Notes and guidance: C#

The C# code is described in this resource to help students prepare for their AQA GCSE Computer Science exam (8525/1).

We will use this consistent style of C# code in all assessment material. This will ensure that, with enough preparation, students will understand the syntax of the code used in assessments. Students do not have to use this style of code in their own work or written assessments, although they are free to do so. The only direction to students when answering questions or describing algorithms written in code is that their code is clear, consistent and unambiguous.

This resource may be updated as required and the latest version will always be available on our website. It is not confidential and can be freely shared with students.

General Syntax

- Code is shown in `this font`.
- `DataType` means a datatype such as `string`, `int`, `char` or `single`.
- `Exp` means any expression.
- `IntExp`, `RealExp`, `BoolExp`, `StringExp`, `CharExp` and `ArrayExp` mean any expression which can be evaluated to an integer, real, Boolean (`false` or `true`), string, character or array respectively.

Indentation

C# code will use indentation to indicate the range of statements controlled by iteration and selection statements (as well as declarations for subroutines). Indentation will be shown with three spaces per indentation level, although if doing so makes lines too long for the page, this may be reduced to two spaces. Questions will show indentation guides (vertical lines) within the answer space. Students should be encouraged to use these to explicitly show their indentation.

Comments

Single line comments	`// A comment`	
Multi-line comments	`// A comment` `// Another comment`	

String and character literals

String literals will be delimited using the `"` (double quote) character, while character literals will be delimited using the `'` (single quote) character.

Variables and constants

Variable names will be written in camel case, eg `numberOfItemsSold`. Camel case is the practice of writing phrases without spaces or punctuation, indicating the separation of words with a single capitalised letter and the first word starting with either case.

Constant names will be written in upper case, using an underscore to indicate a break between words, eg `ACCELERATION_DUE_TO_GRAVITY`.

Although C# considers names such as `message` and `Message` to refer to distinct variables, questions will not include any variables whose names differ solely in case.

Questions will use meaningful variable names wherever possible, eg `quantityInStock`, `quantity` or `qty` rather than just `n` to hold the quantity of an item in stock. For layout and/or lack of context reasons, this resource may not always follow this advice. This rule may not be followed for common idioms, eg using `i` as a loop index or for exam-related reasons.

Variable declaration	`DataType Identifier;`
	
	or
	
	`DataType Identifier = Exp;`
		`int aNumber;`
		`int anotherNumber;`
		`string theString;`
		
		`string message = "Invalid number";`
		`int totalNumberOfItems = 0;`

Variable assignment	`Identifier = Exp;`	`aNumber = 3;`
		`anotherNumber = aNumber + 1;`
		`theString = "Hello";`

Constant assignment	`const DataType IDENTIFIER = Exp;`	`const int CLASS_SIZE = 23;` `const double PI = 3.141;`

Arithmetic operations

Standard arithmetic operations	`+`
	`-`
	`*`
	`/`
		Used in the normal way with brackets to indicate precedence where needed. So, `a + b * c` would multiply `b` and `c` together and then add the result to `a`, whereas `(a + b) * c` would add `a` and `b` together and then multiply the result by `c`.
		
		Brackets may be used to indicate precedence even where not strictly necessary, eg testing for divisibility by 3 could be written as `(n % 3) == 0` rather than `n % 3 == 0`

Integer division
 	
If the two operands to divide are integers (literals or variables) then `/` performs integer division.
eg `34 / 7 = 4`. If one or both of the operands is single or double then normal division is carried out, eg `34 / 7.0 = 4.8571428`
	`IntExp / IntExp`	`9 / 5` evaluates to `1`
		`5 / 2` evaluates to `2`
		`8 / 4` evaluates to `2`

Modulus operator	`IntExp % IntExp`	`9 % 5` evaluates to `4`
		`5 % 2` evaluates to `1`
		`8 % 4` evaluates to `0`

Relational operators for types that can be clearly ordered (numbers, characters)

Less than	`Exp < Exp`	`4 < 6` `'A' < 'B'`
Greater than	`Exp > Exp`	`4.1 > 4.0`
Equal to	`Exp = Exp`	`3 == 3`
Not equal to	`Exp != Exp`	`qty != 7`
Less than or equal to	`Exp <= Exp`	`3 <= 4` `4 <= 4`
Greater than or equal to	`Exp >= Exp`	`4 >= 3` `4.5 >= 4.5`

Relational operators for strings

Currently C# does not allow strings to be compared using < and > so the `CompareTo` method is used, returning `-1` if the first string is less than the second, `0` if the first string is equal to the second (although strings can also be compared for equality using == and `!=`), and `+1` if the first string is greater than the second.

Less than	`StringExp.CompareTo(StringExp)`	`"adam".CompareTo("adele") == -1`
Greater than	`StringExp > StringExp`	`"adele".CompareTo("adam") == 1`
Equal to	`StringExp == StringExp`	`"ada" == "ada"`
Not equal to	`StringExp != StringExp`	`"adam" != "ada"`

Boolean operations

Logical AND	`BoolExp && BoolExp`	`(3 == 3) && (3 <= 4)`
Logical OR	`BoolExp || BoolExp`	`(x < 1) || (x > 9)`
Logical NOT	`! BoolExp`	`! (a < b)`

Block structure

Braces { and } will always be included around blocks even when the block consists of only a single statement.

The position of the opening brace { may be at the end of the introducing statement (while, do, if, etc) or on the next line, depending on space and other requirements.

Indefinite (condition controlled) iteration

WHILE (while the Boolean expression is true, repeat the statements). If the Boolean expression is false the first time the while statement is reached then the indented statements are never executed.	`while (BoolExp)`
	`{`
	`  // indented statements here`
	`}`
		`int a = 1;`
		`while (a < 4)`
		`{`
		`  Console.WriteLine(a);`
		`  a = a + 1;`
		`}`
		`// outputs 1, 2, 3`
		
		`a = 5;`
		`while (a < 4)`
		`{`
		`  Console.WriteLine(a);`
		`  a = a + 1;`
		`}`
		`// outputs nothing since a < 4 is`
		`// false the first time while is met`

REPEAT-UNTIL (repeat the statements until the Boolean expression is True).	`do`
	`{`
	`  // indented statements here`
	`} while BoolExp;`
		`int a = 1;`
		`string carryOn;`
		
		`do`
		`{`
		`  Console.WriteLine(a);`
		`  a = a + 1;`
		`  Console.Write("Continue? ");`
		`  carryOn = Console.ReadLine();`
		`} while (carryOn != "N");`
		`// outputs 1, … until 'N' is entered`

Definite (count controlled) iteration

<table>
<tr>
<td>FOR
(repeat the statements the number of times indicated by the range function, each time giving the loop variable (<code>Identifier</code>) the value of the next value/number in the range).</td>
<td>

```
for (DataType Identifier = Exp; BoolExp; Change)
{
   // indented statements here
}

// Change changes the value of Identifier
// at the end of each iteration of the loop.
// Common change statements for integers are
// Identifier++
// to increment the value of Identifier by 1, and
// Identifier--
// to decrement the value of Identifier by 1.




// To increase or decrease an integer by something
// other than 1 use
// Identifier += Exp
// to increase the value of Identifier by the
// value of Exp and
// Identifier -= Exp
// to decrease the value of Identifier by the
// value of Exp
```

</td>
<td>

```
for (int i = 0; i < 7; i++)
{
   Console.WriteLine(i);
}
// outputs 0, 1, 2, 3, 4, 5, 6
// 7 values are output, the last
// one being 6


for (int i = 1; i < 7; i++)
{
   Console.WriteLine(i);
}

// outputs 1, 2, 3, 4, 5, 6

for (int i = 1; i < 7; i += 2)
{
   Console.WriteLine(i);
}
// outputs 1, 3, 5

for (int i = 7; i > 1; i -= 2)
{
      Console.WriteLine(i);
}
// outputs 7, 5, 3
```

</td>
</tr>
</table>

Definite (count controlled) iteration (continued)

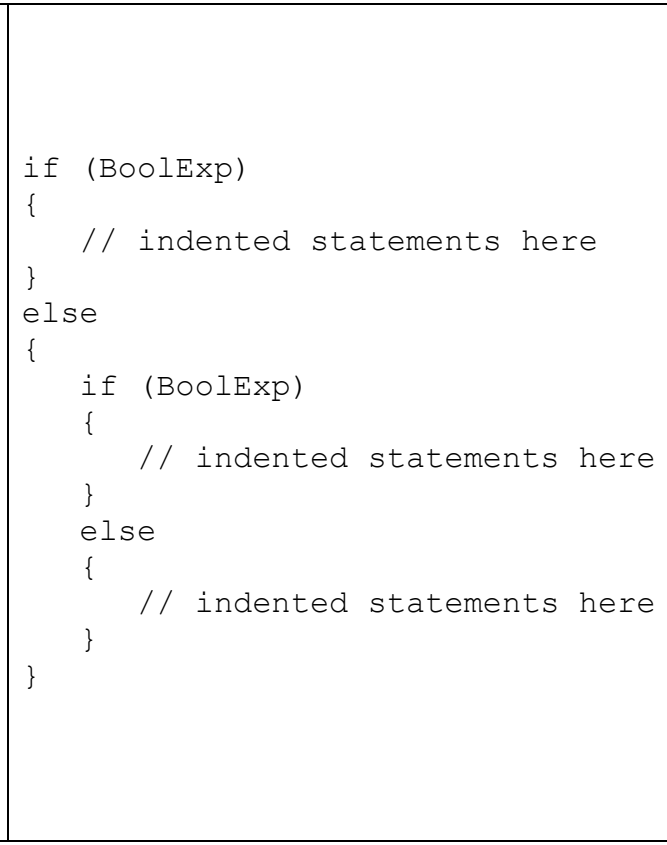
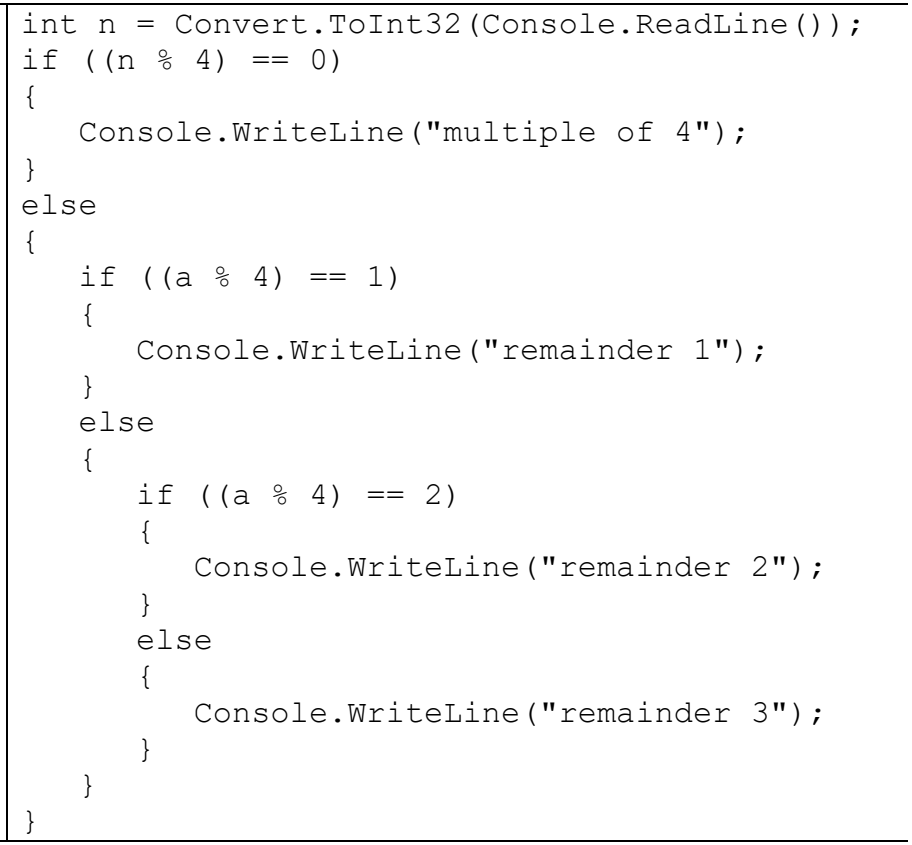
FOR
(repeat the statements the number of times that there are characters in a string, each time giving `Identifier` the value of the next character in the string).
	`foreach (char Identifier in StringExp)`
	`{`
	`    // indented statements here`
	`}`
		`string message = "Hello";`
		
		`int length = 0;`
		`foreach (char c in message)`
		`{`
		`    length = length + 1;`
		`}`
		`Console.WriteLine(length);`
		`// calculate the number of`
		`// characters in message (5)`
		`// and output it`
		
		`string reversed = "";`
		`foreach (char c in message)`
		`{`
		`    reversed = c + reversed;`
		`}`
		`Console.WriteLine(reversed);`
		`// reversed is set to the`
		`// reverse of message and`
		`// output ("olleH")`

Selection

IF-THEN-ENDIF (execute the statements only if the Boolean expression is `true`).	`if (BoolExp)`
	`{`
	`  // indented statements here`
	`}`
		`int n = Convert.ToInt32(Console.ReadLine());`
		`if ((n % 2) == 0)`
		`{`
		`  Console.WriteLine("even");`
		`}`

IF-THEN-ELSE-ENDIF (execute the statements following the THEN if the Boolean expression is `true`, otherwise execute the statements following the ELSE).	`if (BoolExp)`
	`{`
	`  // indented statements here`
	`}`
	`else`
	`{`
	`  // indented statements here`
	`}`
		`int n = Convert.ToInt32(Console.ReadLine());`
		`if ((n % 2) == 0)`
		`{`
		`  Console.WriteLine("even");`
		`}`
		`else`
		`{`
		`  Console.WriteLine("odd");`
		`}`

Selection (continued)

NESTED IF-THEN-ELSE ENDIF (use nested versions of the above to create more complex conditions).
 	
Note that IF statements can be nested inside the THEN part, the ELSE part or both.
	
	`if (BoolExp)`
	`{`
	`  // indented statements here`
	`}`
	`else`
	`{`
	`  if (BoolExp)`
	`  {`
	`  // indented statements here`
	`  }`
	`  else`
	`  {`
	`  // indented statements here`
	`  }`
	`}`
		
		`int n = Convert.ToInt32(Console.ReadLine());`
		`if ((n % 4) == 0)`
		`{`
		`  Console.WriteLine("multiple of 4");`
		`}`
		`else`
		`{`
		`  if ((a % 4) == 1)`
		`  {`
		`  Console.WriteLine("remainder 1");`
		`  }`
		`  else`
		`  {`
		`  if ((a % 4) == 2)`
		`  {`
		`  Console.WriteLine("remainder 2");`
		`  }`
		`  else`
		`  {`
		`  Console.WriteLine("remainder 3");`
		`  }`
		`  }`
		`}`

Selection (continued)

IF-THEN-ELSE IF ENDIF (removes the need for multiple indentation levels).	
	
	`if (BoolExp)`
	`{`
	`  // indented statements here`
	`}`
	`else if (BoolExp)`
	`{`
	`  // indented statements here`
	`}`
	`// possibly more else ifs`
	`else`
	`{`
	`  // indented statements here`
	`}`
		`int n = Convert.ToInt32(Console.ReadLine());`
		`if ((n % 4) == 0)`
		`{`
		`  Console.WriteLine("multiple of 4");`
		`}`
		`else if ((n % 4) == 1)`
		`{`
		`  Console.WriteLine("remainder 1");`
		`}`
		`else if ((n % 4) == 2)`
		`{`
		`  Console.WriteLine("remainder 2");`
		`}`
		`else`
		`{`
		`  Console.WriteLine("remainder 3");`
		`}`

Arrays

Declaration	`DataType[] Identifier = new DataType[IntExp];`
	
	`DataType[] Identifier = {Exp, … ,Exp};`
		`int[] array1 = new int[5];`
		`// declares an array containing 5 integers`
		
		`int[] primes = {2, 3, 5, 7, 11, 13};`

Assignment (one-dimensional array)	`Identifier = new DataType[] {Exp, … ,Exp};`	`array1 = new int[] {2, 4, 6, 8, 10};`

Accessing an element	`Identifier[IntExp]`	`Console.WriteLine($"The only even prime is {primes[0]}");`
		
		`// outputs "The only even prime is 2"`

Updating an element	`Identifier[IntExp] = Exp;`	`primes[5] = 17;`
		
		`// primes is now {2, 3, 5, 7, 11, 17}`

Assignment (two-dimensional array)	`DataType[,] Identifier[,] = {{Exp, … , Exp}, …, {Exp, …, }};`	`int[,] t = {{1, 2}, {2, 4}, {3, 6}};`

Accessing an element in a two-dimensional array	`Identifier[IntExp, IntExp]`	`Console.WriteLine($"Row 3 column 2:) {t[2, 1]}");`
		
		`// prints "Row 3 column 2: 6" since the`
		`// second element (with index 1) of the`
		`// third element (with index 3) in the`
		`// array is 6`
		
		`// Note that t[1, 2] would give an error`
		`// since there is no third element of the`
		`// second element in t`

Arrays (continued)

Updating an element in a two-dimensional array	`Identifier[IntExp, IntExp] = Exp;`	`t[2, 1] = 16;`
		
		`// t is now`
		`// {{1, 2}, {2, 4}, {3, 16}}`

Array length	`Identifier.Length`
	
	`Identifier.GetLength(Index)`
		`primes.Length`
		`// evaluates to 6`
		`t.Length`
		`// evaluates to 6 using example above`
		
		`t.GetLength(0)`
		`// evaluates to 3 using example above`
		`// primes.GetLength(0) evaluates to 6`
		
		`t.GetLength(1)`
		`// evaluates to 2 using example above`
		`// primes.GetLength(1) gives an error`

Further examples of array lengths	`Console.WriteLine($"The entire array t contains {t.Length} items");`
	`Console.WriteLine($"There are {t.GetLength(0)} sub arrays");`
	`Console.WriteLine($"Each sub array contains {t.GetLength(1)} items");`
	
	`// outputs:`
	`// The entire array t contains 6 items`
	`// There are 3 sub arrays.`
	`// Each sub array contains 2 items.`	

Arrays (continued)

FOR
(repeat the statements the number of times that there are elements in a list, each time giving Identifier the value of the next element in the list).
	`foreach (DataType Identifier in ArrayExp)`
	`{`
	`    // indented statements here`
	`}`
		`double[] ages = {15.5, 27, 19, 18, 17.2};`
		`double total = 0;`
		`foreach (double age in ages)`
		`{`
		`    total = total + age;`
		`}`
		`double mean = total / ages.GetLength(0);`
		`Console.WriteLine(mean);`
		
		`// calculates the total of the ages`
		`// held in the array and the mean`
		`// age and then outputs the mean`

Records

Record declaration (exam questions will use this method to declare and use records).	`struct RecordName`
	`{`
	`  public DataType Identifier;`
	`  public DataType Identifier;`
	`  …`
	`  public DataType Identifier;`
	
	`  public RecordName(Type Identifier,`
	`  …,`
	`  Type Identifier)`
	`  {`
	`  this.Identifier = Identifier;`
	`  …`
	`  this.Identifier = Identifier;`
	`  }`
	`}`
	
	`// The name of the structure`
	`// will start with a capital letter.`
		`struct Car {`
		`  public string make;`
		`  public string model;`
		`  public string reg;`
		`  public double price;`
		`  public int noOfDoors;`
		
		`  public Car(string mk,`
		`  string md,`
		`  string rg,`
		`  double pr,`
		`  int drs)`
		`  {`
		`  this.make = mk;`
		`  this.model = md;`
		`  this.reg = rg;`
		`  this.price = pr;`
		`  this.noOfDoors = drs;`
		`  }`
		`}`

Variable Instantiation	`RecordName Identifier =`
	`  new RecordName(v1, v2, …);`
		`Car myCar = new Car("Ford", "Focus",`
		`  "DX17 GYT",`
		`  1399.99, 5);`

Records (continued)

Assigning a value to a field in a record	`varName.field = Exp;`	`myCar.model = "Fiesta";`
		
		`// The model field of the myCar`
		`// record is assigned the value`
		`// "Fiesta".`

Accessing values of fields within records	`varName.field`	`Console.WriteLine(myCar.model);`
		
		`// outputs the value stored in the`
		`// model field of the myCar record`

Subroutines

Note: subroutines that have a type that is not `void` and contain a `return` keyword are functions. Those that have a type that is `void` and do not contain a `return` keyword are procedures.

Subroutine definition	`Type Identifier(parameters)`
	`{`
	`  // indented statements here`
	`}`
		`void showAdd(double a, double b)`
		`{`
		`  double result = a + b;`
		`  Console.WriteLine(result);`
		`}`
		
		`void sayHi()`
		`{`
		`  Console.WriteLine("Hi");`
		`}`
		`// Both of these subroutines are procedures`

Subroutine return value	`return Exp;`	`double add(double a, double b)`
		`{`
		`  double result = a + b;`
		`  return result;`
		`}`
		`// This subroutine is a function`

Calling subroutines	`// Subroutines without a return value`
	
	`Identifier(parameters)`
	
	`// Subroutines with a return value`
	
	`Identifier = Identifier(parameters)`
		`// Subroutine without a return value`
		`showAdd(2, 3);`
		
		`// Subroutine with a return value`
		`Console.Write("First number? ");`
		`double num1 =`
		`  Convert.ToDouble(Console.ReadLine());`
		`Console.Write("Second number? ");`
		`double num2 =`
		`  Convert.ToDouble(Console.ReadLine());`
		`double answer = add(num1, num2) * 6;`
		`Console.WriteLine(answer);`

String handling

String length	`StringExp.Length`	`"computer science".Length` `// evaluates to 16 (includes space)`

Position of a character	`StringExp.IndexOf(CharExp)`	`"computer science".IndexOf("m")`
		`// evaluates to 2`
		
		`string title = "Algorithms";`
		`int space = title.IndexOf(" ");`
		`// space will have the value -1`
		`// since title does not contain a`
		`// space`

Substring (the substring runs from the character **at** the first `IntExp` for the number of characters specified by the second `IntExp`).	`StringExp.Substring(IntExp, IntExp)`	`string t = "Computer Programs"`
		`Console.WriteLine(t.Substring(9, 8));`
		
		`// outputs the string 'Programs'`
		
		`// If the second number is omitted it`
		`// defaults to the value that gives`
		`// the rest of the string. So`
		
		`Console.WriteLine(t.substring(12));`
		`// outputs the string "grams"`

Concatenation	`StringExp + StringExp`	`Console.WriteLine("comp" + "science");`
		`// outputs the string "compscience"`
		
		`// Note that no space is automatically`
		`// added between each string`

String and character conversion

Converting string to integer	`Convert.ToInt32(StringExp)`	`Convert.ToInt32("16")`
		
		`// evaluates to the integer 16`

Converting string to real	`Convert.ToDouble(StringExp)`	`Convert.ToDouble("16.3")`
		
		`// evaluates to the double (real) 16.3`

Converting integer to string	`Convert.ToString(IntExp)`	`int number = 16;`
		`Convert.ToString(number)`
		
		`// evaluates to the string "16"`

Converting real to string	`Convert.ToString(RealExp)`	`double number = 16.3;`
		`Convert.ToString(number)`
		
		`// evaluates to the string "16.3"`

Converting character to character code	`Convert.ToInt32(CharExp)`	`Convert.ToInt32('a')`
		
		`// evaluates to 97 using ASCII/Unicode.`
		`// 'a' represents the character a`
		`// Using "a" will not work`

Converting character code to character	`Convert.ToChar(IntExp)`	`Convert.ToChar(67)`
		
		`// evaluates to 'C' using ASCII/Unicode`

Input/output

User input	Console.ReadLine()
	
	// If you need a prompt then use
	// Console.Write(prompt) before
	// using Console.ReadLine
		Console.Write("What is your name? ");
		string name = Console.ReadLine();
		
		// Console.ReadLine() returns a string,
		// so Convert.ToInt32() or
		// Convert.ToDouble() must be used to
		// convert the string to an integer or
		// to a double (real / float)
		
		Console.Write("How many cans? ");
		int quantity =
		  Convert.ToInt32(Console.ReadLine());
		Console.Write("How much? ");
		double price =
		  Convert.ToDouble(Console.ReadLine());

Input/output (continued)

Output	`Console.WriteLine(Exp);`
	
	`Console.Write(Exp);`
	`// Doesn't move to new line after`
	`// outputting Exp`
		`int a = 45;`
		`Console.Write(a);`
		`double g = 23.45;`
		`Console.Write(g);`
		
		`// To output more than one thing, or to`
		`// include text at the same time`
		`// use $ strings, eg`
		
		`int qty = 15;`
		`Console.WriteLine($"Quantity {qty}");`
		`// outputs the string "Quantity 15"`
		
		`Console.Write("Mary had ");`
		`Console.WriteLine("a little lamb");`
		`// will print the text on one line, but`
		
		`Console.WriteLine("Mary had ");`
		`Console.WriteLine("a little lamb");`
		`// will print the text over two lines`

Formatted outputs will be shown using interpolated strings (`$""` strings).	`$"…{identifier}…{identifier}…"`	`string name = "BT";`
		`int staff = 125000;`
		
		`Console.WriteLine($"{name} has {staff}`
		`staff");`
		
		`// outputs "BT has 125000 staff"`

Random number generation

Random integer generation (The first `IntExp` is inclusive, the second `IntExp` is exclusive).	`Random Identifier = new Random();`
	
	`int varName = Identifier.Next(IntExp, IntExp);`
		`Random rGen = new Random();`
		
		`int myNum = rGen.Next(3, 7);`
		
		`// generates an integer between 3 and 6`