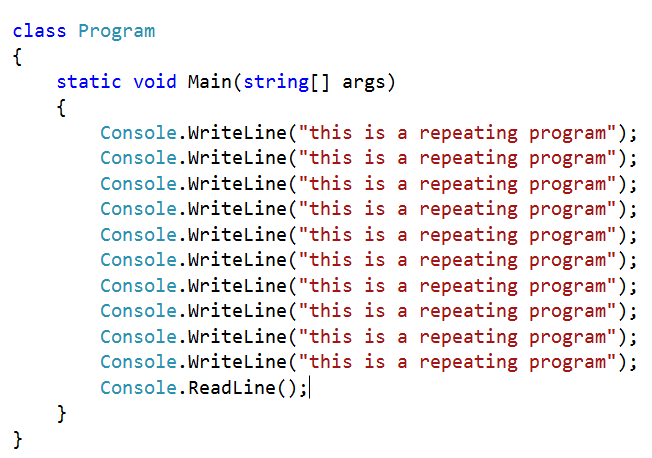
# Chapter 4 – Looping

## The need for repetition in programming

Very often there is a need in programs to carry out repetitive tasks or actions.

The simple example below of repeating lines of identical text shows that without repetition statements in programming the amount of code we might have to write would be enormous and it would take away the flexibility of conditional repetition.

Loops of various kinds allow us to repeat sections of code without the need to write identical lines of code repetitively.



## Types of loop: the FOR loop

The FOR loop is known as a fixed or unconditional loop. It will repeat the contained code statements for the specified number of times. It is composed of a number of features.

**for (*init*; *condition*; *increment*)**

**init** – the loop counter. It is a variable that holds the number of iterations (loops) that the following statements (in the brackets) will make.

**condition** –evaluates the value of the counter, and the body of the loop is executed if the condition is true.

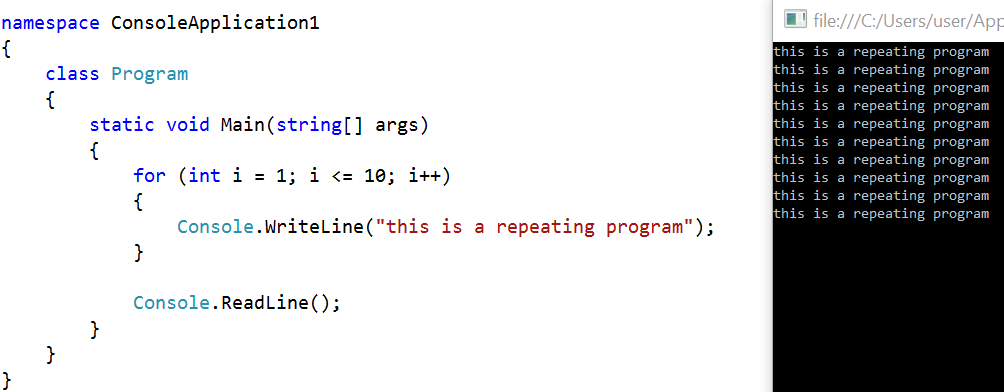
**increment**–updates the counter after the loop executes; also called the loop control variable.

**{**

***Code to be repeated goes here***

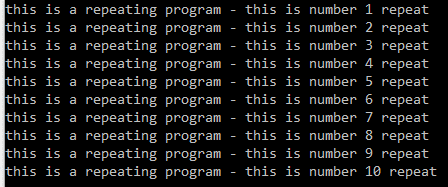
**}**

**A worked example**

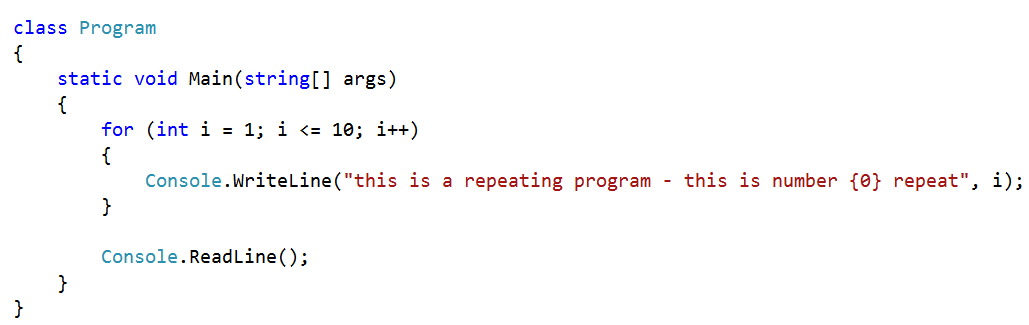


The program above will print out ‘this is a repeating program’ 10 times.

* **init (**the loop counter in this example) is declared as an integer variable and will start at 1.
* *Condition:* The code contained in the curly brackets will repeat as long as the init variable is less than or equal to 10. This means it will repeat 10 times.
* *Increment:* this is set so that the loop counter increments by 1 after each repetition.

**Using the loop counter**

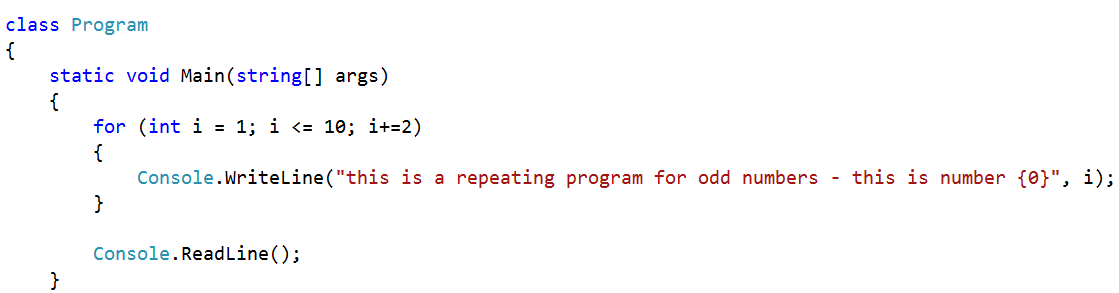
In this program the loop counter is used to indicate in the console window the number of the repetition.



**Incrementing in larger steps**

In this program, the loop counter starts at 1 and goes up in steps of two so that if the loop condition is less than or equal to 10 then the loop counter will go from 1 to 3 to 5 to 7 and stop after 9.

Therefore, this loop will execute five times. It is also possible for the loop counter to start at a higher number and decrement instead of increment.





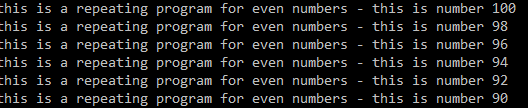


**{ }**

Coding Task: *Program 4.1 – Ordering numbers*

1. Write a program to output all even numbers between 100 and 1.

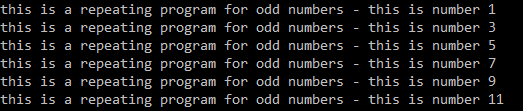
This is some of the output you need to produce to the console window.



…

1. Amend your program so that it outputs all odd numbers from 1 to 99.

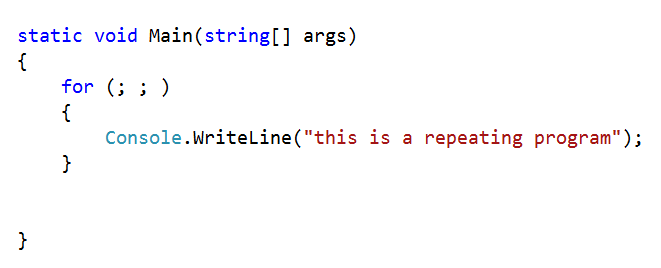
This is some of the output you need to produce to the console window.



…

## Infinite loops

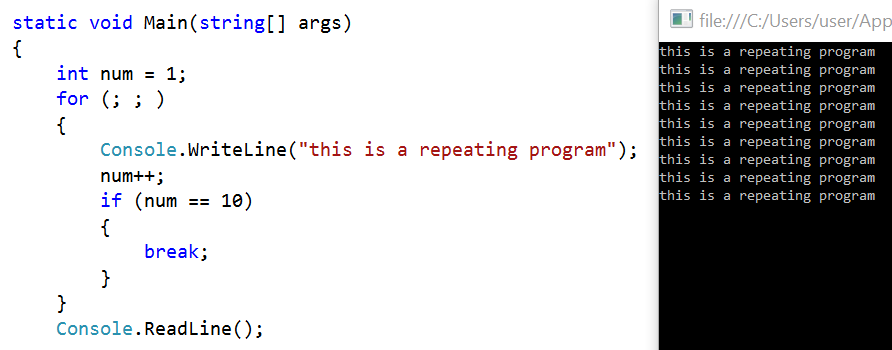
The code below will produce an infinite loop, i.e. a loop that continues to execute indefinitely until the program is closed. Infinite loops are occasionally useful.





**Infinite loops and the break statement**

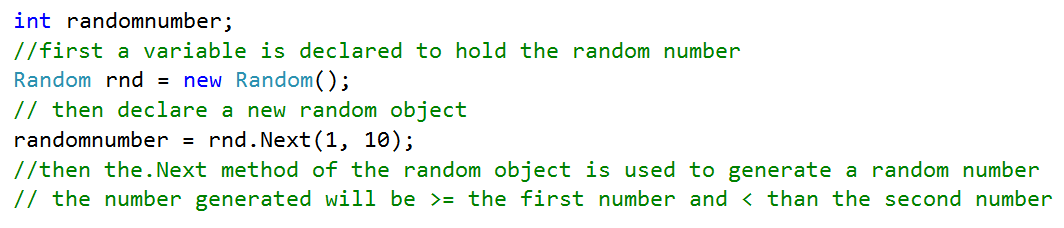
It is possible to break out of an infinite loop using a **break** statement. In the program below, the looping will stop when num is equal to 10. However, in this case it would be easier to set this as a condition at the beginning of the loop rather than adding an if/break statement inside an infinite loop.



## Generating random numbers

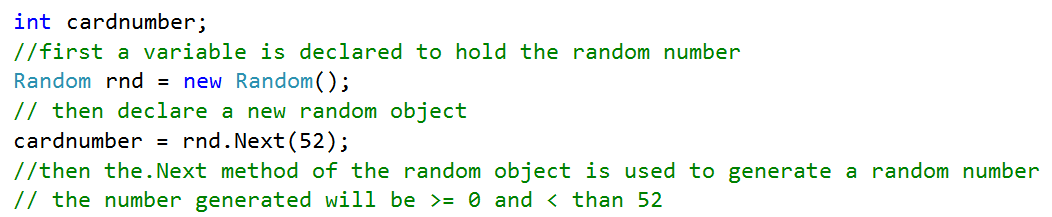
It is common in programming to generate random numbers, and C# provides a RANDOM object to use when we need to create a random number. First we declare a new RANDOM object, and then we specify the range that the number should be generated from using the **.Next** method.

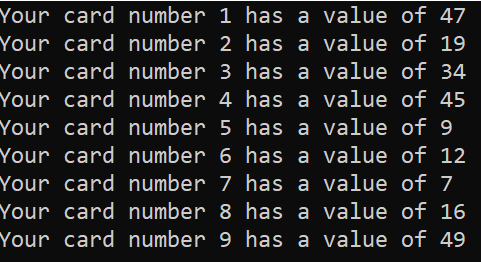
**Example 1 – generating a random number between 1 and 10**



**Example 2 – generating a random number between 0 and 52**

In this case there was only a need to place the maximum value in the brackets as the starting value was 0.

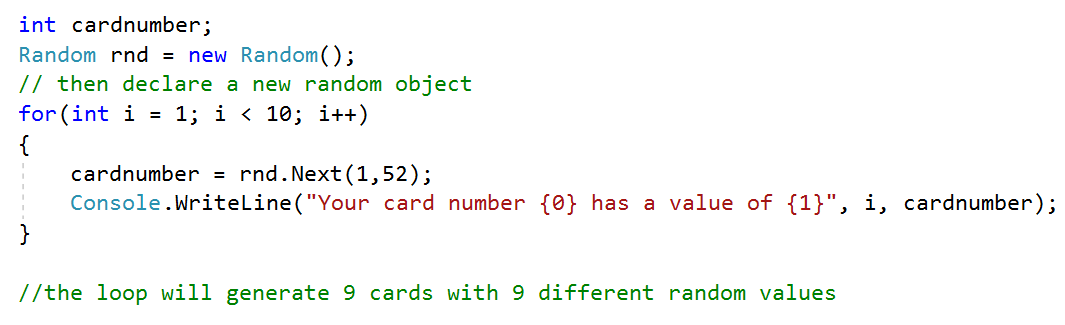




**Using a loop to generate multiple random numbers**



When there is a need to generate multiple random numbers, the **.Next** method is placed inside the loop whereas the declaration of the random object is placed outside the loop.





**{ }**

Coding Task

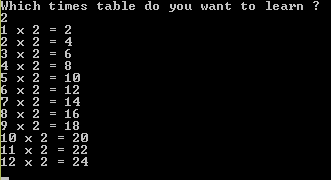
Find out what happens if you place the Random object declaration statement   
(Random rnd = new Random()) inside the loop.

Coding Task: *Program 4.2 – Times Table*



**{ }**

Write a program that will output the times table for whatever number the user inputs (up to 12 ×).   
The output to the console window should look something like the output window below.

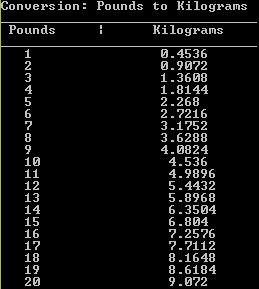




**{ }**

Coding Task: *Program 4.3 – Weight Conversion*

Write a program that displays a conversion table for pounds to kilograms, ranging from 1 pound to   
20 pounds [1 pound = 0.4536 kg]. The output to the console window should look something like the output window below.



## The WHILE loop

This loop is a conditional loop. This means that it will terminate only when a certain specified condition evaluates to false. This condition is checked before the loop statements are allowed to execute. The loop will continue to iterate (repeat) while the outcome of the condition is true. The syntax and layout for a conditional loop are shown below.

**while** (Condition)

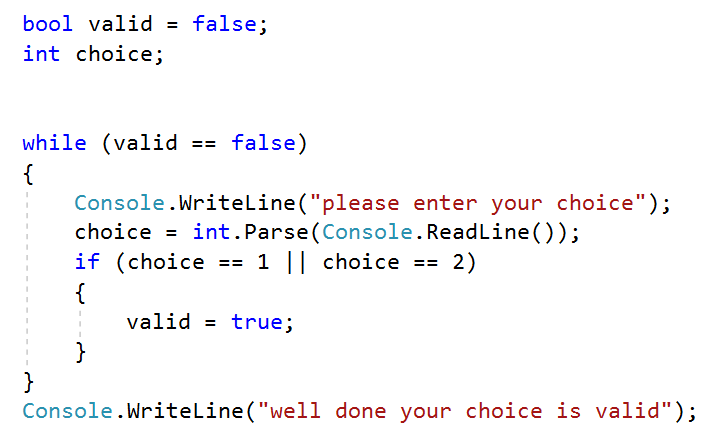
{

*Code to be repeated goes here*

}

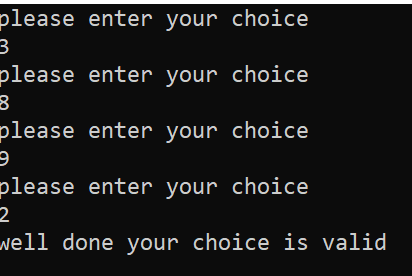
**A worked example**

The program below is part of a larger program, but this snippet is checking to see that the choice entered on a menu system is either 1 or 2 (as these are the only valid choices). The program continues to ask for a choice until it receives a valid choice.



This condition will be checked before the statements inside the loop execute. If the **condition evaluates to true** (i.e. valid is false), then the statements inside the loop block execute. If the **condition evaluates to false**, then the statements inside the loop block will not be executed and the program jumps to the next line of code after the loop block

As can be seen from the output from this program (below), the loop only stops executing when the user enters a valid choice. In this case, the loop executes four times.



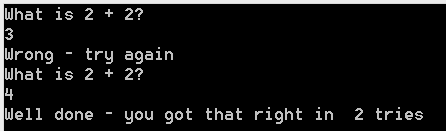


**{ }**

Coding Task: *Program 4.4: The Question Program*

Write a program that asks the user a question, checks the answer and outputs a statement that says how many tries the user took to get the answer right.

You can ask whatever question you prefer but the output screen below shows an example of an addition question.



## The DO WHILE loop

This loop is similar to the WHILE loop except the code statements inside the loop block will always execute at least once. This is because the condition is tested at the end of the loop rather than at the beginning. The syntax and layout for a **do while** conditional loop are shown below.

**do**

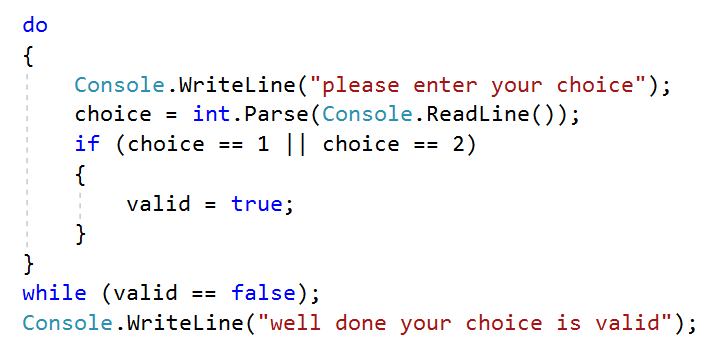
{

*Code to be repeated goes here*

}

**while** (Condition);

**A worked example**



The **do** keyword is placed at the beginning of the loop, and the **while** keyword and the condition are placed at the end of the loop.

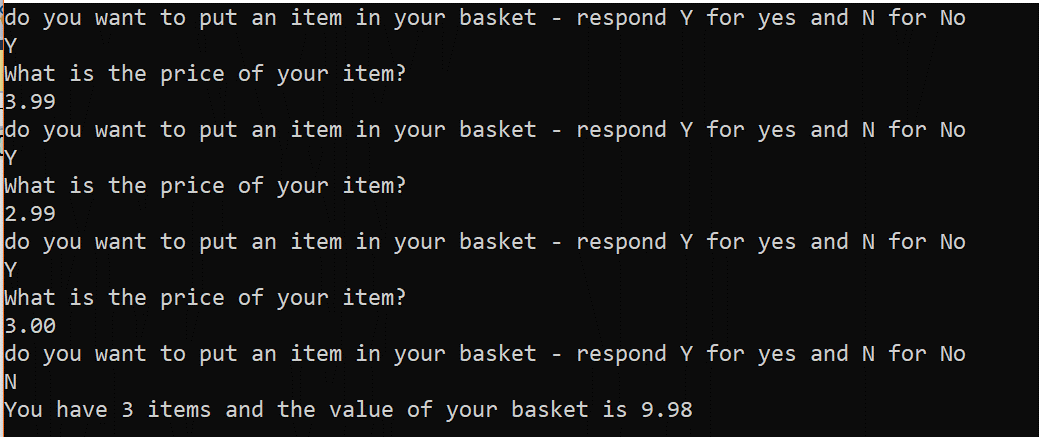


**{ }**

Coding Task: *Program 4.5 – The Basket Program*

Write a program that asks a user whether they want to add items to their basket, and then inputs the   
price of the item if they say yes. When they have finished adding items the program should output how many items are in the basket and the total price for those items.

The image below shows how the program should look in the output window.



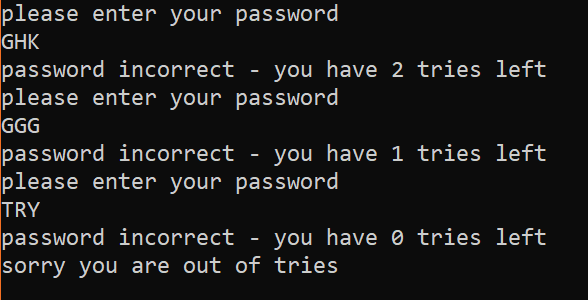


**{ }**

Coding Task: *Program 4.6 – The Password Program*

Prompt the user to enter their password; you then have to check their password against the one in the system. If their password is correct, display ‘You have successfully logged in’. Otherwise, continue to prompt for a password.

**For an extra challenge**,if the user enters an incorrect password three times, display the message  
‘Sorry you are out of tries’ and quit.



## Chapter 4 – Consolidation Tasks

### Looping consolidation – fixed loops

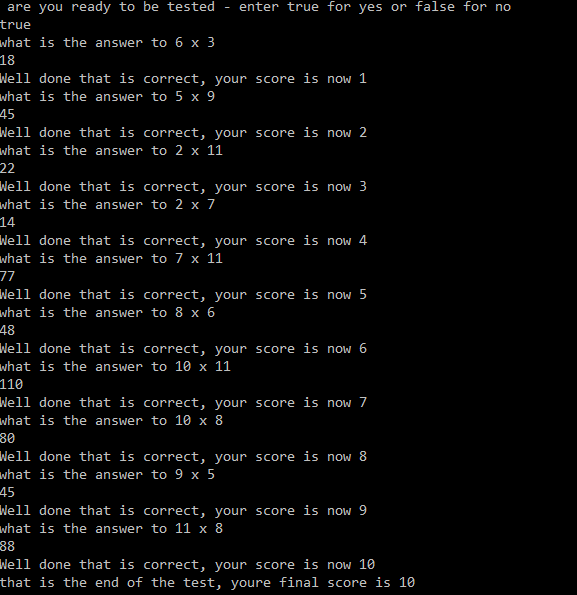
Program 4.7

Write a maths program that outputs random number multiplication questions to the console window and then checks the given answers – keeping a score until 10 questions have been answered. Users get one point for every correctly answered question.

At the beginning of the program, the user should be asked whether they are ready to be tested – the program will output questions, check answers and keep score only if they reply true to the question. If the test is not played, then an alternative message, “sorry you didn’t want to play”, or a similar message, should be output.

At the end of the test a message should be output saying that the test is finished and outputting the final score.

The output from the console window should look something like the output window below.



**Evidence required:**

1. An **annotated code listing** for the program above.
2. **Screenshots of testing** carried out where:
   1. The user enters ‘true’ and then answers all the multiplication questions correctly
   2. The user enters ‘true’ and then answers 8 out of 10 multiplication questions correctly
   3. The user enters ‘false’ and appropriate exit message appears
3. Explain what changes you would need to make to the program for it to work for any operation (addition, multiplication, subtraction, division, modulus).

### Looping consolidation – choosing loops

Here you are expected to use the most appropriate loops to solve three short programs.   
This may involve using a for loop, a while loop or a do while loop.

Program 4.8

A credit card has a balance of £50 owing on it. The bank charges 2% interest every month. Write a program that works out how many months it would take for the balance plus interest charges to reach £100.

**Evidence to submit for this task program:**

1. An annotated code listing for the above program.
2. A screenshot of the output window after the program has been run.
3. An explanation of why the loop you chose was appropriate for this program.

Program 4.9

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Write a program to find the sum of all the multiples of 3 and 5 below 1000.

**Evidence to submit for this task program:**

1. An annotated code listing for the above program.
2. A screenshot of the output window after the program has been run.
3. An explanation of why the loop you chose was appropriate for this program.

Program 4.10

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

Write a program to find the forty-second Fibonacci number.

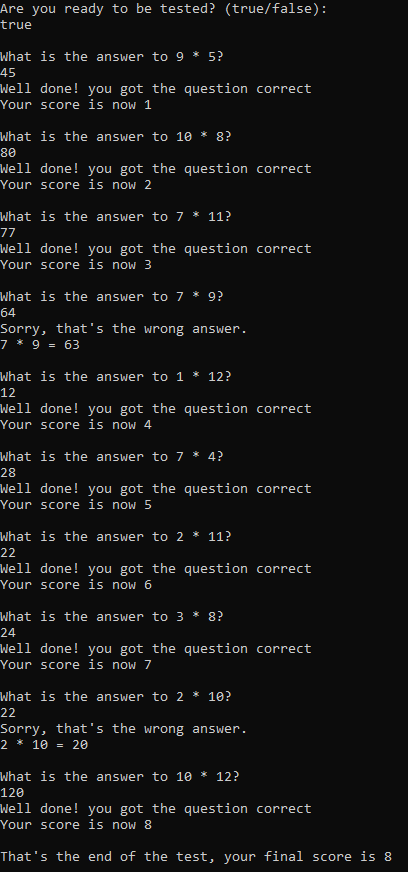
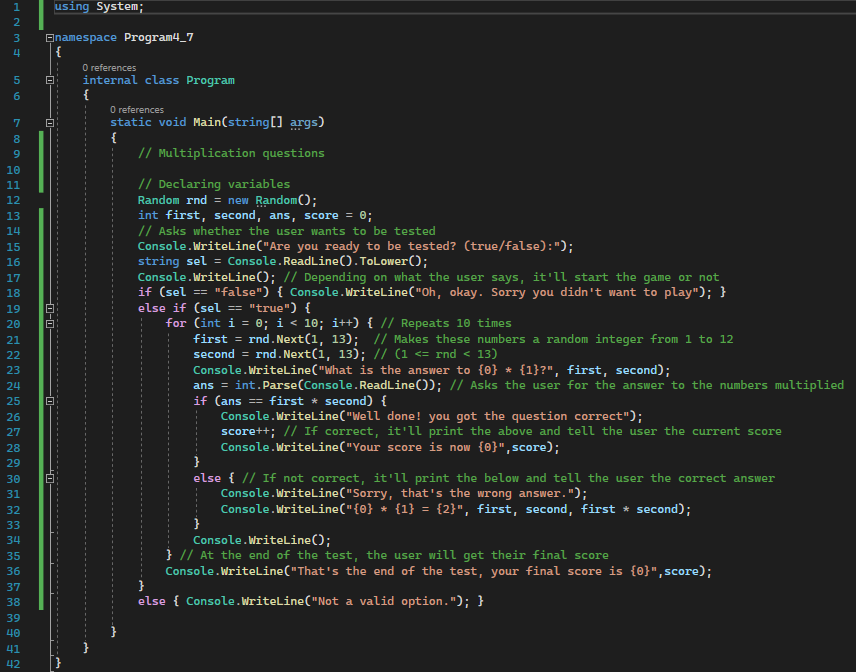
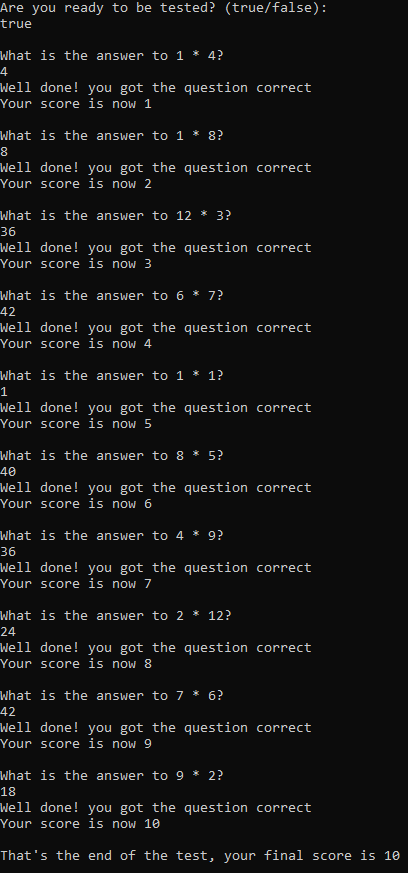
**Evidence to submit for this task program:**

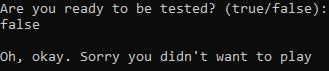
1. An annotated code listing for the above program.
2. A screenshot of the output window after the program has been run.
3. An explanation of why the loop you chose was appropriate for this program.

## Answers

Programs 4.1 to 4.6 can be sent with the sheet

**Program 4.7**





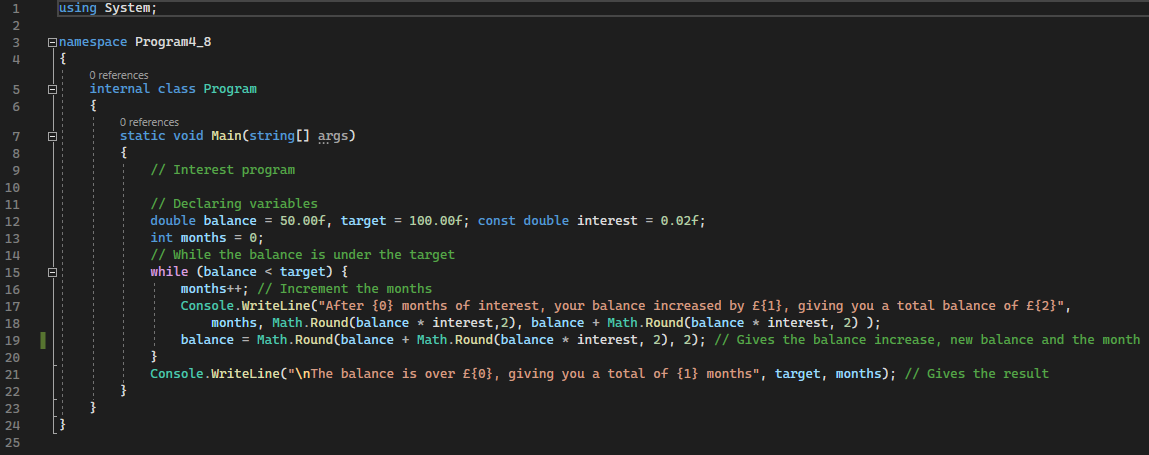
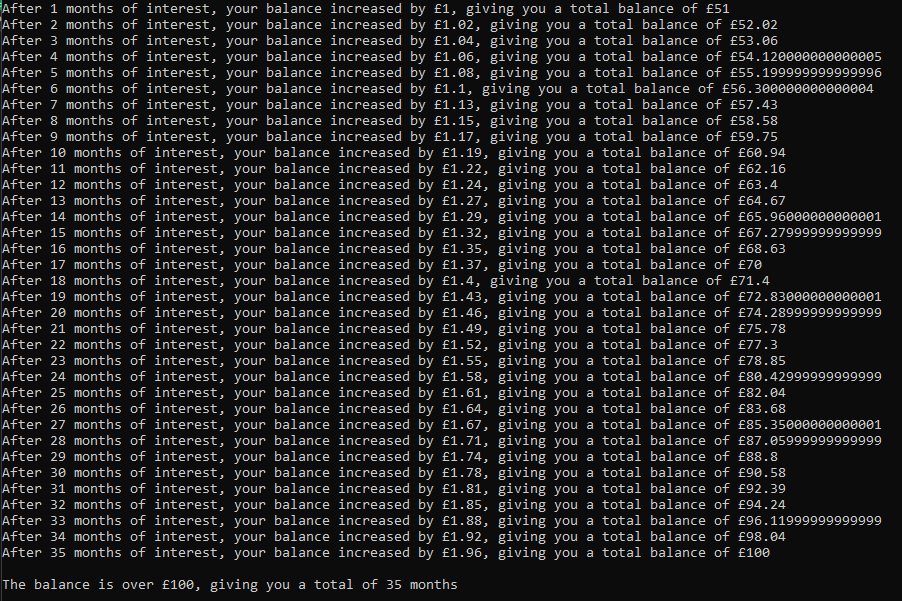
First, you’d have to add a random number and

for each number have a different operation

corresponds for it. Then for each one you’d have

to switch case to the operation you’d want

**Program 4.8**

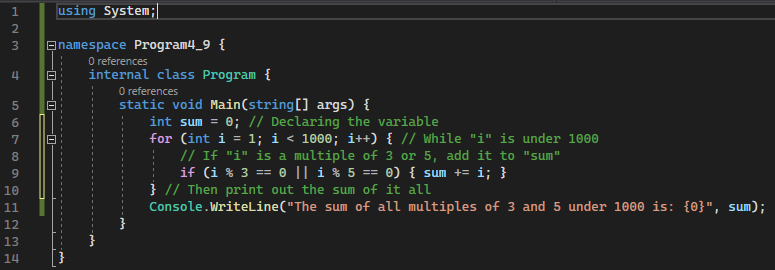


I used a while loop because I didn’t know how many

times I would have to loop (how many months it takes)

until the limit was reached. This disproves using the for loop

**Program 4.9**



I used a for loop because I knew how many

numbers I wanted to check (all up to 1000).

I could use a while loop with this, but that

would require another variable



**Program 4.10**

I used a for loop this time, this is because

I wanted to try and add enough

elements to the array as to have 42 in

Text

Description automatically generatedthe array.