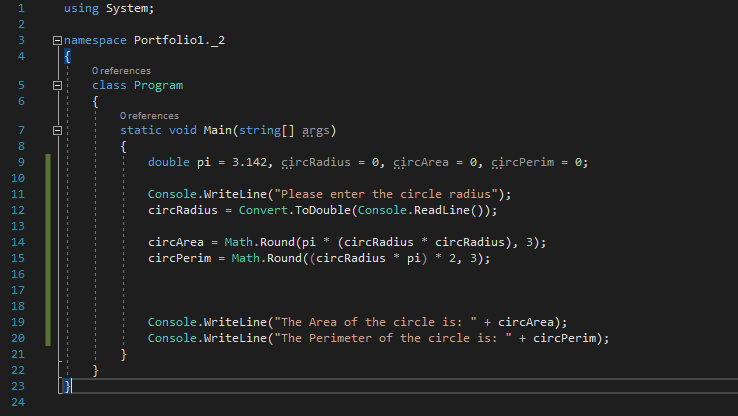
Then using the formula for calculating the area of a circle (A=πr2) and the formula for calculating the perimeter/circumference (C=2πr), the programme was able to calculate the area and circumference of the circle using only the radius entered by the user.

After they had both been calculated, the area and perimeter of the circle were displayed separately on new lines, as shown in Fig 4.1. Although the area and perimeter are shown to 3 decimal places, this was not the case after the first attempt. In order to get the number to focus to only 3 decimal places, Math.Round was used around the calculation to force the answer to be displayed to 3 decimal places.

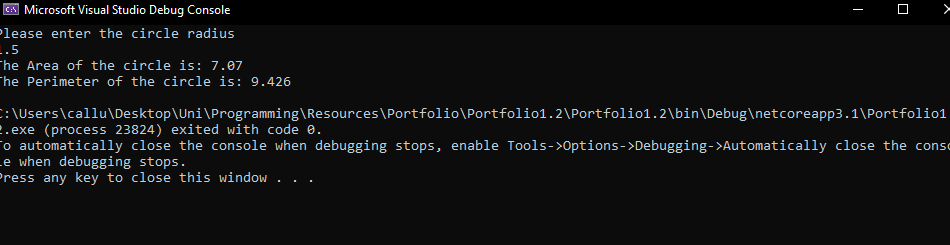
## **Reflection**

I had to do some research to find the equations for finding the area of a circle and perimeter of a circle as I could not remember them off the top of my head. I also needed to figure out how I could round up the number when the calculation was done. I found the Math.Round method could be used to do this, and I implemented it into my programme.

### Fig. 4



### Fig. 4.1



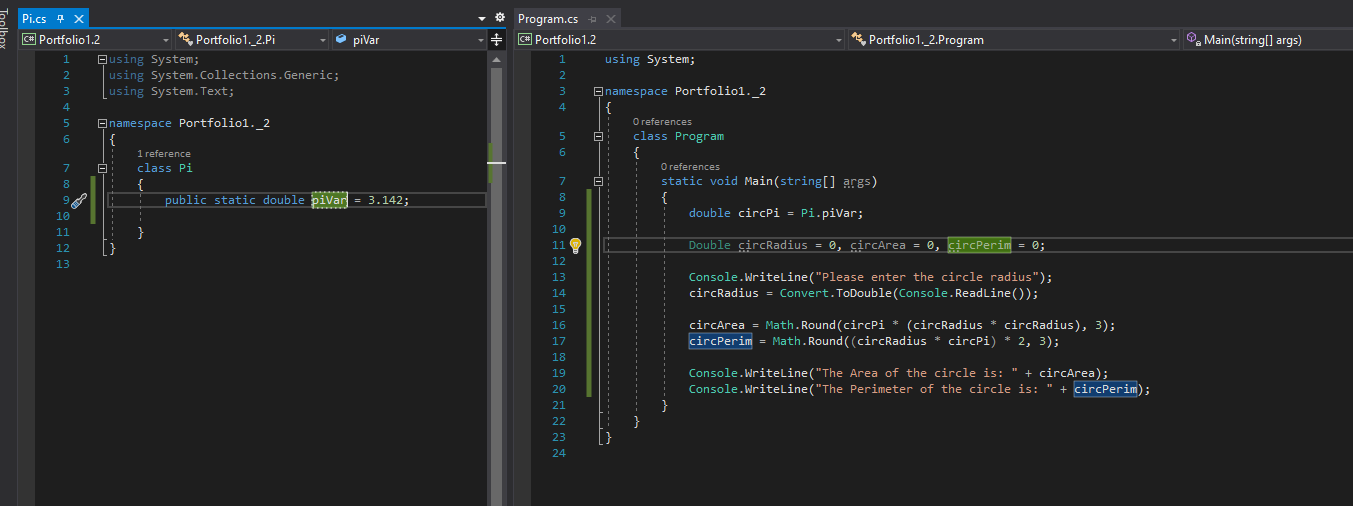
## **Exercise 5**

The next exercise was quite straightforward and simply asked that a separate class be defined for π. A new class called Pi.cs was created and the constant for π was stored there as a variable (piVar) which was then accessed from the Program.cs by creating another pi variable called circPi and setting it equal to the value of the constant piVar in the Pi.cs class. This can be seen in line 9 of Fig. 5.

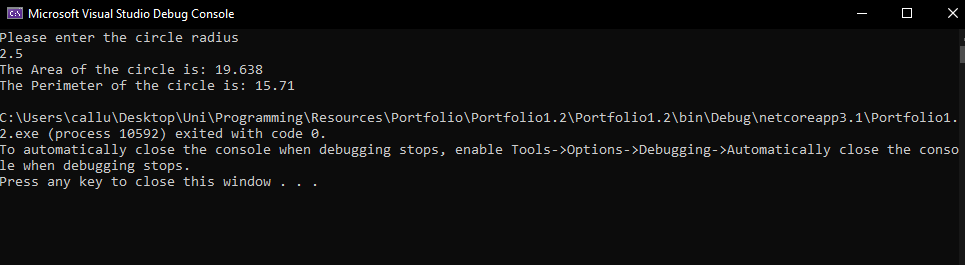
## **Reflection**

The next exercise was the shortest of the lot and I was able to complete quickly using the notes from black board. I done some extra research on classes as well and how public and private variables differ and how they can be accessed.

### Fig. 5



### Fig. 5.1



# **Portfolio 2**

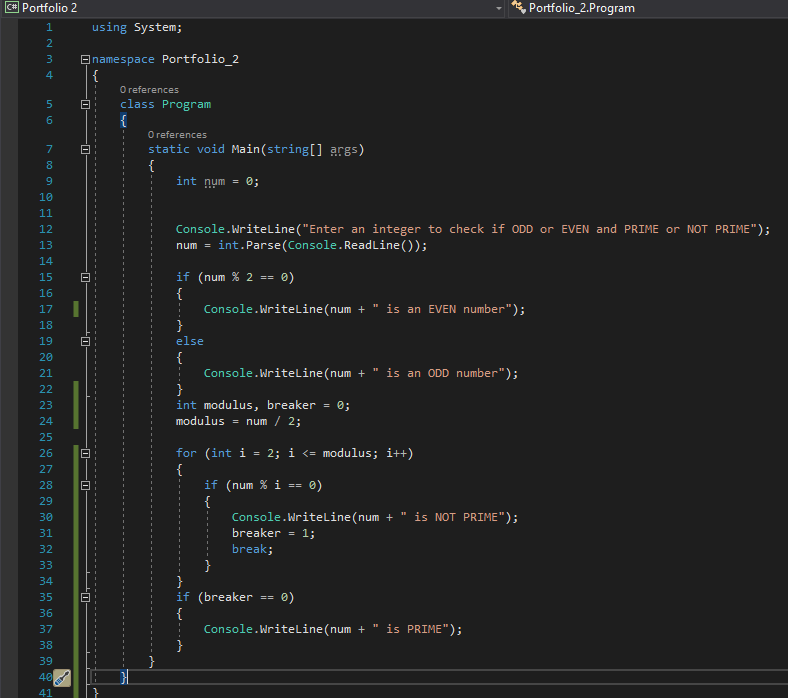
## **Exercise 1**

Exercise one asked that a programme be created that reads a number from the console and checks if they number entered is ODD or EVEN. The way this was done was by asking for a number to be entered by the user and using modulus, it could be determined whether or not the number was odd or even. If there was a remainder after using modulus, the number was odd, if not it was an even number. This can be seen in the code below.

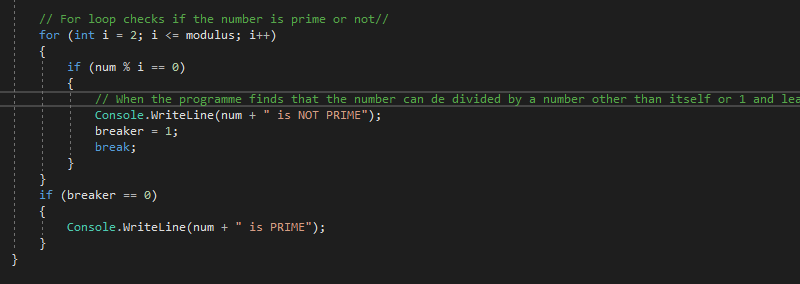
## **Reflection**

This programme only required that the entered number be divided by 2 and if there was a remainder, you would know the number is even and if there is no remainder its even. I had to do some quick research to figure out how to use modulus in the most efficient way and I decided on an if statement would be the best way to approach it.

### Fig. 6



### Fig. 6.1



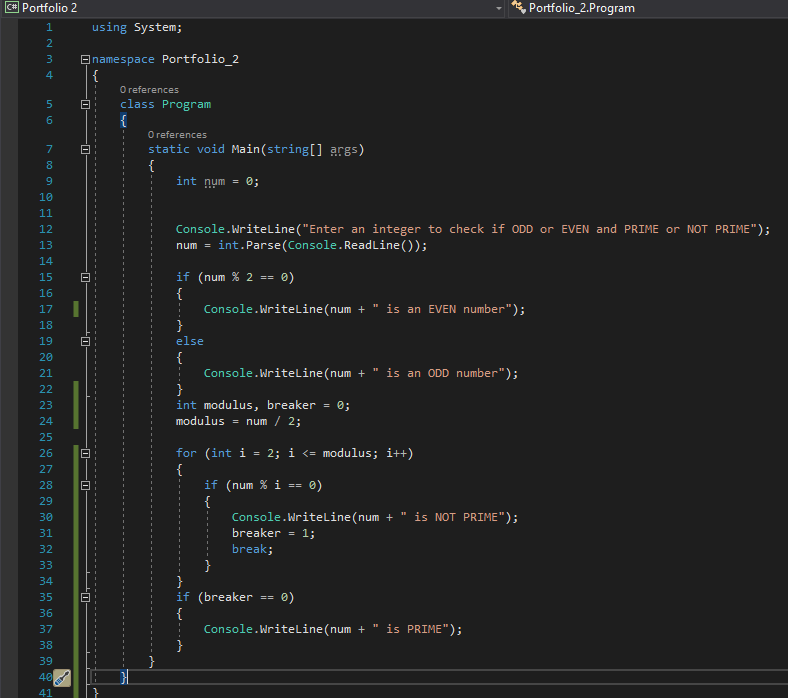
## **Exercise 2**

The second exercise asked that the first programme be extended to determine whether or not the entered integer was a prime number or not. By checking whether or not the entered number could only be divided by itself or 1 using a for loop which ran to half of the number entered, the programme can determine whether or not the entered integer is a prime number. This can be seen in the code below starting from line 23.

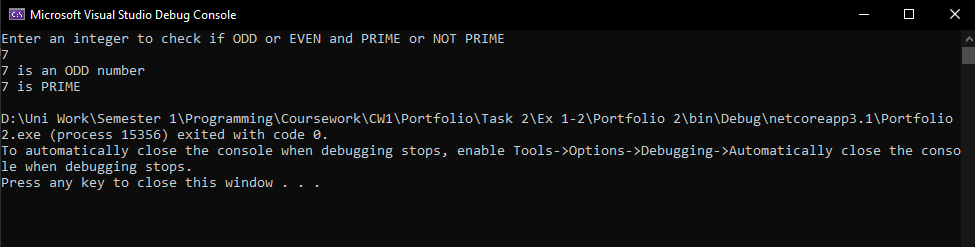
## **Reflection**

This exercise required some extra research in order to work out how I was going to determine whether or not the number was a prime number or not. There were a few ways I could have done this, but I decided the best way to do this would be to use a for loop and start by using modulus on the number and checking for a remainder.

### Fig. 7



### Fig. 7.1



## **Exercise 3**

The next exercise set the task of writing a programme that reads an odd number from the console and displays an x shape of asterisks in the console. The max number of lines of asterisks from top to bottom in the x shape is the entered odd number.

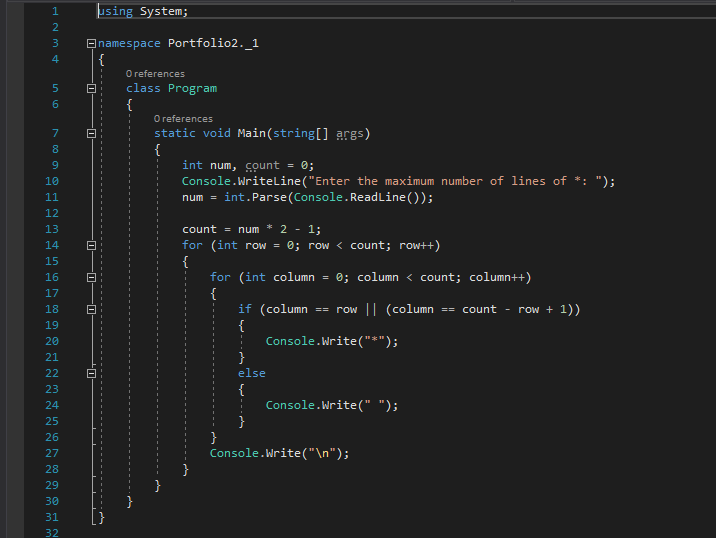
The logic behind the programme is the pattern consists of N\*2-1 rows and columns. So an outer loop is run to iterate through rows and an inner loop is used to iterate through columns. There are two cases that stars are printed, otherwise a space is printed.

This one I found quite tricky and did not actually get it to work properly. I was able to create an x shape of stars from the number entered but I always have an extra star at the top left-hand corner of the X shape as shown below.

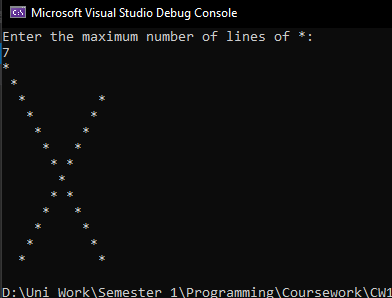
## **Reflection**

The next exercise I was unable to complete fully as the “x” I made doesn’t actually form perfectly, its crooked at the top and I haven’t been able to fix it or figure out why its not a perfect x. I left it as it is as I believe I have still understood the task and produced something that fits what was asked.

### Fig. 8



### Fig. 8.1



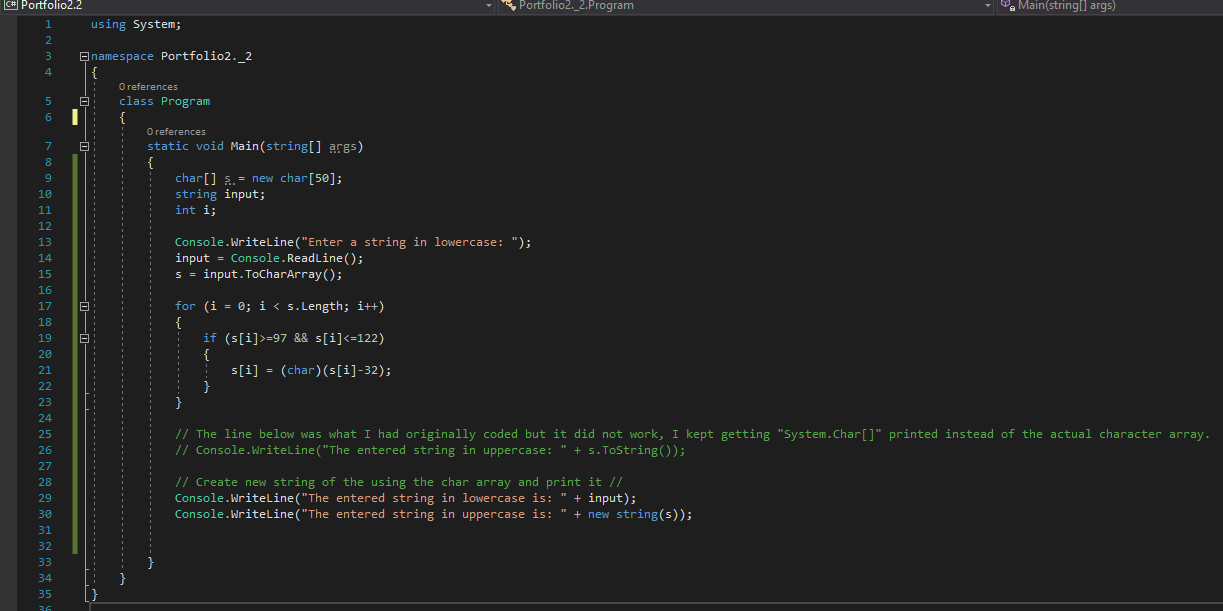
## **Exercise 4**

The next task was to write a programme that reads in a sentence from the console and display the sentence from lower to upper case without using any library functions. This took some research until a solution was found. The string could be stored in a character array then using the ascii values of the characters, the characters could be changed from lowercase to uppercase by subtracting 32 from the ascii value of each lowercase character. This is shown below.

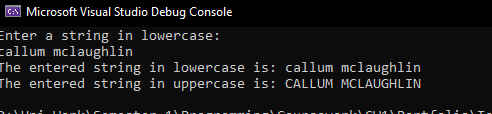
## **Reflection**

This task had a number of different ways that it could be completed. I could have used a “toUpper” method, but I decided to use something simple like the Ascii codes as you can add 32 in the ascii code of a lowercase letter and it will give you the uppercase version.

### Fig. 9



### Fig. 9.1



## **Exercise 5**

This exercise asked that a programme be created to check whether or not a given input string is a palindrome or not. The string is to be read from the console and provide a yes/no decision as output.

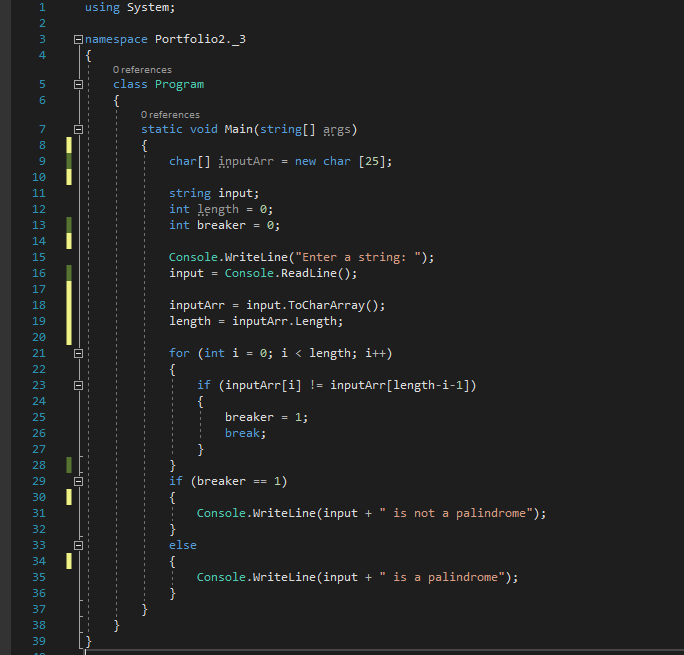
The solution is achieved by creating an array of characters which will be filled with the string that the user inputs, giving us a way of checking each individual character against the others. When the user enters a string, it is stored in a variable called “input” which is then used to populate the character array. A for loop can then be run to the length of the string that was input by assigning a “length” variable equal to the length of the char array (string input by the user).

The for loop completes the necessary checks to evaluate whether or not the word is a palindrome. E.g. a five-letter word, RADAR, will be checked by comparing the first and last characters and checking if they are the same, then the 2nd and the 4th and so on until all characters are checked. Middle characters are compared to themselves as it will always be in the same position when the word is read both ways. If all character’s match with their respective characters when checked, the word is a palindrome, if one does not, the breaker value is set to one and the word is not a palindrome.

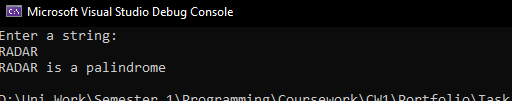
## **Reflection**

This exercise was difficult to wrap my head around although when I took the time to read what was asked of me and played around with a few different ways of doing the task I decided the easiest way for me to complete the task was to put the word into a character array and go through the character array using a for lop to check if the first and last letter are the same then the second and 2nd last and so on. If all of the letters match their equivalent, then the word is a palindrome.

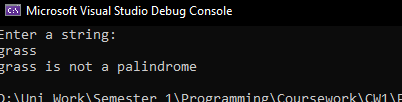
### Fig. 10.



### Fig. 10.1



### Fig. 10.2



## **Exercise 6**

The sixth exercise gave the task of creating a programme which could generate a random number between 0 and 100 and allow the user to guess the number up to a maximum of 5 times, each time giving them a hint as to how close they are to the correct number. If they guess the correct number, a message should be displayed on screen informing the user.

Firstly, a random number must be generated and stored in a variable, we will call this variable “rNum”. Next, the user must be prompted to make their first guess. When the user enters a number, it is stored in another variable which will be called “userInput”.

When the user input is stored, we can then check if the guess is equal to the random number with an if statement, if it is, output the message that you user guessed correctly, if not, continue down the list of if statements and output the appropriate message depending on the difference between the users guess and the random number.

For example, if the random number is 3 and the users guess is 95. The first if statement will be triggered as the difference between the guess and the random number is greater than 50. A message would also be displayed saying “Your guess was very high” as the guess was 50 or more greater than the random number.

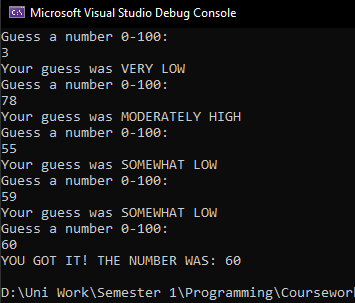
The hints continue giving the user varying accuracy of how close they are to guessing correctly. If the user does not guess within 5 tries, the programme displays a message showing the random number.

For this exercise I needed to do some research to figure out how to generate a random number and assign it to a variable, I have done this before in java and was unsure of the syntax for C#. The information that I used is noted in the appendices, Appendix 1.

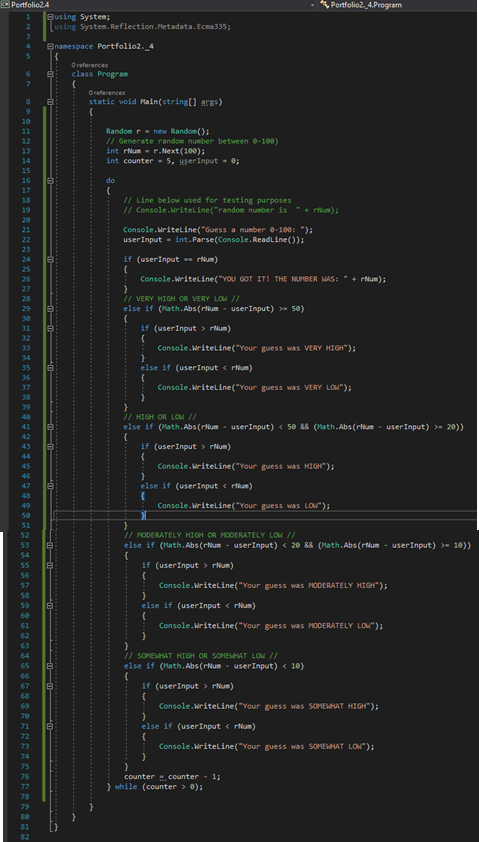
## **Reflection**

The next programme was quite a long one to start from scratch, but I was able to complete this one quite easily and could not see a simpler way of doing if other than some if statements. There are probably other ways I could have completed the task, but I thought this was the simplest and I did not bother to look for more ways as I knew this way would work.

### Fig. 11



### Fig. 11.1



# **Portfolio 3**

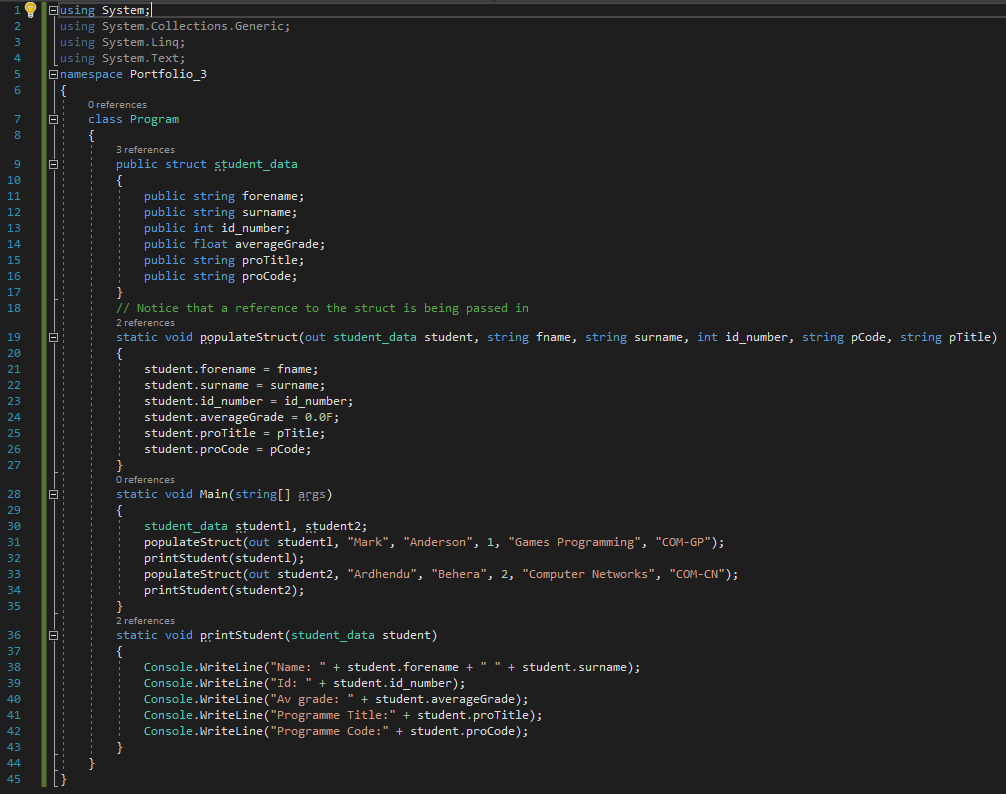
## **Exercise 1**

This exercise gave the task of amending the “student\_data” structure in the given piece of code so that a student also has a programme title and programme code. Both fields must be strings. In order to give each student a programme title and programme code, they must be declared in the data structure then passed into the constructor and set with a default value. Then each student can be given a programme title and programme code in the main method and “printStudent” function.

## **Reflection**

This task took me some time to complete as I had never used structures before, I am used to using classes and using inheritance to create programmes similar to this. I reviewed the slides and lectures on “BlackBoard” relating to structures and that allowed to eventually wrap my head around the concept and complete the tasks.

### Fig. 12



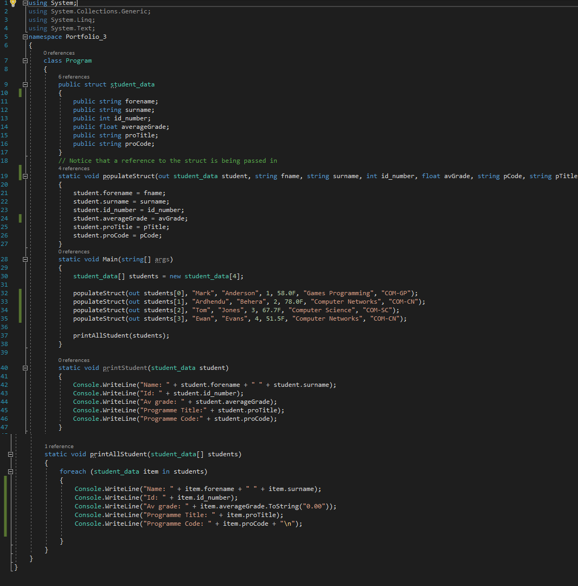
## **Exercise 2**

Exercise two set the task of amending the student\_data programme to hold an array of four students in the main method rather than a single student. In the main method, an array was declared and populated using four calls to the populateStruct method. The next part of the task was to write a new printAllStudents method which loops though the array and prints out all the elements in the array. This was done using a for each loop, the loop goes through the array for each element in the array, hence the name for each loop. It prints each element in the array as it loops through it. This also could have been done with a for loop set to the size of the array which would have worked equally as well. The output when 4 students are entered is shown in Fig. 13.1.

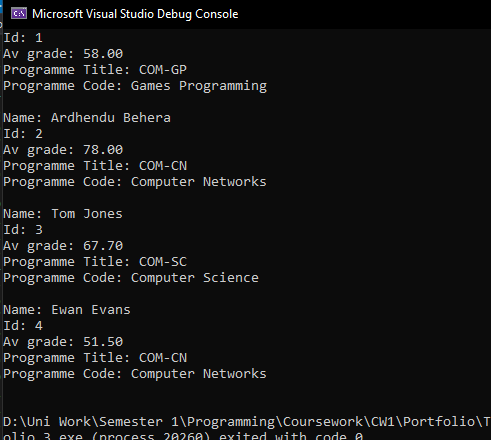
## **Reflection**

I was able to complete this task relatively easily as I have done arrays a lot in the past, especially in C#. I did some extra reading on 2-dimensional arrays and structures after I completed the task to give me a better understanding of how they can be used together.

### Fig. 13



### Fig. 13.1



## **Exercise 3**

The next exercise asked that the previous programme be extended by allowing the user to enter the number of students they want to input into the programme then an array was to be dynamically created equal to the size that the number entered by the user. Each student’s name, surname and id were to be entered via the console.

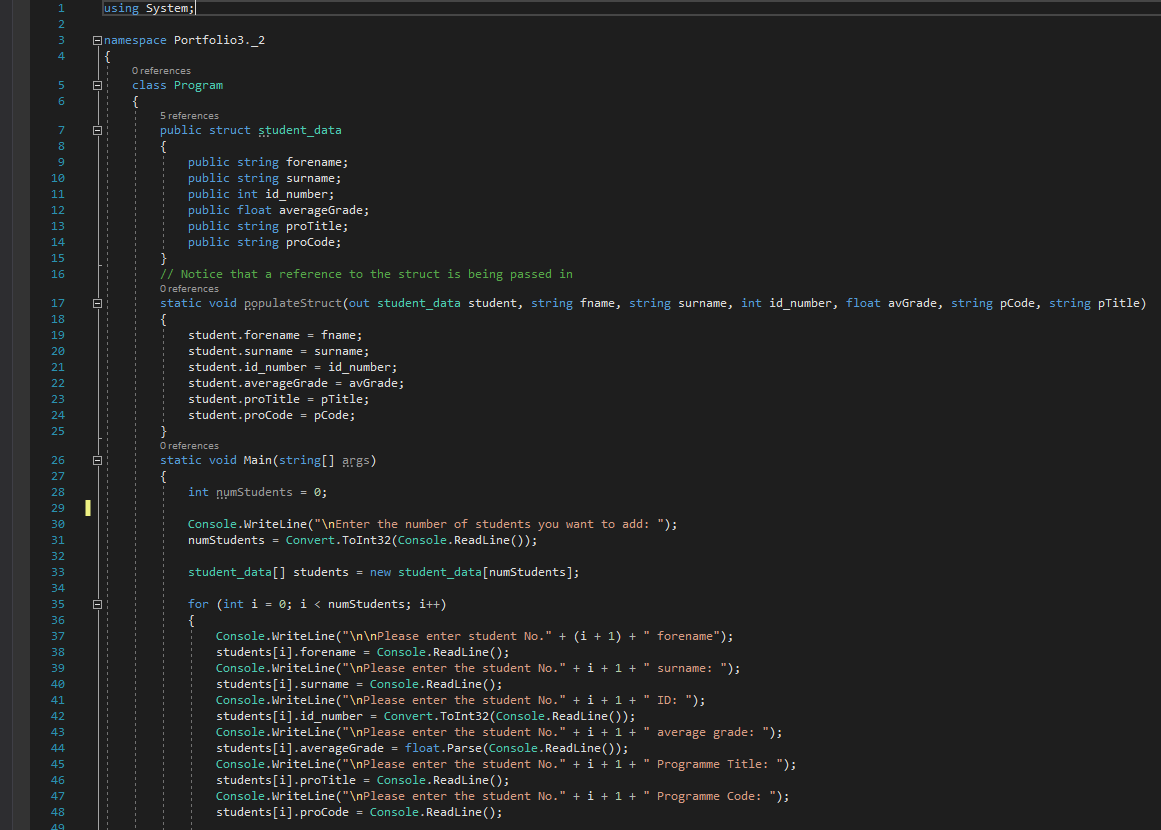
To start, the user is asked to enter the number of students they wish to enter into the programme. This number is then stored as numStudents and an array is initialised with a size of “numStudents” creating a dynamically sized array. A for loop is then run to number numStudents (or the number entered by the user e.g. 5 times). The for loop asks the user for the relevant information needed to create a student record such as their forename surname, id, grades etc. All of the entered data is stored in the two-dimensional array of student\_data (students).

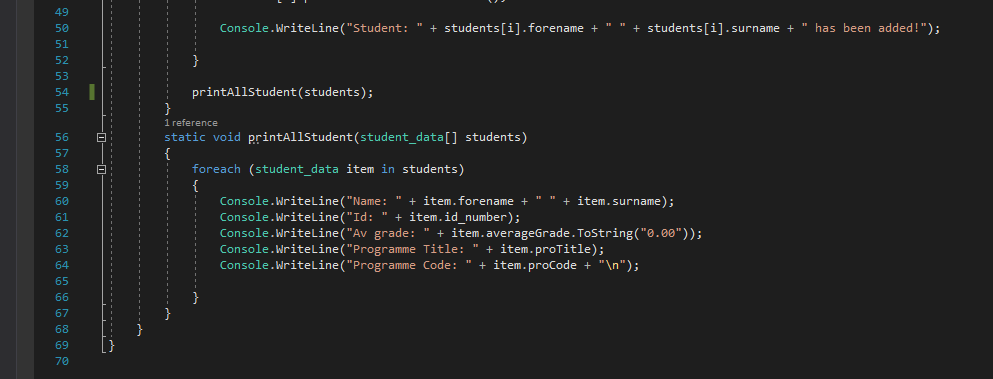
When the programme exits the for loop, the printAll Students method is called and all of the entered students records are displayed on the console as seen in Fig 14.1. In order to get the average grade to show to two decimal places, I had to do some extra research and I found a useful solution which was to “pass the format to the ToString method, e.g. myFloatVariable.ToString(“0.00”); //2dp Number”, Michael (2017), Stack Overflow Post, StackOverflow.com.

## **Reflection**

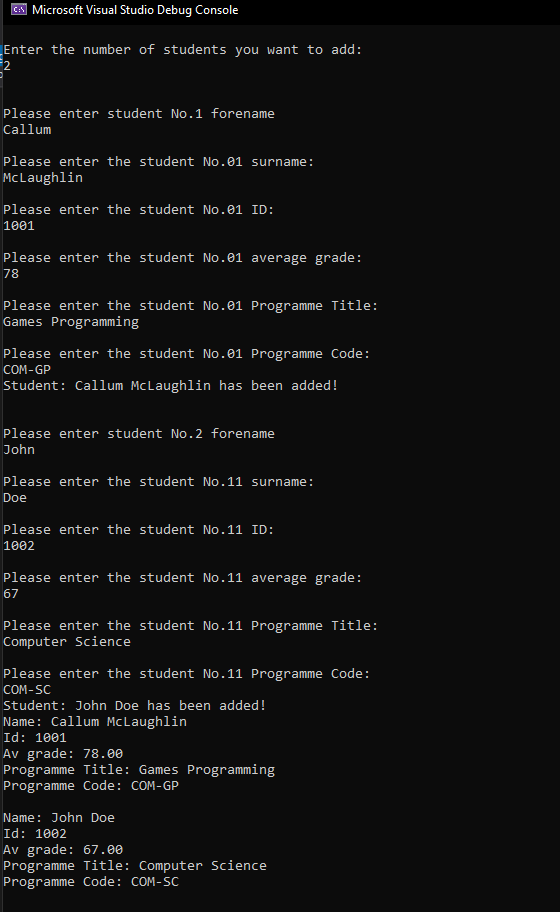
This task was easily completed as it was simple to convert the pre-filled array to a dynamic array of a size dependant on what number the user entered. I did some extra work than required and added in the programme details as well as it was only a few extra lines. I could have also used a for each loop to run through the array but I decided to use a for loop instead as it was simpler.

### Fig. 14





### Fig. 14.1



## **Exercise 4**

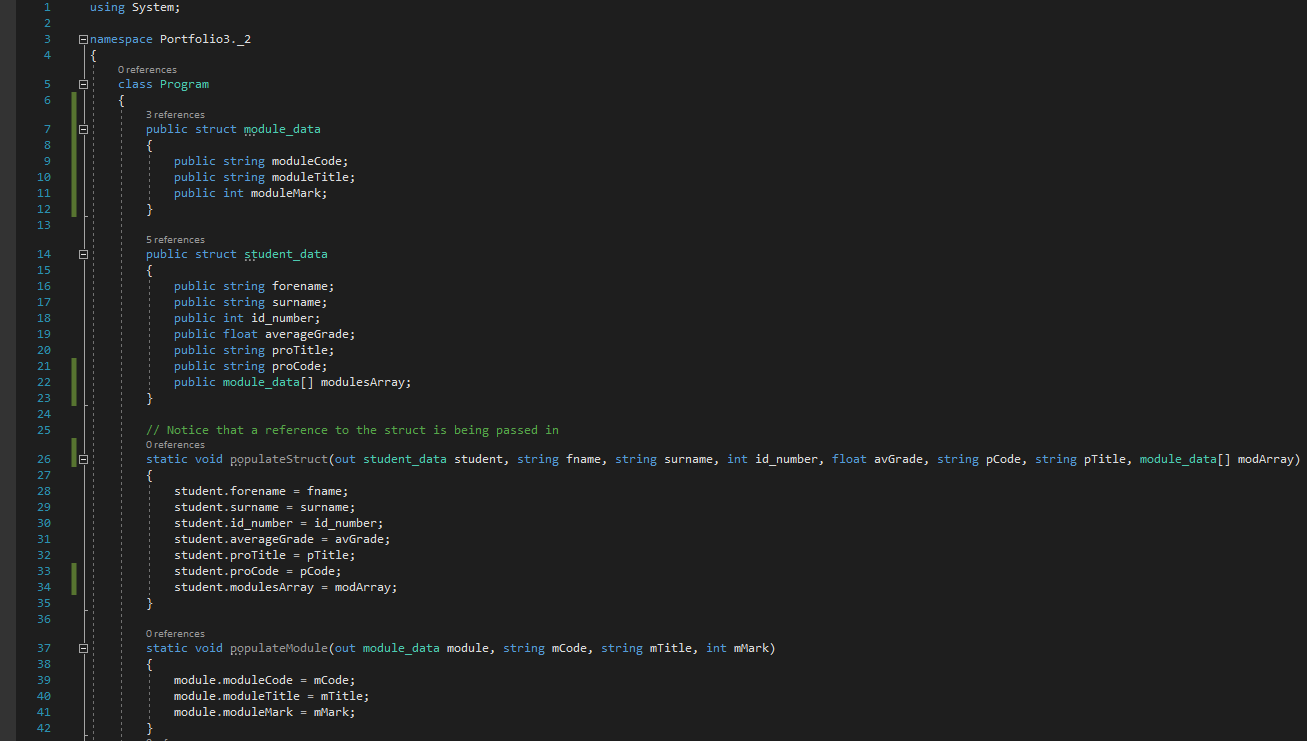
The next task was to amend the student\_data programme and add a new structure which would hold module data. In the module\_data structure, there is a module code, module title and module mark. Assuming that a student takes 6 modules, the student\_data struct was to be extended to hol six module\_data items.

Firstly, a module struct was created and moduleCode, moduleTitle and moduleMark were declared inside of it. Next an array of module\_data was declared as a variable inside the student\_data struct. Then inside the for loop which is used to take the input from the user for the student entries another for loop was created to allow the user to enter the modules that the student would be taken after their other details were entered. The loop was set to run six times as the brief specified that a student takes six modules. The loop included the relevant output messages and took the input for each of the values in the modules struct and placed them in the array of module\_data. After the six modules are entered the loop is exited and the students are printed to the console as before although the module data is not as this was not specified in the exercise/portfolio brief.

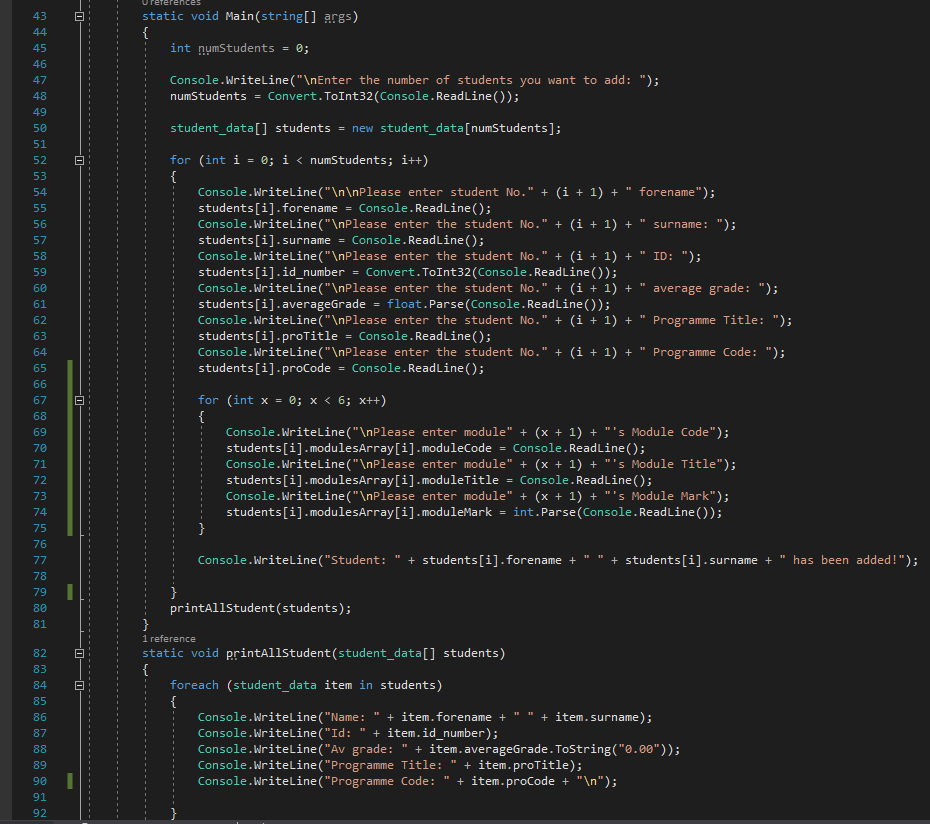
## **Reflection**

Adding the module structure was easily done but adding the array of module\_data to the student\_data struct became a problem as I was using a different array declaration, one where I would usually specify the size of the array but this wouldn’t work so I decided to not specify the size to see if that would fix it and it did.

### Fig. 15



### Fig. 15.1



## Exercise 5

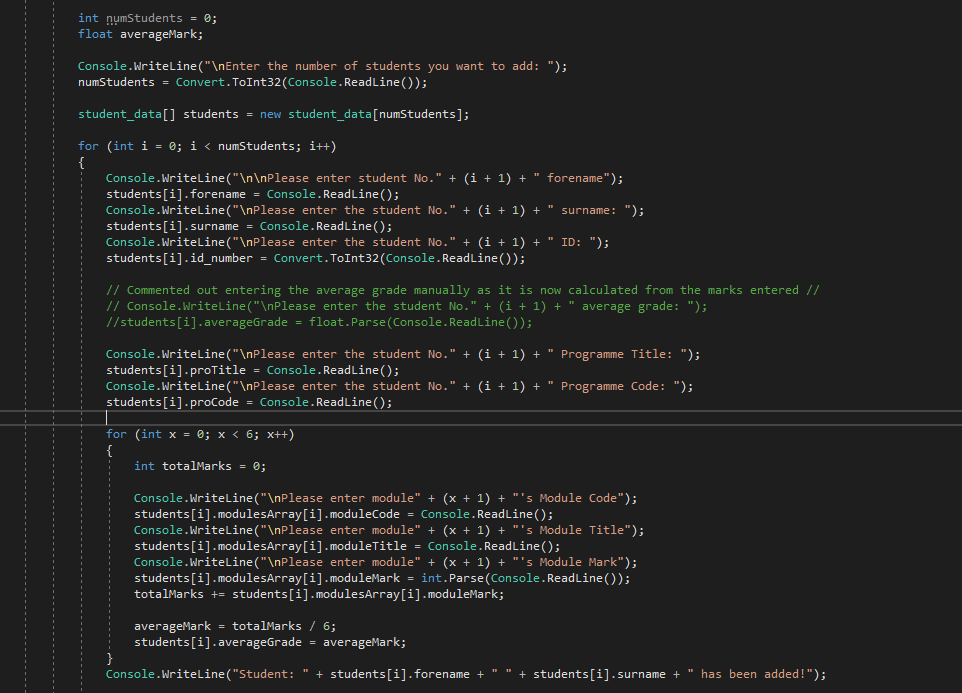
To complete this exercise I edited the for loop that is used to take the students modules and made 2 variables, one to store the total marks of their modules and one to work out their average mark after dividing it by 6. The average mark was then placed into the student’s array under the average grade attribute. This method was best as it updates each time the loop passes the line of code which sets the average mark into the array meaning if you edited the programme so that you can enter any number of modules it will always update the correct number of times because it runs to the loop.

There were problems with this implementation as when the programme is run, there is a null reference meaning that the programme is trying to access a variable or attribute which does not have a value. I have dealt with many of these errors before and they are usually easy to fix as they are logic errors more often than not but I did not want to waste time on correcting this so instead I will try and come back to it when I have completed all other work for the assignment.

## **Reflection**

This task was a simple adding and dividing exercise in order to find out the average mark for each student, again I probably could have used a for each loop to run through the module\_data array but a for loop was the easier option.

### Fig. 16



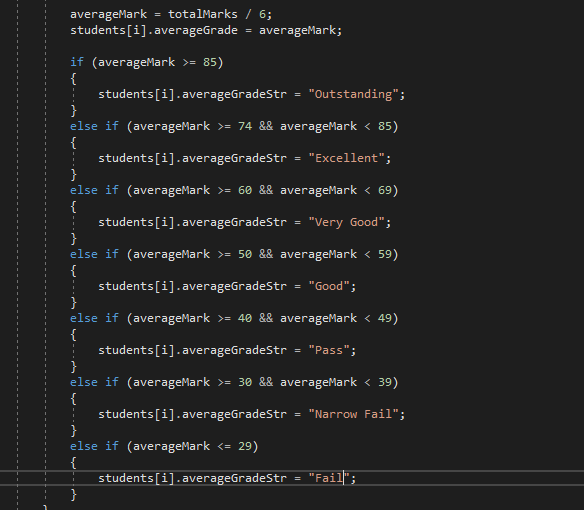
## Exercise 6

This exercise was straightforward and set the task of amending the previous student\_data programme so that the average score is assigned as a grade (string), with the grade being determined on score boundaries. All that had to be done for this was a new attribute to be added to the student\_data struct called “averageGradeStr” which would be a string to hold the grade and an if statement which would determine which grade the student would get dependiong on what score bracket they fall into. If the condition is met in the statement, the students “averageGradeStr” in the students array is set using the “pointer” from the first for loop which handles the students data being entered and uses that “I” value in the for loop to determine which index in the array to store the value.

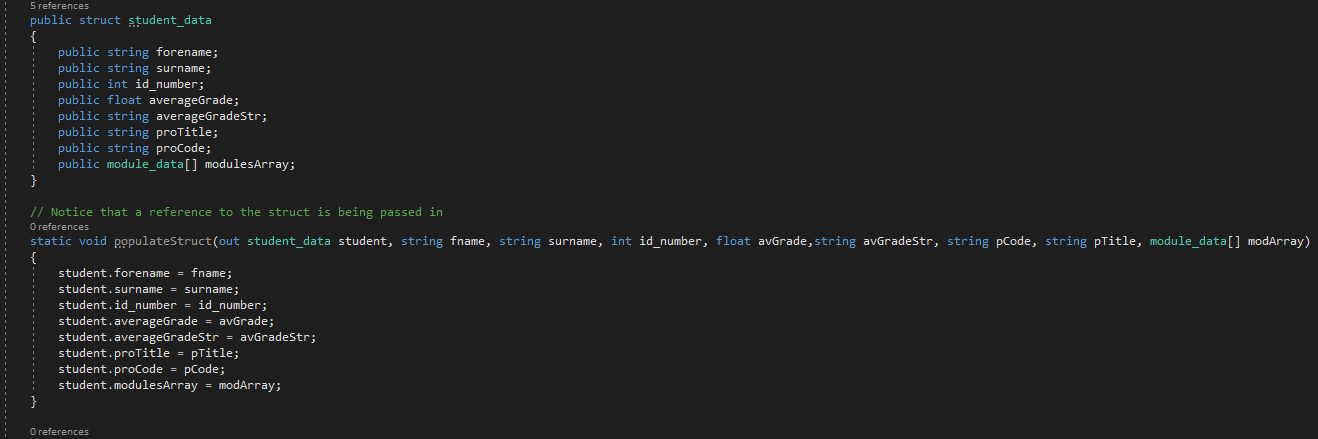
## **Reflection**

I could not think of any of way to do this exercise more simply and effectively than using if statements and starting from the highest number and working down makes for the most efficient solution. I did not have any problems with this task as I have created similar if statements before.

### Fig. 17



### Fig. 17.1



## Bibliography

### How to find the difference between two integers C#

S.L. Barth (2012) StackOverflow.com

<https://stackoverflow.com/questions/339961/difference-between-2-numbers>

### Formatting a Float to 2 Decimal Places

Michael (2017) StackOverflow.com.

<https://stackoverflow.com/questions/6356351/formatting-a-float-to-2-decimal-places>

### General Extra Reading Materials

Concepts of Programming Languages by Robert W. Sebesta – 2016

Software Engineering by Ian Sommerville – 2016

C# Notes for Professionals by Goalkicker.com - 2019