Coursework 1

Module: CIS2147 Programming Languages: Theory to Practice 2021

Module Leader and Lecturer: Ardhendu Behera

By: Christopher Diaz Montoya
Student ID: 24707686

Contents

Contents of Figures

Introduction

Within this report, breakdowns of multiple exercises completed using the C# programming language were written, each problem was completed using suitable data types and algorithms showcasing the skills alongside programming techniques learnt each week, paired with techniques researched and independently learnt. The visual code studios' IDE (Dechalert, 2021) was used across all exercises.

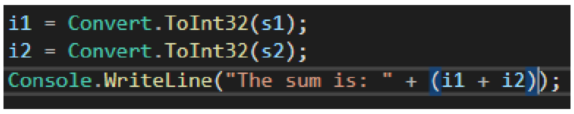
The aim was to progressively learn new techniques and to implement them into relevant sets of exercises. The purpose of this report was to showcase how these techniques were implemented, if they were successful, what was learnt along the way, and the problems which were faced in each exercise. There are four sections included in this report, each about a different set of problems worked on.

Week 1

Overview – Week 1

During Week 1, the fundamental topics for the C# programming language were introduced. Along with some basic exercises to ensure the fundamentals were understood before progressing onto the more complex weeks. Week 1 was a breakdown of the first set of exercises. Within each subsection of Week 1 an explanation to what the task was, what the expected outcome was, the research needed, the features which were incorporated, and the source code created were all presented.

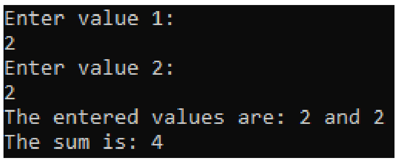
Exercise 1



```
i1 = Convert.ToInt32(s1);
i2 = Convert.ToInt32(s2);
Console.WriteLine("The sum is: " + (i1 + i2));
```

Figure 1 - Code with parenthesis

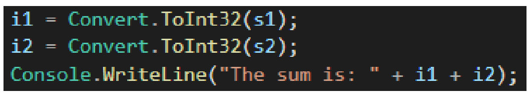
The aim of this exercise was to remove two brackets from some given code and to explain why the output was what it was. The code took in two inputs which were strings (Microsoft, 2021), converted them into integers and finally printed out the sum of the two integers. The expected outcome for this was the sum of the two integers, both before and after the parenthesis were removed.



```
Enter value 1:
2
Enter value 2:
2
The entered values are: 2 and 2
The sum is: 4
```

Figure 3 - Parenthesis output

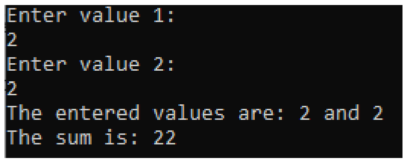
The code with the parenthesis can be seen in Figure 1 - Code with parenthesis. With the output shown in Figure 3 - Parenthesis output**.** As seen the output was as expected. When the parentheses were removed as shown in Figure 2 - Code without parenthesis the output was still expected to behave the same as the variables. As shown in Figure 4 - Without parenthesis output, this was not the case. The parenthesis help read the code in a specific way, the plus signs in Figure 4 - Without parenthesis output were strictly to print out multiple variables on one line (Microsoft, 2021). If the two variables are of different types, the plus sign printed out the two variables side by side, if they are of both numerical type then it would be a math operator (Jones and Freeman, 2003). As it was the first case then the output worked as follows,



```
i1 = Convert.ToInt32(s1);
i2 = Convert.ToInt32(s2);
Console.WriteLine("The sum is: " + i1 + i2);
```

Figure 2 - Code without parenthesis

2+2=22 instead of the sum of two numbers 2+2=4. It just prints what is stored in the variable, the parenthesis helped force precedence (Microsoft, 2021).



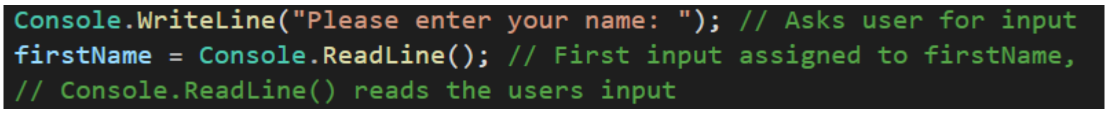
```
Enter value 1:
2
Enter value 2:
2
The entered values are: 2 and 2
The sum is: 22
```

Figure 4 - Without parenthesis output

Exercise 2

This exercise was about being able to get the user to input their name and surname separately, with the output as their full name. The exercise was broken down into as many steps as possible.

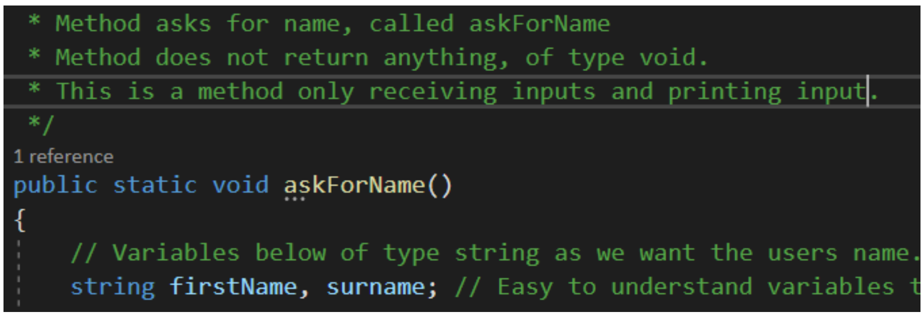
- How the user can add input
- How to read and store the users input
- How to print this out.



```
Console.WriteLine("Please enter your name: "); // Asks user for input
firstName = Console.ReadLine(); // First input assigned to firstName,
// Console.ReadLine() reads the users input
```

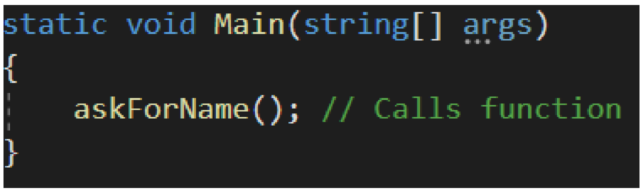
Figure 5 - Reading and storing user input

The expected outcome was two lines asking for the user's forename and surname then printing them both out on a single line.



```
 * Method asks for name, called askForName
 * Method does not return anything, of type void.
 * This is a method only receiving inputs and printing input.
 */
1 reference
public static void askForName()
{
    // Variables below of type string as we want the users name.
    string firstName, surname; // Easy to understand variables t
```

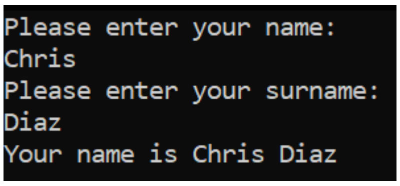
Figure 6 - Method for input



```
static void Main(string[] args)
{
    askForName(); // Calls function
}
```

Figure 7 - Input method call

As the first step was to see how the user can add input, the same structure was used as in Week – 1 Exercise 1, this can be seen in Figure 5 - Reading and storing user input. This was researched and confidently implemented, as Krossing (2019) confirmed this, it was certain that Console.WriteLine was to print what was need as the output along with asking a user for input. Console.ReadLine was to read whatever input the user added to the question printed (Hejsberg, 2009). As shown the variable firstName of type string (Microsoft, 2021) stores whatever the user inputter, this varible name was used to easily identify the variable. The final problem was to print out both the first name and surname, as they were stored in easy-to-understand variables, these were both printed out on the same line side by side. This was all inside a method called askForName, this was learnt last academic year and was taught that it is good to add methods were possible especially if more was going to be later added. This was recapped and although not needed it was confidently implemented thanks to w3schools (n/a) and Jones and Freeman (2003), it was explained that methods worked in the same way as Java methods. The output can be seen in Figure 8 - Exercise 2 output, this was the expected output, and no problems arose.

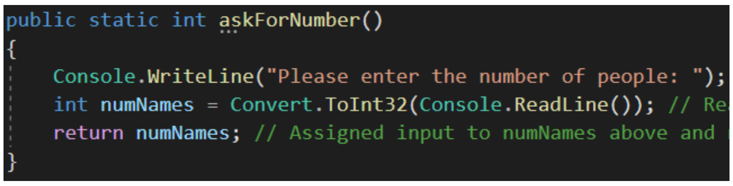


```
Please enter your name:
Chris
Please enter your surname:
Diaz
Your name is Chris Diaz
```

Figure 8 - Exercise 2 output

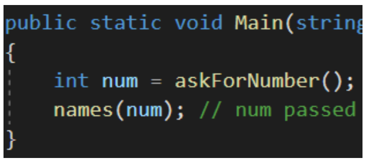
Exercise 3

This exercise was an extension of exercise 2. This exercise asked that the user also inputted the number of people they would like to have their name inputted. Again, the exercise was broken down into as many steps as possible; step 1, ask user for the number of names to be inputted and store the integer; step 2, add a loop which runs the number of times as the integer inputted; step 3, ask for name and surname and print the whole name for each loop. The expected outcome was for the user to be asked how many people, then asked for the first name of the first person, then the second name of the first person, finally the whole name of the first person would be printed, after this, repeat with the second person and third until the number the user inputted was reached.



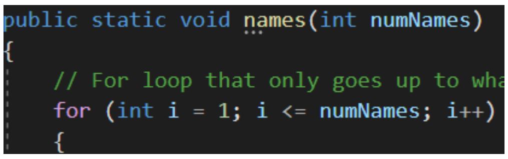
```
public static int askForNumber()
{
    Console.WriteLine("Please enter the number of people: ");
    int numNames = Convert.ToInt32(Console.ReadLine()); // Re
    return numNames; // Assigned input to numNames above and
}
```

Figure 9 - Ask for number method



```
public static void Main(string
{
    int num = askForNumber();
    names(num); // num passed
}
```

Figure 10 - Method calls

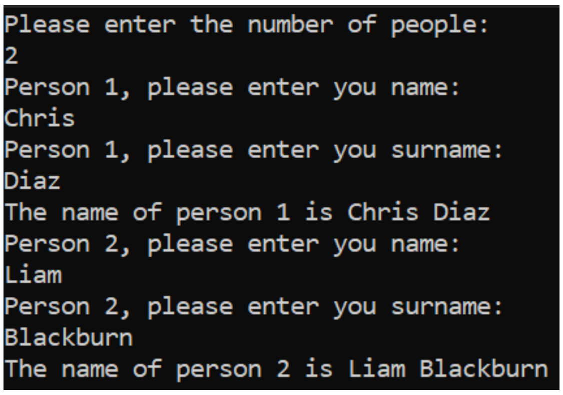


```
public static void names(int numNames)
{
    // For loop that only goes up to wha
    for (int i = 1; i <= numNames; i++)
    {
```

Figure 11 - Loops

The first step was to create a method which returned an integer which was the number of people's names the user wanted to input, this was all learnt and done in exercise 2 and this was transferred over to exercise 3. This method returned an integer which was assigned to num in Figure 9 - Ask for number method. This allowed the integer to then be passed as a parameter for the name's method (James and Freeman, 2003). The next step was to create a loop which had a counter, in this method, which ran up to the number of times the user had inputted (Dane, 2017) (Microsoft, 2021), thus allowing the code in the loop to be executed (Sebesta, 2016), this is shown in Figure 11 - Loops.

The output was as expected, this can be seen in Figure 12 - Exercise 3 output. There was a small issue with the for loop created as a comma was used instead of a semicolon which would not allow program to run, this could have been avoided if loops were properly understood from the start.

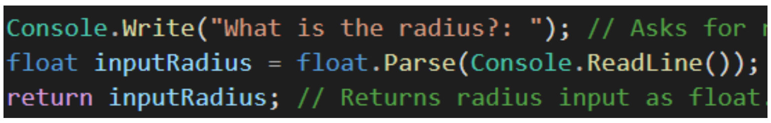


```
Please enter the number of people:
2
Person 1, please enter you name:
Chris
Person 1, please enter you surname:
Diaz
The name of person 1 is Chris Diaz
Person 2, please enter you name:
Liam
Person 2, please enter you surname:
Blackburn
The name of person 2 is Liam Blackburn
```

Figure 12 - Exercise 3 output

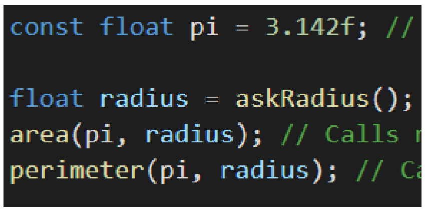
Exercise 4

This exercise needed to calculate the area and the perimeter of a circle, ask the user for the radius, and have pi as a constant of 3.142. The steps were broken down are as follows; step 1, create a constant pi of 3.142; step 2, ask the user for the radius and store in a variable; step 3, create two methods one for the perimeter and one for the area; step 5, each method needs the respective equation and to print out the value. The expected output was two things, the perimeter of the circle and the area of the circle.



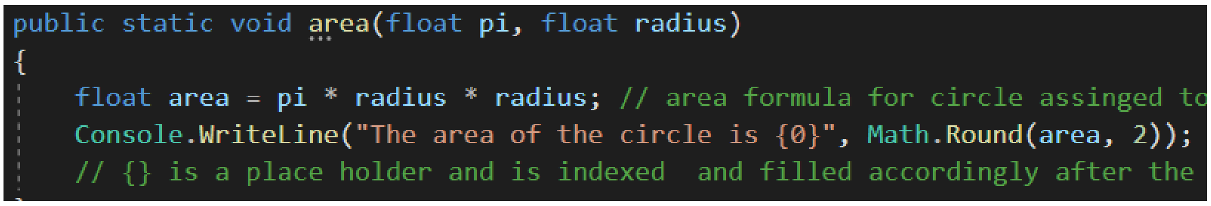
```
Console.Write("What is the radius?: "); // Asks for r
float inputRadius = float.Parse(Console.ReadLine());
return inputRadius; // Returns radius input as float
```

Figure 13 - Ask for radius



```
const float pi = 3.142f; //

float radius = askRadius();
area(pi, radius); // Calls r
perimeter(pi, radius); // Ca
```

Figure 14 - Const pi, method calls

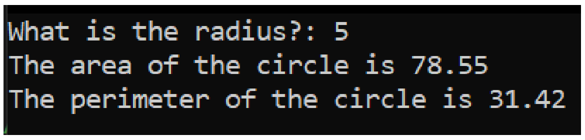


```
public static void area(float pi, float radius)
{
    float area = pi * radius * radius; // area formula for circle assinged to
    Console.WriteLine("The area of the circle is {0}", Math.Round(area, 2));
    // {} is a place holder and is indexed  and filled accordingly after the
```

Figure 15 - Area method

Firstly, the constant pi needs to be created, this can be seen in Figure 14 - Const pi, method calls, this is done using the word const (GeeksforGeeks, 2021) before declaring the data type (Wagner, 2011). The next step was to ask to user to input the radius, this then needed to be converted into of type float, this was done in a sperate method. To convert the input into a float, as all inputs are strings (Holzner, 2003), float.parse() was used this can be viewed in Figure 13 - Ask for radius. This allowed the user to input a decimal number if desired while not using as much memory as the double data type would have (Microsoft, 2021). The input could have been converted to an integer, but this would mean that the user would only be able to input a whole number. This float is passed back and stored in the main as radius. This is then passed as a parameter (Jones and Freeman, 2003) alongside pi into area and perimeter respectively, here the relevant calculation was preformed, the equations were both found on BBC bitesize(n/a). Both the area and perimeter method are of type void as they print the output and do not return anything to the main method (Jones and Freeman, 2003). Math.Round(,) was used to round the decimals to two, this was found in Holzner's (2003) book, although not directly, inside it talk about math.sqrt which was the square root and .round was attempted which worked almost straight away. The problem encountered was that it was not treated as a method the first time. The second time, it was thought that the method needs brackets in which it needs two parameters, the variable we want to round and to how many places, which worked, this should have been researched instead of guessed so it would have worked the first time around.

The output was as expected, this can be seen in Figure 16 - Exercise 4 output.



```
What is the radius?: 5
The area of the circle is 78.55
The perimeter of the circle is 31.42
```

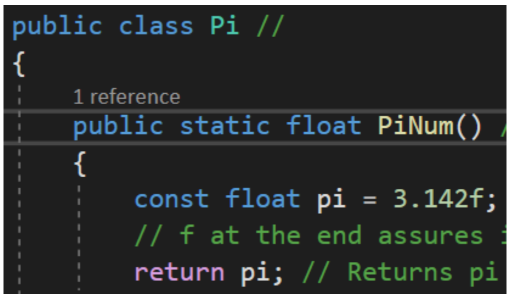
Figure 16 - Exercise 4 output

Exercise 5

This exercise was an extension of exercise 4, the only difference being that the constant pi had to be in a separate class. This was broken down into 3 steps:

- Create a class with a method for pi
- Call pi method from the main in the separate class
- See if additional research was needed to call methods from another class

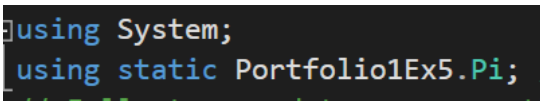
The expected output of this exercise is identical to that of exercise 4.



```
public class Pi //
{
    1 reference
    public static float PiNum() /
    {
        const float pi = 3.142f;
        // f at the end assures i
        return pi; // Returns pi
```

Figure 17 - Pi in another class

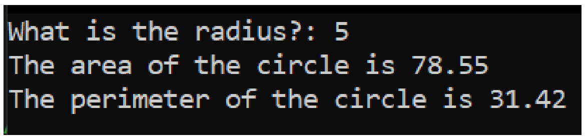
Figure 17 - Pi in another class shows the method created for pi, again the const keyword was assigned to pi, the method returns a number which is stored in pi hence why return pi was added along with the data type float being declared in the method, as there were no parameters needed the brackets were left empty (Jones and Freeman, 2003) (w3schools, n/a). The method was then called in the main within the separate class as a normal method, but this did work immediately. Additional learning was needed to identify how to call a method from another class, the needed code can be seen in Figure 18 - Call method from another class. Grepper (n/a) explained the need to mention at the start that a specific class was being used along with the folder or namespace in which it was in.



```
using System;
using static Portfolio1Ex5.Pi;
```

Figure 18 - Call method from another class

The output was as expected, this is shown in Figure 19 - Exercise 5 output.



```
What is the radius?: 5
The area of the circle is 78.55
The perimeter of the circle is 31.42
```

Figure 19 - Exercise 5 output

Blog – Week 1 (Entered during Week 1)

Week 1 was a fun way to get into c#. The lecture explained a bit about what we were going to be doing this module like the coursework and expectations, there was also a slide in blackboard I found with some basic concepts on memory and key words for c#. The Seminar started off with having some issues with visual code studio as I had downloaded the wrong IDE, but the teacher helped me sort it out. Then following the word document given, I managed to get use to visual code studio fairly easily and then started the portfolio work. I started doing exercises 1, 2 and 4 and they were the easiest as I could reuse code learnt from the class and from last year's module which taught us java. I did struggle a little with exercise 3 and needed to go over last year's Java loops a bit more in depth as they are similar languages. I put a comma down instead of a semi colon and it took me 2 hours to figure out my mistake. I also struggled with exercise 5 as I could not figure out why I could not call in the method from Pi.cs, I then realised I needed to declare I am using the class and use "using static Portfolio1Ex5.Pi;" I figured this out thanks to https://www.codegrepper.com/code-examples/csharp/c%23+how+to+call+methods+from+another+class. I did also need to relearn methods for this. I need to practice C# and all basic things such as methods, lists and loops in this language to make sure I have a solid base for what's to come.

Reflection – Week 1

The group of exercises from group 1 were not particularly challenging, they recapped techniques taught and learnt last academic year, but in other programming languages, this was a fun warm up to start the academic year while ensuring we have a solid foundation for what is to come. No handling exception techniques were used as the exercises ran and performed as expected the first or second time. The issues which occurred in exercise 3 could have been avoided with additional learning from apps such as solo learn or resources such as books or YouTube to ensure confidence before writing code.

Bibliograph – Week 1

BBC BITESIZE., n/a. *Perimeter and Area* [online]. Available from: https://www.bbc.co.uk/bitesize/guides/ztndmp3/revision/6 [Accessed 10 October 2021].

DANE, M., 2017. *For Loops | C# | Tutorial 20* [online video]. Available from: https://www.youtube.com/watch?v=ad38jqt7XDs [Accessed 10 October 2021].

DECHALERT, A., 2021. Is Visual Studio Code Really The Best Code Editor? *Tabine.* [blog online]. 28 Feburary. Available from: https://www.tabnine.com/blog/visual-studio-code-really-the-best-code-editor/ [Accessed 9 October 2021].

GEEKSFORGEKKS, 2021., *Difference between readonly and const keyword in C#* [online]. Available from: https://www.geeksforgeeks.org/difference-between-readonly-and-const-keyword-in-c-sharp/ [Accessed 10 October 2021].

GREPPER., n/a. *C# how to call methods from another class* [online]. Available from: https://www.codegrepper.com/code-examples/csharp/c%23+how+to+call+methods+from+another+class [Accessed 10 October 2021].

HEJLSBERG, A., TORGERSEN, M., WILTAMUTH, S. and GOLDE, P., 2008. *The C# Programming Language* [ebook]. 3rd ed. Seattle, Washington: Pearson Education (US). Available from: https://books.google.co.uk/books?id=_GMLkAEACAAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false [Accessed 10 October 2021].

HOLZNER, S., 2003. *Microsoft Visual C# .NET 2003 Kick Start* [ebook]. Indianapolis, Indiana: Sams Publishing. Available from: https://www.google.co.uk/books/edition/Microsoft_Visual_C_NET_2003_Kick_Start/YVeQDIFUuq4C?hl=en&gbpv=1&dq=float.parse&pg=PA31&printsec=frontcover [Accessed 10 October 2021].

JONES, A. and FREEMAN, A., 2003. *C# FOR JAVA DEVELOPERS.* Redmond, Washington: Microsoft Press.

KROSSING, D., 2019. *11: Get User Input In C# | Preparing You For Project | C# Tutorial For Beginners | C Sharp Tutorial* [online video]. Available from: https://www.youtube.com/watch?v=fZioOGu6DlI [Accessed 10 October 2021].

MICROSOFT, 2021. *Built-in types (C# reference)* [online]. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/built-in-types [Accessed 10 October 2021].

MICROSOFT, 2021. *C# operators and expressions (C# reference)* [online]. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/ [Accessed 10 October 2021].

SEBESTA, W, R., 2016. *Concepts of Programming Languages.* 11th ed. Essex, England: Pearson Education Limited.

WAGNER, B., 2011. *Effective C# (Covers C# 4.0): 50 Specific Way to Improve Your C#.* 2nd ed. Boylston Street, Boston: Pearson Education, Inc.

W3SCHOOLS, n/a. *C# Methods* [online]. Available from: https://www.w3schools.com/cs/cs_methods.php [Accessed 10 October 2021].

Week 2

Overview – Week 2

During Week 2, the fundamentals were gone over, this included:

- Variables
- Compilers & interpreters
- Conditional statements
- Loops
- Arrays

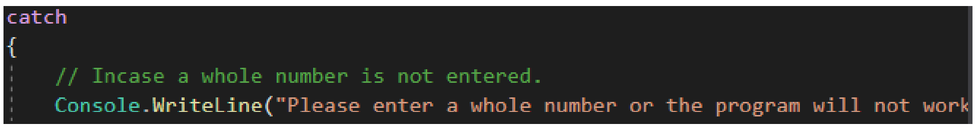
These were thoroughly gone over to ensure the fundamentals for the C# programming language were properly understood so problem, solving techniques could be implemented for the week 2 exercises as they were problem solving exercises. Week 2 contained a breakdown of the exercises. Within each subsection an explanation to what the task was, what the expected outcome was, the research needed, the thought process around the exercise, the features which were incorporated, the issues which occurred, and the source code created were presented.

Exercise 1

The aim of this exercise was to create a program which allowed the user to input a number and then the program would identify and be able to print out if the number was odd or even. The exercise was broken down into three steps:

- Create a method to allow the user to input a number
- Only allow the user to input a whole number
- Create a method to be able to read if a number is even or odd and print the result

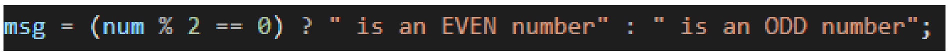
The output should be the inputted number along with a string saying if it was even or odd.



```
catch
{
    // Incase a whole number is not entered.
    Console.WriteLine("Please enter a whole number or the program will not work
```

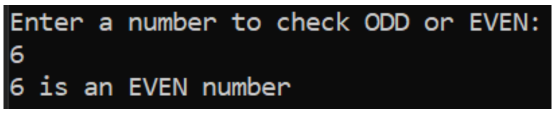
Figure 20 - Exception handling for floats

Firstly, a method to allow a user to input a number was created, this was self-taught in Week 1, then the data type inputted of type string was converted into an integer and sent to main, this was learnt in week 1 thanks to Holzner (2003). Secondly a conditional statement needed to be made to evaluate a condition, within it, '%', which was learnt last academic year and in week 1 (Microsoft, 2021), identifies the remainder of a number divided. Next the method to identify if it was even or odd was created. Within this method, a conditional statement was used, as additional learning was being done through solo learn (n/a) a different way to write if-else statements was taught compared to lectures. This technique used a '?' as an operator and ':' to help differentiate the if and else statement, this allowed for the left side of the ':' to print the if statement and the remaining side to print the else, this can be seen implemented in Figure 21 - Conditional statement . The input method was incapsulated in a try catch block shown in Figure 20 - Exception handling for floats, the try allowed the code wanted to be run and the catch allowed a message to be displayed if the try code failed (Microsoft, 2021) (Dane, 2017), in this case the catch would only display an error if the user inputted a decimal number as floats and doubles are not being used, only integers.



```
msg = (num % 2 == 0) ? " is an EVEN number" : " is an ODD number";
```

Figure 21 - Conditional statement



```
Enter a number to check ODD or EVEN:
6
6 is an EVEN number
```

Figure 22 - Exercise 1 output

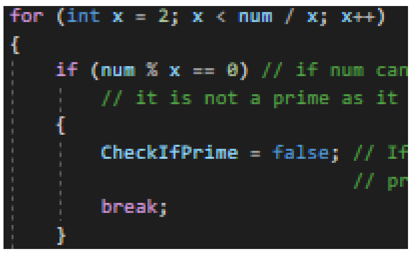
No problems arose with this exercise and shown in Figure 22 - Exercise 1 output the output was as expected.

Exercise 2

Exercise 2 was an extension of exercise 1, with the extension being to see if the numbered entered was a prime number. The breakdown of this project is as follows:

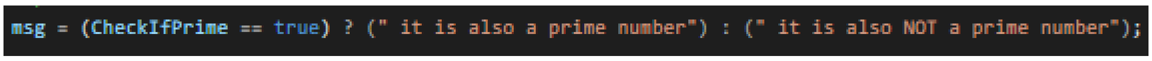
- Use previous exercise as a base
- Add a new method to check if prime
- Loop over every number up to entered number to check if divisible by it
- Print within CheckIfPrime method if it is or is not a prime number

The expected outcome was that the output would accurately predict if the number was odd or even along with if it was a prime number or not.



```
for (int x = 2; x < num / x; x++)
{
    if (num % x == 0) // if num can
        // it is not a prime as it
    {
        CheckIfPrime = false; // If
                              // pr
        break;
    }
```

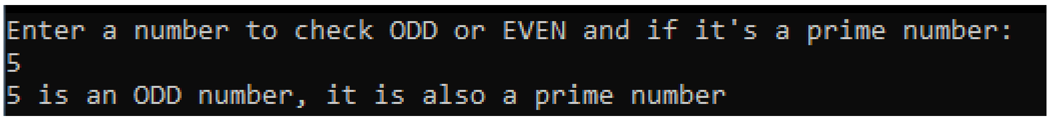
Figure 23 - Check if prime



```
msg = (CheckIfPrime == true) ? (" it is also a prime number") : (" it is also NOT a prime number");
```

Figure 24 - If prime print is, if not print is not

The method to check if the entered number by the user was prime or not was created, which passed as a parameter the number from the input number method from the main method when the relevant method was called. This number was the variable num in the for loop, which can be viewed in Figure 23 - Check if prime. This loop ran up to when x was the same as num divided by 2, Each time the loop occurred it was incremented by 1, this was learnt in week 1. Within the loop there was a conditional statement which evaluates if the num is divisible by x, it means it is not prime as x is not equal to num, so it would equate to false which would print not be a prime number as it is divisible by something other than itself. If the if statement returns false then the conditional statement in Figure 24 - If prime print is, if not print is not, will print the right string stored in the string varibale msg as it does not meet the requirements in the if statement asking if true.



```
Enter a number to check ODD or EVEN and if it's a prime number:
5
5 is an ODD number, it is also a prime number
```

Figure 25 - Exercise 2 output

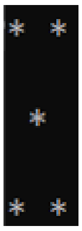
The output can be viewed in Figure 25 - Exercise 2 output, as shown within the output is as expected. The prime number was not being accurately predicted the first few times and guess work was used, as it was guessed work the code did not run. After this written pseudocode was used to help break down the for loop along with a drawing of how the loop would run, this helped visualise the errors along with helping understand what was being written more. This should have been done from the start and could have been used for week 1 to help visualise what was about to be written.

Exercise 3

The goal of this exercise was to allow the user to enter any odd number and to print out an X made of '*' with the height being the amount of star. The height had to be the same amount as the inputted number. The exercise was broken down as follows:

- Create a method to allow the user to enter a number and not an even number and convert string to int
- Create a method for the creation of the star and pass the inputted integer as a parameter
- Create a nested loop (learnt last academic year in other programming languags) with the outer loop printing a new line and the inner loop printing horizontally on the line

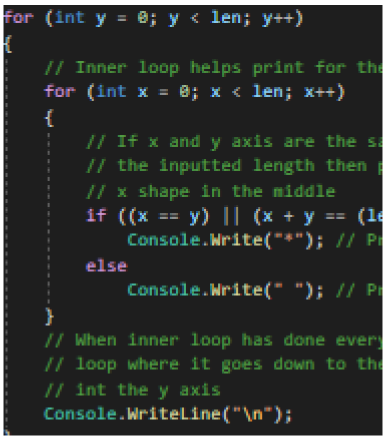
The expected outcome as prompting the user for input of integer type, then to print a star as shown in Figure 26 - '*' Star.



```
* *

 *

* *
```

Figure 26 - '' Star*

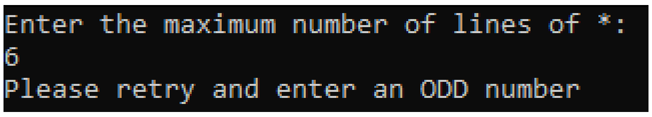
The method to ask for user input was created using Console.WriteLine to ask the user a question, Console.ReadLine to read that input and it was assigned and converted to an int and stored in the variable len, this was learnt in week 1 through Hozner (2003). This value was returned to main so it could be passed as a parameter for the printStar method which was learnt in week 1. The first challenge presented was how to create the nested loop this was figured out thanks to Curry (2019) and Nakiv and team (2019). The goal was to ensure the outer and nested for loop went up to the number inputted by the user, ensuring the number of lines was no more than the integer inputted. Whenever the nested loop broke the outer loop would then move down to a new line hence the "/n" which was learnt last academic year in python and trailed to see if it was implemented the same which it was. The nested loop was a bit more complicated, this needed written pseudocode and a step-by-step hand drawn image to see when to print an asterisk and when not to. One asterisk line in the x was figured out as an x and y axis graph was hand drawn and when x was exactly equal to y it always printed one of the lines in the x with asterisks. This was written in the form (x == y), this was done in brackets as an OR operator needed to be used to print the other line of the x. This the logical OR operator in the nested loop tutorials along with week 1 (Microsoft, 2021). The other line which completed the x was identified as in the graph the x and y axis started from 0 as this was the number they were assigned in the loop. This meant that each asterisk found in the other line for the x conformed to (x+y) == (len - 1), so this was incorporated with (x==y) and the OR operator, this can be seen in Figure 27 - Nested loop. This was inserted into an if else statement, this meant if it did not meet these requirements then it would print an empty space.



```
for (int y = 0; y < len; y++)
{
    // Inner loop helps print for the
    for (int x = 0; x < len; x++)
    {
        // If x and y axis are the sa
        // the inputted length then p
        // x shape in the middle
        if ((x == y) || (x + y == (le
            Console.Write("*"); // Pr
        else
            Console.Write(" "); // Pr
    }
    // When inner loop has done every
    // loop where it goes down to the
    // int the y axis
    Console.WriteLine("\n");
```

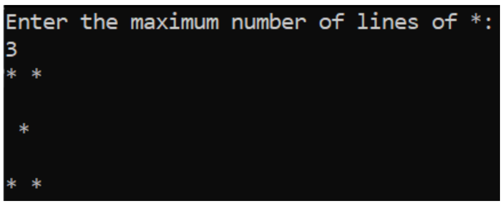
Figure 27 - Nested loop

The even number and odd number inputs output can be seen in Figure 28 - Exercise 3 output even number and Figure 29 - Exercise 3 output odd number. The even number output worked thanks to what was written in Figure 30 - Even number not allowed, here it evaluates if a number is odd by checking if it has a remainder when divided by 2. There was an issue getting the code to run due to no knowing the difference between Console.Write() and Console.WriteLine(). As it was not known the output was as is in Figure 31 - Exercise 3 incorrect output. After the lecturer mentioned the difference, the issue was easy to correct. This should have been remembered from week 1.



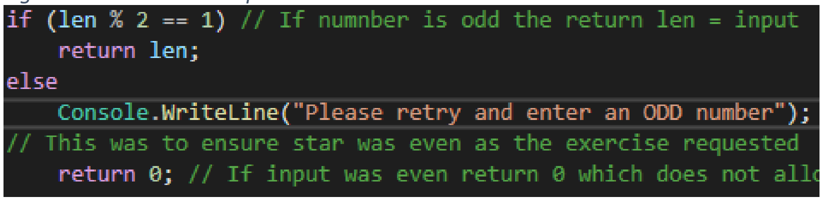
```
Enter the maximum number of lines of *:
6
Please retry and enter an ODD number
```

Figure 28 - Exercise 3 output even number



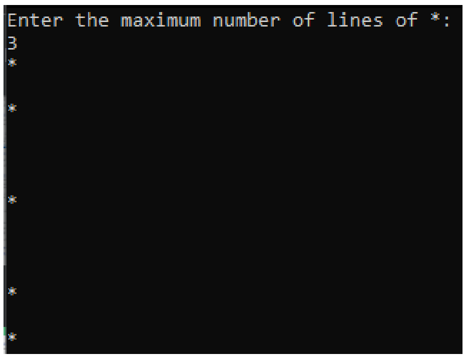
```
Enter the maximum number of lines of *:
3
* *

 *

* *
```

Figure 29 - Exercise 3 output odd number



```
if (len % 2 == 1) // If numnber is odd the return len = input
    return len;
else
    Console.WriteLine("Please retry and enter an ODD number");
// This was to ensure star was even as the exercise requested
    return 0; // If input was even return 0 which does not allo
```

Figure 30 - Even number not allowed



```
Enter the maximum number of lines of *:
3
*


*




*




*


*
```

Figure 31 - Exercise 3 incorrect output

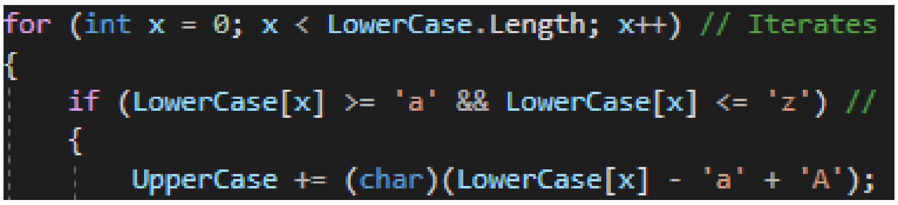
Exercise 4

Exercise 4 asked the user to input a sentence in lower case, upper case, or both and to return the sentence in all upper cases without using library functions such as String.Upper. The problem was broken down as follows:

- Create a method which allows the user to input a sentence store in a variable in main and pass as parameter to ToUpperCase method.
- Create method to turn all characters in the sentence to upper case.
- Create an empty string to store upper case characters.
- Create a loop to loop over each character in the sentence and breaks when the length of sentence is reached.
- If lowercase figure out a way to turn to upper case and store in upper case variable.
- Else if character is already upper case from the lower-case string add straight to upper case string variable.
- Print upper case.

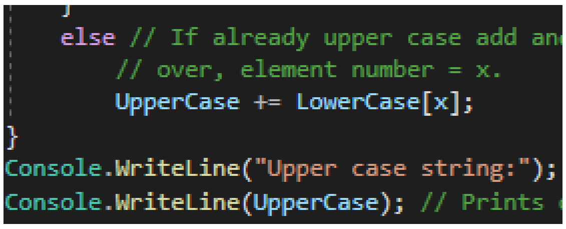
The expected output is the user to ne prompt to input a sentence and the output to be all in upper case no matter what case the character entered is in.

The method readInput was created to allow the user to input a sentence and input was returned to main to be turned to upper case when the ToUpperCase method was called and the input was passed as a parameter, this was all learnt in week 1. Next the method to convert upper case characters was created and the code within inserted into a try catch statement to help identify issues, this was learnt for week 2 exercise 1. Next a for loop was created which looped after each character up to the inputs length this was done with variable.Length (Microsoft, 2021)in the loop as shown in Figure 32 - Lower case to upper. After this an if else statement was created, the if could not be figured out at this stage so the else was created so that if it was already upper case, it was added and assigned to the already created empty upper case string variable. Finally, the whole upper-case variable was printed, this is shown in Figure 33 - Upper case to upper case + print along with the else statement. As the if statement had not been attempted but implemented with no success a lot of research and logically guessed trails were done there was no success in receiving the desired output for a whole day. Thanks to Hecknal (2019) an idea was sparked, although Hecknal used the variable.ToUpper library function Hecknal also included this in the code "[a-z]". This helped as it seemed as though lower-case characters and uppercase characters could have their own range or so to speak, so the range ensure the character in the same index as the loop which was lower case had to be in the lower-case alphabet range, this took a lot of trail and error and the AND operator was used to ensure the character was between lower case a and lower case z. To turn to upper case the lower case was deleted, and the upper-case version of the same character was added to the upper-case variable which allowed the program to run.



```
for (int x = 0; x < LowerCase.Length; x++) // Iterates
{
    if (LowerCase[x] >= 'a' && LowerCase[x] <= 'z') //
    {
        UpperCase += (char)(LowerCase[x] - 'a' + 'A');
```

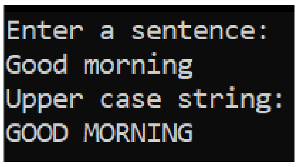
Figure 32 - Lower case to upper



```
    }
    else // If already upper case add an
        // over, element number = x.
        UpperCase += LowerCase[x];
}
Console.WriteLine("Upper case string:");
Console.WriteLine(UpperCase); // Prints
```

Figure 33 - Upper case to upper case + print

The output was as expected although took many attempts and a lot of research with only one find that helped. More problem-solving techniques need to be practised as this exercise took longer than expected also to be more creative in the way problems were approached. The output can be seen in Figure 34 - Exercise 4 output.



```
Enter a sentence:
Good morning
Upper case string:
GOOD MORNING
```

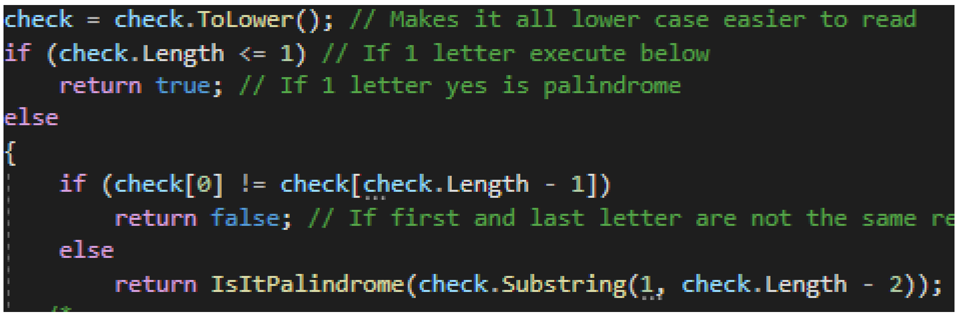
Figure 34 - Exercise 4 output

Exercise 5

Exercise 5 was to check if an inputted word by the user was a palindrome or not. The problem was broken down as follows:

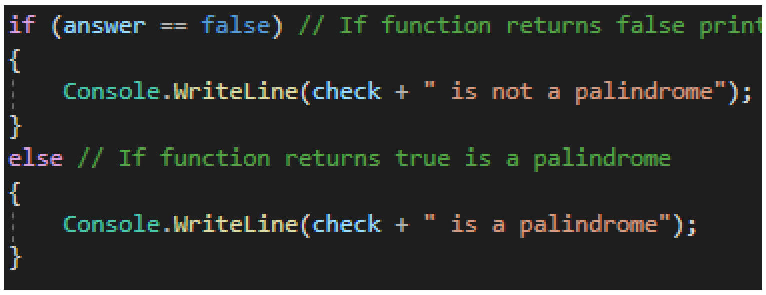
- Create a method which allows the user to input a word store in a variable in main and pass as parameter to is It Palindrome method.
- Make sure all characters are lower case.
- Compare last node to first node.
- Compare second node to penultimate node…
- Return bool to main.
- Conditional statement in main to identify if palindrome or not.

The expected output should be lower case word with a string next to it declaring if it is a palindrome or not.



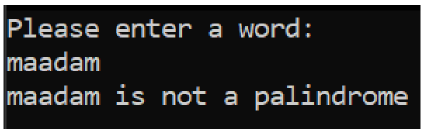
```
check = check.ToLower(); // Makes it all lower case easier to read
if (check.Length <= 1) // If 1 letter execute below
    return true; // If 1 letter yes is palindrome
else
{
    if (check[0] != check[check.Length - 1])
        return false; // If first and last letter are not the same re
    else
        return IsItPalindrome(check.Substring(1, check.Length - 2));
```

Figure 35 - Check if palindrome



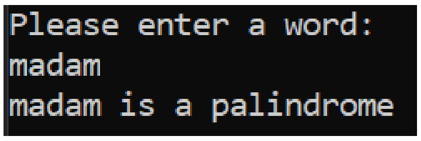
```
if (answer == false) // If function returns false print
{
    Console.WriteLine(check + " is not a palindrome");
}
else // If function returns true is a palindrome
{
    Console.WriteLine(check + " is a palindrome");
}
```

Figure 36 - If else statement bool



```
Please enter a word:
maadam
maadam is not a palindrome
```

Figure 37 - Exercise 5 output not palindrome



```
Please enter a word:
madam
madam is a palindrome
```

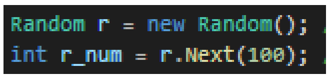
Figure 38 - Exercise 5 output palindrome

Exercise 6

Exercise 6 was to create a random number guessing game which allowed the user five guesses. The game was broken down with handwritten pseudocode as follows:

- Create a method for the game.
- Assign a random number between 0-100 to a variable in the method.
- For loop which gives the user 5 guesses.
- Start game asking player to input a number between 0-100.
- Allow the user to input an integer guess to store in a variable.
- Find the difference between guess and random number.
- Use If statement to break loop when random number == guess.
- Else if statement to check if number is in range.
- Check difference between the two with nested if-else statements.
- If else statement outside of loop to allow the user to decide if they want to play again.
- Recursive method calls to allow the user to play again if desired.
- Print thank you for playing if player decides to leave game which exits method and prints line from main which is after method call.

The expected output should be a string which prompts the user to guess the number, with some hints as to how far the guess is from the random number.

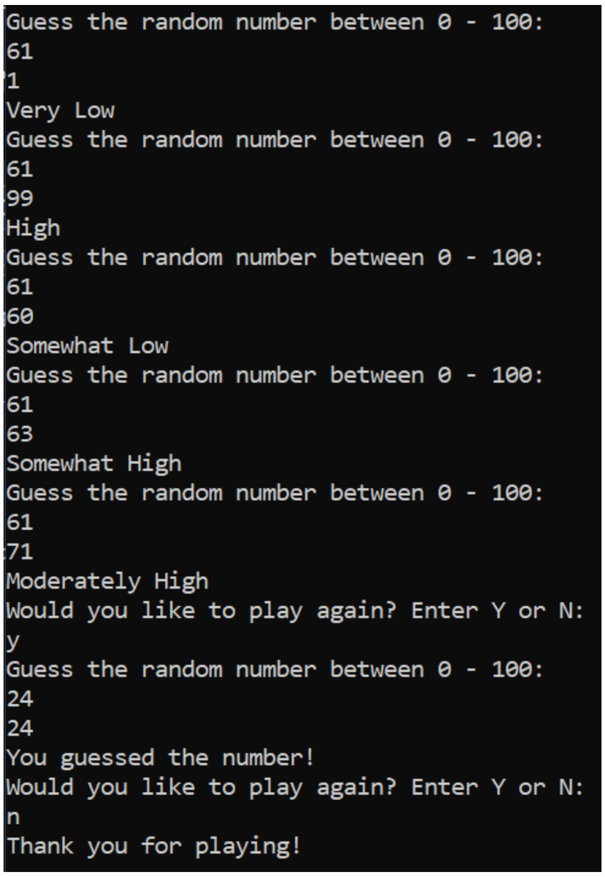


```
Random r = new Random();
int r_num = r.Next(100);
```

Figure 39 - Random number

The background research needed for this was how to create a random number, this is done with new Random(), initialising the random class, this was then capped to 100 (ULLMAN, 2003) and assigned to r_num. This can be seen in Figure 39 - Random number.

The problem which occurred with this was figuring out whether the AND or OR operator should be used to help find how far off the guess the user was. Turns out both were needed. This was to ensure that the range did not overlap in another statement which had previously cause an undesired outcome. The AND was used to ensure the difference was in a specific range, the OR was to use two separate ranges which were needed as the difference calculated could have been a positive or negative integer, meaning it could have been above the guess or below the guess as the difference was calculated with guess – random number. Following this nested if else statements were needed to identify if the guess was above or below the random number and this printout higher if the guess is bigger than guess else lower. After this the expected outcome was the out put which can be seen in Figure 40 - Exercise 6 output with random number printed.



```
Guess the random number between 0 - 100:
61
1
Very Low
Guess the random number between 0 - 100:
61
99
High
Guess the random number between 0 - 100:
61
60
Somewhat Low
Guess the random number between 0 - 100:
61
63
Somewhat High
Guess the random number between 0 - 100:
61
71
Moderately High
Would you like to play again? Enter Y or N:
y
Guess the random number between 0 - 100:
24
24
You guessed the number!
Would you like to play again? Enter Y or N:
n
Thank you for playing!
```

Figure 40 - Exercise 6 output with random number printed

Blog – Week 2 (Entered during Week 2)

This week was more challenging, first we learnt about a few things, memory block, recapped bits and bytes, interpreters and compilers, control-flow, operators, talked about test driven development, and arrays. Then we had our seminar which I felt a bit sick in and didn't do much work, when I got home I did the work, I found them all a little more challenging than last weeks but fun, I did struggle a lot with Ex3 and Ex6, especially with relearning and implementing nested loops, I did also go over try catch so that I could implement that in my code along with starting to look over the big O notation to ensure I have optimised code. I did have an issue with my if-else statements as I accidently put in a semicolon and it wouldn't run, thanks to https://stackoverflow.com/questions/55625259/else-cannot-start-a-statement-miscellanousfiles-csproj/55625271 I realised my mistake and fixed it. I also struggled with turning the sentences to upper case but found on here https://stackoverflow.com/questions/913090/how-to-capitalize-the-first-character-of-each-word-or-the-first-character-of-a you could just use the alphabet as such and get rid of 'a' for example and replace it by adding 'A' and letting it loop over the alphabet, the example used a library but I found a way around it. I also had issues with figuring out palindromes in Ex5 as I was trying to split the word in half but not all words are even, and then I remembered the substring method from last year and looked over my notes, same with the nested loop. I still need to practice some more with my code as I feel as though I am writing a lot and will try go deeper into methods and loops to reuse code and go over my code again to ensure I used test driven development. I also successfully did the final exercise although as mentioned I believe it is too long and has too many if-else statements which I will try cut down.

Reflection – Week 2

The group of exercises from this week were a little more challenging, especially exercises 3 and 6. Week 2 explored techniques both taught last academic year and in week 1 along with needing some additional research, this was an interesting group of exercises which tested which techniques were known along with assessing which creative ways were used to implement them. It also was a step above and extension from the foundations learnt from week 1. Handling exception techniques were used as the exercises ran and performed not as expected after a few times. As nested loops were an issue, the difference between Console.Write and Console.WriteLine were unknown, and some basic mistakes were made some additional learning recapping the basics should be done to ensure small mistakes do not become an often occurrence.

Bibliograph – Week 2

CURRY, C., 2019. *C# Programming tutorial 29 – Nested for Loops (Triangles and Pyramids)* [online video]. Available from: https://www.youtube.com/watch?v=HUyLBV-DQ4U [Accessed 15 October 2021].

DANE, M., 2017. *Exception Handling | C# | Tutorial 24* [online video]. Available from: https://www.youtube.com/watch?v=ZJRg8nrNeeA [Accessed 15 October 2021].

HENCKEL, J., 2019. How to capitalize the first character of each word, or the first character of a whole string, with C#? *Stackoverflow* [blog online]. 9 Aug. Available from: https://stackoverflow.com/questions/913090/how-to-capitalize-the-first-character-of-each-word-or-the-first-character-of-a [Accessed 16 October 2021].

HOLZNER, S., 2003. *Microsoft Visual C# .NET 2003 Kick Start* [ebook]. Indianapolis, Indiana: Sams Publishing. Available from: https://www.google.co.uk/books/edition/Microsoft_Visual_C_NET_2003_Kick_Start/YVeQDIFUuq4C?hl=en&gbpv=1&dq=float.parse&pg=PA31&printsec=frontcover [Accessed 10 October 2021].

MICROSOFT, 2021. *Built-in reference types (C# reference)* [online]. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/reference-types#the-string-type [Accessed 16 October 2021].

MICROSOFT, 2021. *Built-in types (C# reference)* [online]. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/built-in-types [Accessed 10 October 2021].

MICROSOFT, 2021. *C# operators and expressions (C# reference)* [online]. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/ [Accessed 10 October 2021].

MICROSFOT, 2021., *try-catch (C# Reference)* [online]. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-catch [Accessed 15 October 2021].

NAKIV, S. AND TEAM., 2019. *Programming Basics with C# Book and Video Lessons* [ebook]. London, England: Faber Publishing. Available from: https://www.google.co.uk/books/edition/Programming_Basics_with_C/vzKyDwAAQBAJ?hl=en&gbpv=1&dq=nested+loop+C%23&pg=PA224&printsec=frontcover [Accessed 16 October 2021].

SOLOLEARN, n/a. *The ?: operator* [online]. Available from: https://www.sololearn.com/learning/1080/2599/5403/1 [Accessed 15 October 2021].

THEGENERAL, 2019., 'else' cannot start a statement. *Stackoverflow* [Blog online]. 11 April. Available from: https://stackoverflow.com/questions/55625259/else-cannot-start-a-statement-miscellanousfiles-csproj/55625271 [Accessed 16 October 2021].

ULLMAN, C., KAUFFMAN, J., HART, C., SUSSMAN, D. AND MAHARRY, D., 2003. *Beginning ASP.NET 1.1 with Visual C# .NET 2003* [ebook]. Indianaplois, Indiana: Wiley Publishing Inc. Available from: https://www.google.co.uk/books/edition/Beginning_ASP_NET_1_1_with_Visual_C_NET/NY3URTiwTrgC?hl=en&gbpv=1&dq=new+random+()+C%23&pg=PA151&printsec=frontcover [Accessed 16 October 2021].

Week 3

Overview – Week 3

During Week 3, more in depth techniques were taught and learnt, this included:

- Enumerated data types
- Structured data
- Classes & objects
- Polymorphism
- Passing by reference & value
- Recursion

These were a step up from the fundamentals for the C# programming language, this meant more research was needed and more source code needed to be written. This week's work also included problems which needed to be solved with some techniques learnt in week 2 lectures such as jagged arrays. This section contains a breakdown of the exercises, in each subsection and an explanation to what the task was, what the expected outcome was, the research needed, the thought process around the exercise, the features which were incorporated, the issues which occurred, and the source code created were all presented.

Exercise 1

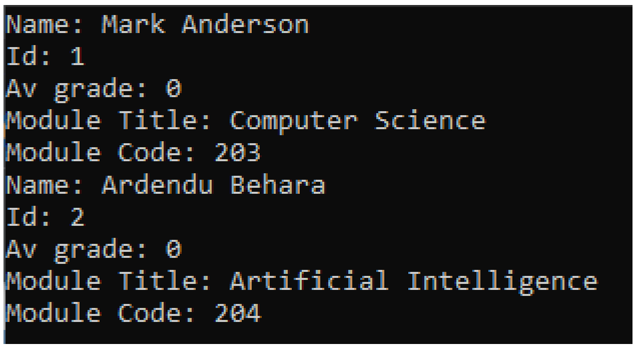
Exercise 1 was to add two new fields to a struct of an already functioning piece of code. The problem was broken down as follows:

- Read and try to understand the give code.
- Learn about structs.
- Follow techniques given to add the two new string fields.

The expected output should be two student each with: a name, an id, an average grade, a module title, and a module code.

The code was simple to understand and made the two required fields to be added to the piece of code, simple as a template was given to how other fields were implemented. Dingle (2017) helped fully conceptualise how a struct works paired with the lecture slides.

As shown in Figure 41 - Exercise 1 output the output is as expected, no errors occurred.



```
Name: Mark Anderson
Id: 1
Av grade: 0
Module Title: Computer Science
Module Code: 203
Name: Ardendu Behara
Id: 2
Av grade: 0
Module Title: Artificial Intelligence
Module Code: 204
```

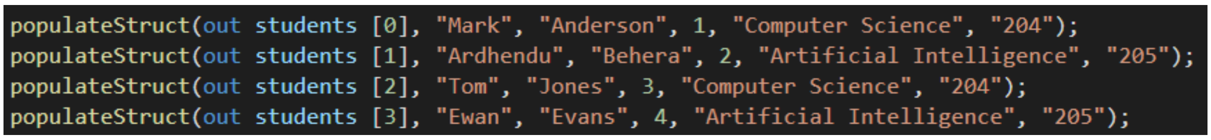
Figure 41 - Exercise 1 output

Exercise 2

Exercise 2 was to alter the code from exercise 1. It asked to create an array which held 4 students, in the main method. Each student had to have a specific name. The problem was broken down as follows:

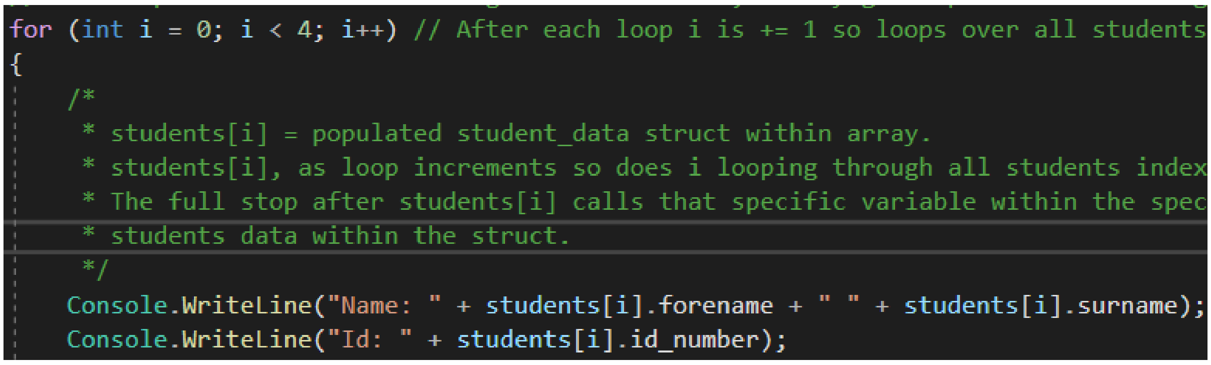
- Create a new array which will holds 4 values, just like Java which was learnt last academic year.
- Populate the struct, see if the struct can be populated at the same time. This can be seen in Figure 42 - Array populated with a struct of data next to array.
- Create new method to print all students.
- Create a loop that loops 4 times over each item in the array, for each student print out all the data for the respective student. See Figure 43 - Loop over array.

The expected output should be four students each with: a name, an id, an average grade, a module title, and a module code.



```
populateStruct(out students [0], "Mark", "Anderson", 1, "Computer Science", "204");
populateStruct(out students [1], "Ardhendu", "Behera", 2, "Artificial Intelligence", "205");
populateStruct(out students [2], "Tom", "Jones", 3, "Computer Science", "204");
populateStruct(out students [3], "Ewan", "Evans", 4, "Artificial Intelligence", "205");
```

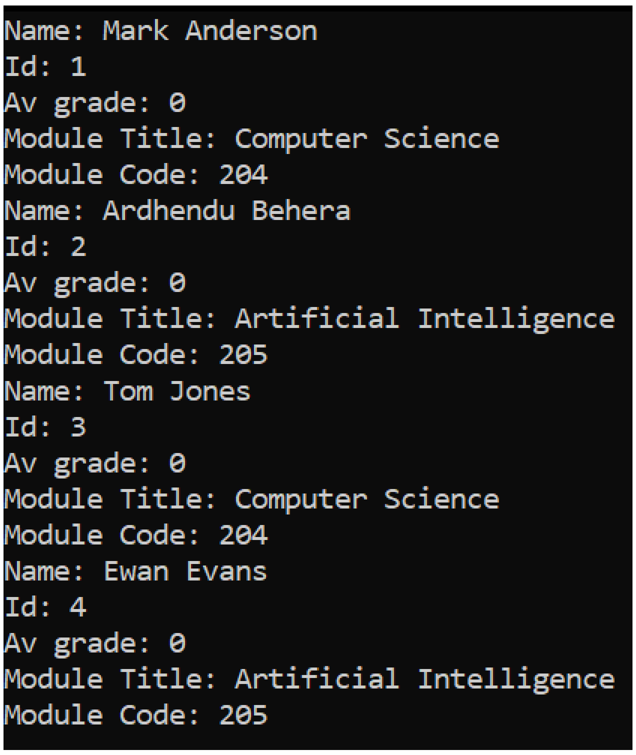
Figure 42 - Array populated with a struct of data next to array



```
for (int i = 0; i < 4; i++) // After each loop i is += 1 so loops over all students
{
    /*
     * students[i] = populated student_data struct within array.
     * students[i], as loop increments so does i looping through all students index
     * The full stop after students[i] calls that specific variable within the spec
     * students data within the struct.
     */
    Console.WriteLine("Name: " + students[i].forename + " " + students[i].surname);
    Console.WriteLine("Id: " + students[i].id_number);
```

Figure 43 - Loop over array

The output was as expected which can be seen in Figure 44 - Exercise 2 output. There was a problem running the code at first as students did not declare the array type with the squared brackets next to it (Overiq.com, 2020). The problem was also solved by as the same method did not accept an array as a parameter.



```
Name: Mark Anderson
Id: 1
Av grade: 0
Module Title: Computer Science
Module Code: 204
Name: Ardhendu Behera
Id: 2
Av grade: 0
Module Title: Artificial Intelligence
Module Code: 205
Name: Tom Jones
Id: 3
Av grade: 0
Module Title: Computer Science
Module Code: 204
Name: Ewan Evans
Id: 4
Av grade: 0
Module Title: Artificial Intelligence
Module Code: 205
```

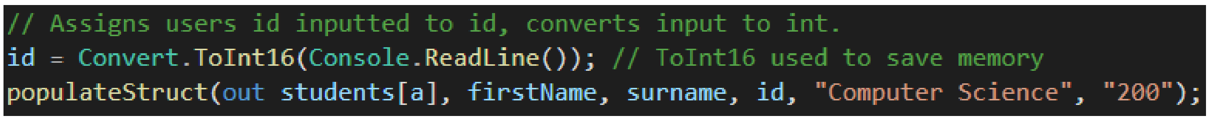
Figure 44 - Exercise 2 output

Exercise 3

Exercise 3 was an extension from exercise 2. Here the exercise needed each student had to have a specific name. The problem was broken down as follows:

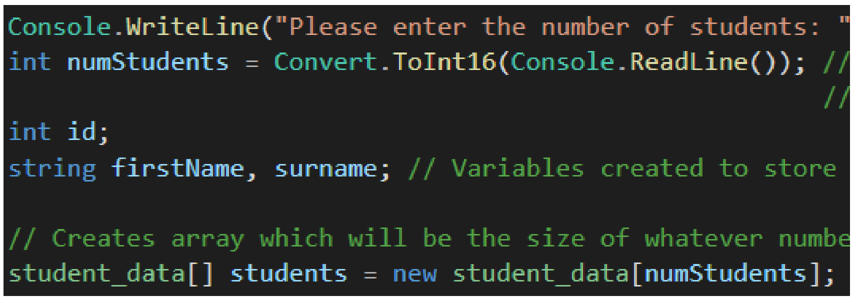
- Allow the user to input a number.
- Create an array which will hold the number of values a user input. Shown in Figure 46 - Dynamic array, user chooses size.
- Loop over array with length already assigned with inputted number. This can be seen in Figure 47 - Loops over each array asks user for input along with asking the user for input.
- Each loop was for 1 student and loop ends when reaches the inputted number.
- Each loop asked the user for name and id and populate struct data at the end of each loop, so data was saved. This can be seen in Figure 45 - Populate struct at the end of each loop.
- Print all students.

The expected output should be x number of students each with an inputted name, an input id, an average grade, a module title, and a module code.



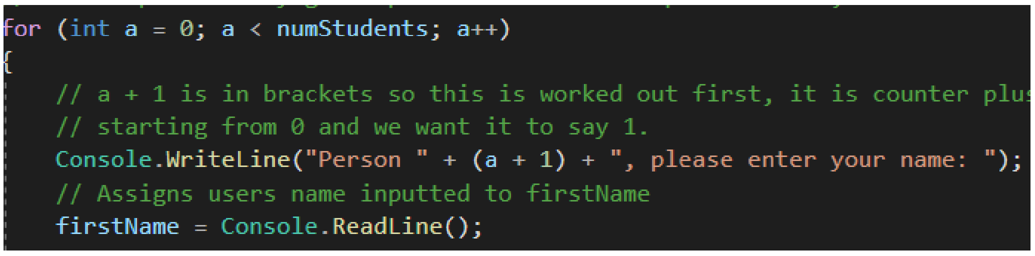
```
// Assigns users id inputted to id, converts input to int.
id = Convert.ToInt16(Console.ReadLine()); // ToInt16 used to save memory
populateStruct(out students[a], firstName, surname, id, "Computer Science", "200");
```

Figure 45 - Populate struct at the end of each loop



```
Console.WriteLine("Please enter the number of students: "
int numStudents = Convert.ToInt16(Console.ReadLine()); //
                                                       //

int id;
string firstName, surname; // Variables created to store

// Creates array which will be the size of whatever numbe
student_data[] students = new student_data[numStudents];
```

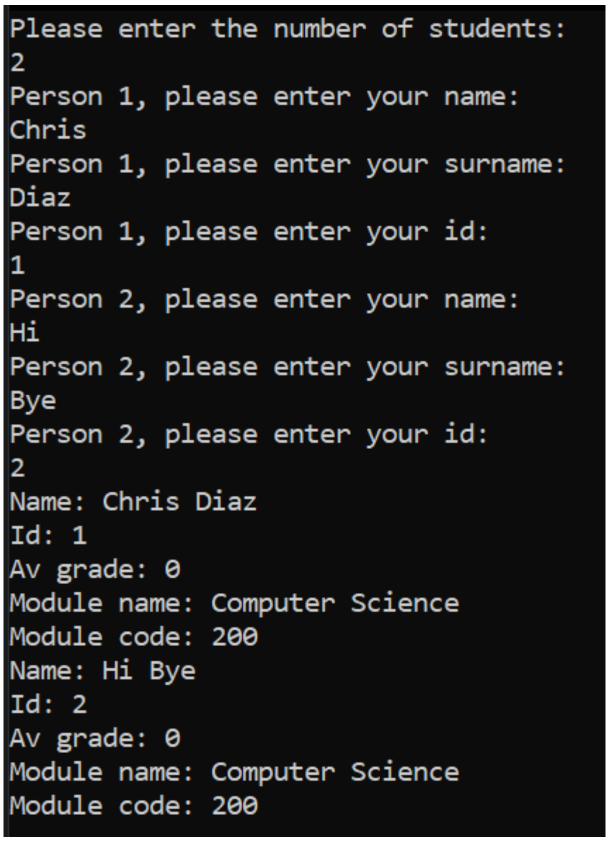
Figure 46 - Dynamic array, user chooses size



```
for (int a = 0; a < numStudents; a++)
{
    // a + 1 is in brackets so this is worked out first, it is counter plu
    // starting from 0 and we want it to say 1.
    Console.WriteLine("Person " + (a + 1) + ", please enter your name: ");
    // Assigns users name inputted to firstName
    firstName = Console.ReadLine();
```

Figure 47 - Loops over each array asks user for input

The output was as expected allowing the user to choose the number of students to input and allowed the user to input their first name, surname, and id. This was shown in Figure 48 - Exercise 3 output. No problems occurred and no new techniques needed as needed techniques were learnt in previous exercises and lectures.



```
Please enter the number of students:
2
Person 1, please enter your name:
Chris
Person 1, please enter your surname:
Diaz
Person 1, please enter your id:
1
Person 2, please enter your name:
Hi
Person 2, please enter your surname:
Bye
Person 2, please enter your id:
2
Name: Chris Diaz
Id: 1
Av grade: 0
Module name: Computer Science
Module code: 200
Name: Hi Bye
Id: 2
Av grade: 0
Module name: Computer Science
Module code: 200
```

Figure 48 - Exercise 3 output

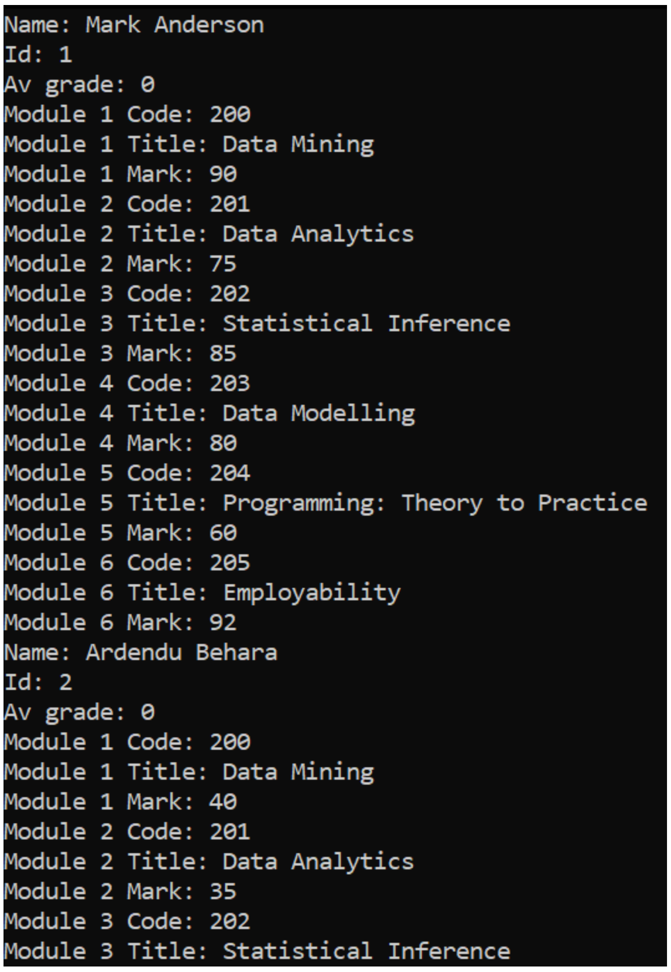
Exercise 4

Exercise 4 was to be made of the basis of the example 1 code given. Here the exercise needed a new structure to hold module data which would be held within student data and populated in the main method. There were two students, and each needed separate data. The problem was broken down as follows:

- Create a new struct for module data.
- Create an array in student_data struct to hold module data.
- Create a populate method for the module struct.
- Ensure the method which populates struct accepts an array as a parameter.
- Create an array which will hold the module data struct.
- Populate module data in accordance with array (considered using a 2d/jagged array).
- Populate module struct with a sperate array of two holding just two students.
- Pass the student whole array through the print method.
- For loop each student with module data.
- Print each student with its own module data.

The expected output should be two students each with a name, an id, an average grade, six module titles, six module codes and six module marks.

This did not work as a single array just did not give the expected output, as a jagged array was considered to help, this was attempted. Deital and Deital (2005) helped understand jagged arrays in C# as this was learnt last academic year in Java and Python. Each row in the jagged array was assigned to a student and each column assigned six modules to the student. This worked and the output was as expected and can be viewed in Figure 49 - Exercise 4 output.



```
Name: Mark Anderson
Id: 1
Av grade: 0
Module 1 Code: 200
Module 1 Title: Data Mining
Module 1 Mark: 90
Module 2 Code: 201
Module 2 Title: Data Analytics
Module 2 Mark: 75
Module 3 Code: 202
Module 3 Title: Statistical Inference
Module 3 Mark: 85
Module 4 Code: 203
Module 4 Title: Data Modelling
Module 4 Mark: 80
Module 5 Code: 204
Module 5 Title: Programming: Theory to Practice
Module 5 Mark: 60
Module 6 Code: 205
Module 6 Title: Employability
Module 6 Mark: 92
Name: Ardendu Behara
Id: 2
Av grade: 0
Module 1 Code: 200
Module 1 Title: Data Mining
Module 1 Mark: 40
Module 2 Code: 201
Module 2 Title: Data Analytics
Module 2 Mark: 35
Module 3 Code: 202
Module 3 Title: Statistical Inference
```

Figure 49 - Exercise 4 output

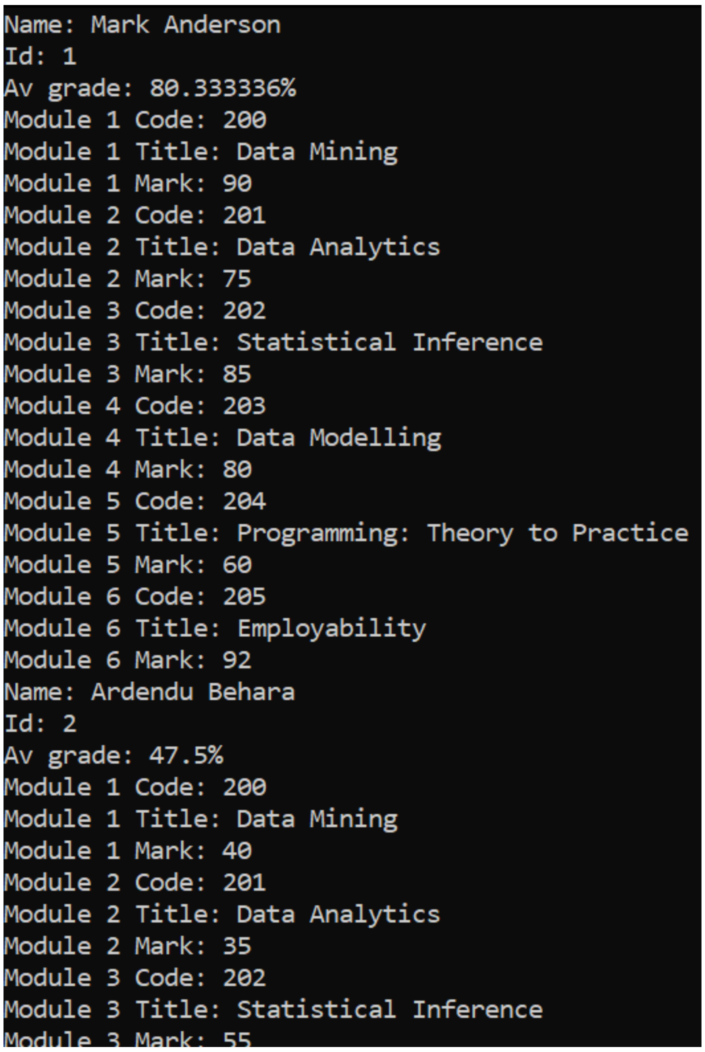
Exercise 5

Exercise 5 was an extension of exercise 4. Here the exercise needed the average grade to be calculated for the two students. The problem was broken down as follows:

- Do not assign a value to average grade in the populate struct method.
- Allow the populate struct method to pass a float average grade as a decimal average grade was wanted not a whole number.
- Create a structure within this method two hold two average grades, one for each student.
- Create a nested loop, outer to identify the student, inner to identify the modules for the student.
- Add and assign each grade in nested loop to an already created empty variable within method.
- If the module number within the nested loop reaches 6 then divide the 6 added grades by 6 to calculate the average and return to main.
- Ensure the correct array is passed through as a parameter to populate struct.
- Print each student with its own data.

The expected output should be the same as exercise 4, apart from the average grade, which should have printed the accurate result.

The output is as expected which can be seen in Figure 50 - Exercise 5 output. No major problems occurred, and the problem took a couple of tried to complete but was done with existing knowledge accumulated over the exercises.



```
Name: Mark Anderson
Id: 1
Av grade: 80.333336%
Module 1 Code: 200
Module 1 Title: Data Mining
Module 1 Mark: 90
Module 2 Code: 201
Module 2 Title: Data Analytics
Module 2 Mark: 75
Module 3 Code: 202
Module 3 Title: Statistical Inference
Module 3 Mark: 85
Module 4 Code: 203
Module 4 Title: Data Modelling
Module 4 Mark: 80
Module 5 Code: 204
Module 5 Title: Programming: Theory to Practice
Module 5 Mark: 60
Module 6 Code: 205
Module 6 Title: Employability
Module 6 Mark: 92
Name: Ardendu Behara
Id: 2
Av grade: 47.5%
Module 1 Code: 200
Module 1 Title: Data Mining
Module 1 Mark: 40
Module 2 Code: 201
Module 2 Title: Data Analytics
Module 2 Mark: 35
Module 3 Code: 202
Module 3 Title: Statistical Inference
Module 3 Mark: 55
```

Figure 50 - Exercise 5 output

Exercise 6

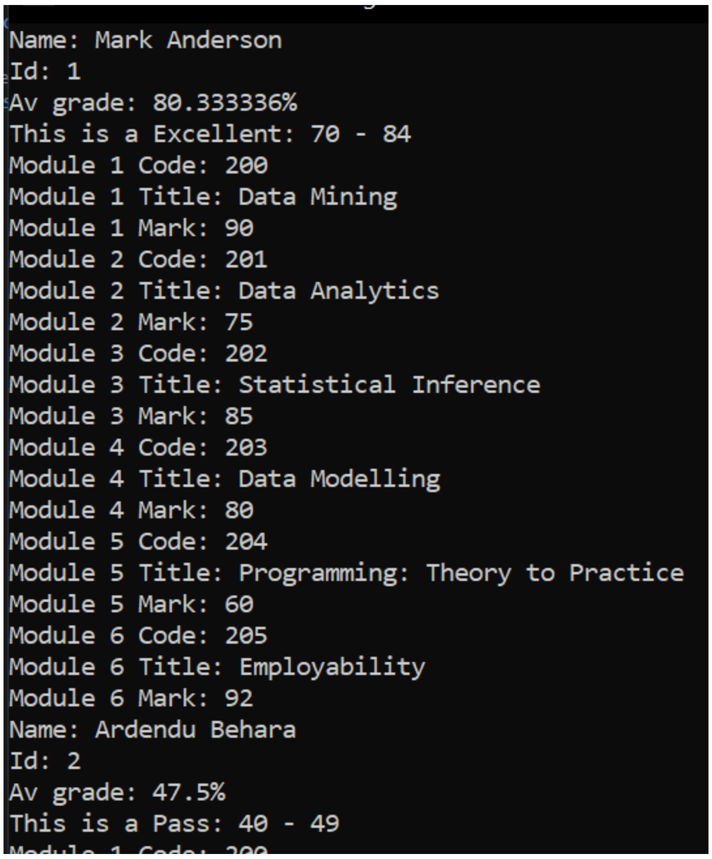
Exercise 6 was an extension of exercise 5. Here the exercise needed the average grade to be given a grade such as fail, pass, very good. The problem was broken down as follows:

- Create a new method which accepts the average grade as the parameter.
- Create a for loop which loops twice once for each student.
- Use if else statements to create a range like week 2 problem 6, does not need to be nested as only looking for a range which can be done using the AND operator. This is to ensure the value is in between the range for the required grade.
- Return grade.
- Print each student with its own data.
- Ensure the year total grade is added to relevant sections, student_data struct, as parameter and in the populate struct method, including when called from main.

- Print all student data.

The expected output should be the same as exercise 4, apart from a new output which would be the year total grade.

As predicted the output was correct the first time around this can be viewed in Figure 51 - Exercise 6 output. No additional research was needed as knowledge for the task has already been acquired form previous research from the previous exercises.



```
Name: Mark Anderson
Id: 1
Av grade: 80.333336%
This is a Excellent: 70 - 84
Module 1 Code: 200
Module 1 Title: Data Mining
Module 1 Mark: 90
Module 2 Code: 201
Module 2 Title: Data Analytics
Module 2 Mark: 75
Module 3 Code: 202
Module 3 Title: Statistical Inference
Module 3 Mark: 85
Module 4 Code: 203
Module 4 Title: Data Modelling
Module 4 Mark: 80
Module 5 Code: 204
Module 5 Title: Programming: Theory to Practice
Module 5 Mark: 60
Module 6 Code: 205
Module 6 Title: Employability
Module 6 Mark: 92
Name: Ardendu Behara
Id: 2
Av grade: 47.5%
This is a Pass: 40 - 49
```

Figure 51 - Exercise 6 output

Blog – Week 3 (Retrospectively entered)

During week 3 structures and arrays were taught in lectures, these helped form the basis to complete this weeks work. Week 3 was one of the hardest weeks for me mentally. Even though the code was broken down I did not pay too much attention to my plan and often drifted away from it slightly causing silly errors which caused panic which in turn cause procrastination from fear of never being able to finish week 3 as I was attempting to rush through it when I should have taken my time.

Reflection – Week 3

The group of exercises from this week were a step up again, especially exercise 4. Week 3 explored new techniques with jagged arrays and structs, jagged arrays were taught in java last academic year. As that was learnt last academic year the problems occurred with declaring data types and figuring out how to use jagged arrays should have not occurred, this is something that needs to be remembered well as it is something which was already gone over. Handling exception techniques were used as the exercises were more complicated and fear for no reason took over ensuring the exercises took a while to complete. This group of exercises took a couple of weeks to complete thus allowing less time for the next set of exercises, more time should have been allocated to completing the exercises sooner. Time management needs to be improved to ensure all techniques were learnt within the week, to prepare and not hinder the progress for the next set of exercises.

Bibliograph – Week 3

DEITEL, M, H. and DEITEL, J. 2005. *C# for Programmers* [ebook]. Available from: https://www.google.co.uk/books/edition/C_for_Programmers/euV7e2f-RzsC?hl=en&gbpv=0 [Accessed 1 November 2021].

DINGLE, S., 2017. *C# for beginners – 14. Structs* [online video]. Available from: https://www.youtube.com/watch?v=1UV1Pd20akM [Accessed 26 October 2021].

OVERIQ.COM., 2020. *Array as Member of Structure in C* [online]. Available from: https://overiq.com/c-programming-101/array-as-member-of-structure-in-c/ [Accessed 29 October 2021].

Week 4

Overview – Week 4

During Week 4, linear data structures were taught and learnt along with one more technique, this included:

- Singly linked lists
- Doubly linked lists
- Recursion

The group of exercises in this week all needed a form of linear linked list to be implemented, this meant additional independent learning as they were not understood well in lectures. In each subsection an explanation to what the task was, what the expected outcome was, the research needed, the thought process around the exercise, the features which were incorporated, the issues which occurred, and the source code created were all presented.

Exercise 1

Exercise 1 was to complete some given code so that a singly linked list would perform certain tasks, these tasks in order were adding new nodes, printing the list; deleting a specific node within the list then printing out the whole list; then inserting a new node after a specified node and printing out the list. The breakdown of this exercise was as follows:

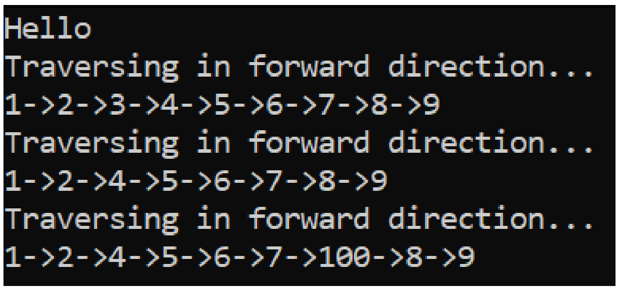
- Figure out how to add new nodes.
- Figure out how to print out the data in a list.

- Figure out how delete a node in the middle of the list.
- Figure out how to insert a node in the middle of the list.

The expected output is the list printed with the required new nodes, the list printed with a deleted node and a list printed with an inserted node.

Extra research and learning were done to ensure the concept along with the code was properly understood, this exercise was attempted with just the pseudocode from the lecture slides which did not work learnt from KC70 (2018), Chastine (2013) and Journey to programming (2019). The concept seemed simple although a couple of days were required to fully understand how to code the data structure.

As shown in Figure 52 - Singly linked list output. The output for this exercise was as expected.



```
Hello
Traversing in forward direction...
1->2->3->4->5->6->7->8->9
Traversing in forward direction...
1->2->4->5->6->7->8->9
Traversing in forward direction...
1->2->4->5->6->7->100->8->9
```

Figure 52 - Singly linked list output

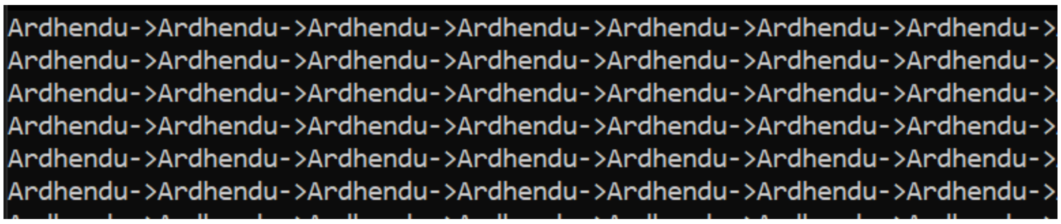
Exercise 2

Exercise 2 was to complete some given code so that a doubly linked list would perform certain tasks, these tasks in order were adding new nodes, printing the list; deleting a specific node within the list then printing out the whole list; then inserting a new node after a specified node and printing out the list. The breakdown of this exercise was as follows:

- Figure out how to add new nodes.
- Figure out how to print out the data in a list going forwards and backwards.
- Figure out how delete a node in the middle of the list.
- Figure out how to insert a node in the middle of the list.

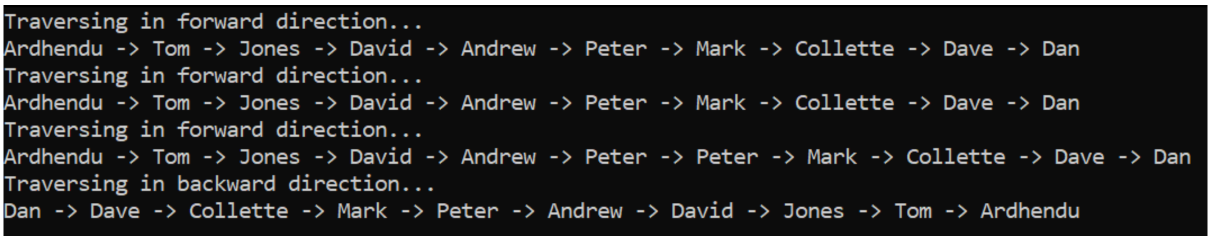
The expected output is the list printed with the required new nodes, the list printed with a deleted node and a list printed with an inserted node but printed backwards in this case.

The output was as expected for all bar the last exercise, when trying to traverse the list backwards guess work was used instead of logic to begin with and then the attempt to switch the head and tail node were made, this produced, and error seen in Figure 53 - Doubly linked list error. This was due to what had been discovered in KC70 (2019), Blue tree code (2010), and My code school (2019), after a few days of struggle with this the lecturer was approached which explained the tail node could be traversed backwards with tail.prev which was much easier and worked. This somehow altered the code and made one of the other tasks to not work, had the lecturer been approached sooner this could have allowed more time to ensure exercise 2 was properly completed. The final out put can be seen in Figure 54 - Doubly Linked list output.



```
Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->
Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->
Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->
Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->
Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->
Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->Ardhendu->
```

Figure 53 - Doubly linked list error



```
Traversing in forward direction...
Ardhendu -> Tom -> Jones -> David -> Andrew -> Peter -> Mark -> Collette -> Dave -> Dan
Traversing in forward direction...
Ardhendu -> Tom -> Jones -> David -> Andrew -> Peter -> Mark -> Collette -> Dave -> Dan
Traversing in forward direction...
Ardhendu -> Tom -> Jones -> David -> Andrew -> Peter -> Peter -> Mark -> Collette -> Dave -> Dan
Traversing in backward direction...
Dan -> Dave -> Collette -> Mark -> Peter -> Andrew -> David -> Jones -> Tom -> Ardhendu
```

Figure 54 - Doubly Linked list output

Exercise 3

This exercise asked for the structure given in Example 1 exercise 3 to be converted into a singly linked list. This exercise asked to ensure the InsertNode, DeleteNextNode and TraverseList method were amended from Week 4 Exercise 1 to insert and delete the requested nodes along with finally printing out the whole list with the TraverseList method.

Due to lake of time management this exercise was not completed, this is something that needs to be worked on as it would affect future work.

Blog – Week 4 (Retrospectively entered)

During week 4 we learnt about linear data structures and recursion, the concepts were explained but the code not really shown (it was shown in pseudocode), this meant independent research needed to be done along with understanding the pseudocode give to implement it into the given tasks. This was a difficult concept to wrap my head around especially with so little time thanks to my time management for week 3. Week 4 was by far the hardest week, this was because almost all of it had to be learnt by myself, the doubly linked list was for sure a harder concept to wrap my head around for traversing backwards with multiple videos shown which explained you had to reassign tail as head when there was a much simpler way to do it. I should have really managed my time for week 3 better so I would have had more time to complete exercise 3. I also should have not been afraid to ask the lecturer to explain the concept of traversing the list backwards so it would have not taken multiple days to figure out and asked the lecturer sooner. The singly linked list while not hard to understand the theory behind it, it was harder than initially thought to wrap my head around how to code. After a good night sleep, I allowed the concept and pseudocode to be conceptualised in my head so I could think of how to the singly linked list. This helped as I was being persistent and upset, I could not be able to complete exercise 1 when the concept seemed so simple, I need to learn to know when to take a breather so I may approach the task in a calm and logical manner as I had done with the weeks before which worked. As time was short due to lack of time management the report started off well but feels as though is a little weaker in the second half, I really regret this as last year I managed to time my assignments well and will not allow myself to be in the position ever again as the stress is not worth it. The main take away are time management and to not get frustrated and panic and let fear creep in, otherwise I will never understand what is wrong. As discovered taking my time (but not too much time) along with following the broken-down plan is essential.

Reflection – Week 4

The linear data structures took some more time to understand, as week 3 was not handled appropriately, week 4 did not have enough time to be completed. As week 4's concepts were harder and needed more research, more time was needed which there was little off, due to poor time management which needed to be improved. This needs to drastically improve as all exercises should have been completed. Due to having little time and not understanding. Help was needed but not seeked out until the last moment, this needs to change as there is no shame in asking for help to understand a concept as long as it is genuinely attempted first, due to this paired with week 3's poor time management week 4 was not completed when it could have been.

Bibliograph – Week 4

BLUETREECODE., 2010. *Doubly Linked List | Insert, Delete, Complexity analysis* [online video]. Available from: https://www.youtube.com/watch?v=ZlNKNSz88Nk [Accessed 15 November 2021].

CHASTINE, J., 2013. *Tutorial 18.2 – Developing a Linked List in C#* [online video]. Available from: https://www.youtube.com/watch?v=3svB0kM6f10 [Accessed 10 November 2021].

JOURNEYTOPROGRAMMING., 2019. *C# LinkedList Tutorial How to Use A Linked List* [online video]. Available from: https://www.youtube.com/watch?v=0AO7OwNzd2Y [Accessed 10 November 2021].

KC70., 2019. *Linked List implemented in C# as a doubly Linked List* [online video]. Available from: https://www.youtube.com/watch?v=GcC5kW9tyOQ [Accessed 14 November 2021].

KC70., 2018. *Singly Linked List implemented in C#* [online video]. Available from: https://www.youtube.com/watch?v=iCkhD5SYTQY [Accessed 11 November 2021].

MYCODESCHOOL., 2013. *Data structures: Introduction to Doubly Linked List* [online vide]. Available from: https://www.youtube.com/watch?v=JdQeNxWCguQ [Accessed 17 November 2021].