# Chapter 2 – Using Variables

## What is a variable?

Variables are like containers. We use them in our programs to hold the values that we input, to hold changing values during the course of a program and possibly to output them during the course of, or at the end of, a program. In C#, when we declare a variable we need to state what type of data is going to be contained within it.

**X**

23

Here we have a **variable called X**.

**X** stores **integer** values and currently has the value of **23** assigned to it.

## Variable data types

The most common data types that you will use in C# programs are listed below. There are more, and these can be found easily on a multitude of help websites using the search term ‘C# variable data types’.

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **C# syntax** |
| **String** | A composite data type – strings are composed of ASCII characters |  |
| **Integer** | Integer data types store whole numbers in the range  -2,147,483,648 to 2,147,483,647 |  |
| **Real** | Real data types store numbers with a fractional part – there are a number of different data types for real numbers in C# |  |
| **Char** | A char data type stores a single ASCII character |  |
| **Boolean** | This data type will only store true or false |  |



Written Task

1. Find two more data types used in C# that are not listed in the table on the previous page.

Describe and give an example of the C# syntax for each.

1. A teacher plans a program to store her students’ mock exam marks. What data type would the teacher choose to store the values listed below?
2. The student’s surname
3. The student’s raw score
4. The student’s percentage mark
5. The student’s grade (A, B, C, etc.)
6. Whether or not the student has passed the exam

## Declaring variables

It is common practice to declare variables before using them. This serves the purpose of reserving a place in memory for that variable and the values that will be assigned to it. In C#, variables are declared by stating the data type followed by the identifier. The identifier is the name we give to the variable. This will normally be reflective of the values stored in the variable. It is good programming practice to use meaningful identifier names for variables and constants.



**Identifier**

**Data type**

## Initialising variables

Initialising variables is giving them their starting or initial value. This can be done at the same time as declaration, or afterwards.



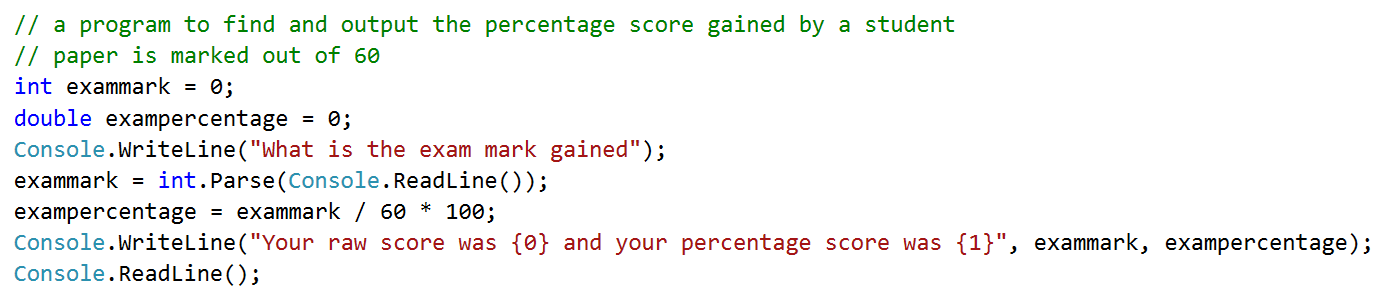
Here, this integer variable is both declared and initialised in the same statement



Here, the same integer variable is declared in one statement and initialised in a subsequent statement

## Assigning values to variables

A key aspect of the definition of a variable is that it is a value that can be changed; therefore, it is usual for a variable to be assigned a new value (perhaps many times) during the execution of a program.



In the program above, we can see that both exammark and exampercentage have been initialised with a default value of 0 (since it is not possible for a student to gain less than 0 in their exam) and then assigned a different value after the input of the exam mark by the user and after the actual exam percentage has been worked out.

## Assignment operations

There are a number of different assignment operations that can be used in a C# program.

|  |  |  |
| --- | --- | --- |
| **Operation** | **Example** | **Explanation** |
| = | exampercentage = exammark / 60 \* 100; | This is simple assignment. The value of the right-hand part of the statement is assigned to the left-hand part. |
| += | int total = 4;  total += 10;  the value of total after this operation will be 14 | Adds the right to the left and then assigns the value to the left  e.g. A+=B is equal to A = A + B |
| -= | int total = 4;  total -= 1;  the value of total after this operation will be 3 | Takes the right from the left and assigns the resulting value to the left  e.g. A-=B is equal to A = A - B |
| \*= | int total = 4;  total \*= 2;  the value of total after this operation will be 8 | Multiplies the right with the left and assigns the resulting value to the left  e.g. A\*=B is equal to A = A\*B |
| /= | int total = 8;  total /= 2;  the value of total after this operation will be 4 (as 8 divided by 2 = 4) | Divides the left with the right and assigns the resulting value to the left  e.g. A/=B is equal to A = A / B |
| %= | int total = 8;  total %= 3;  the value of total after this operation will be 2 (as this is the remainder when 8 is divided by 3) | This (%) is a modulus operator – it computes the remainder after division of the value on the left by the value on the right  e.g. A %= B is equal to A = A % B |

## Arithmetic operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Explanation** |
| **+** | Total = num1 + num2 | Adds two values together |
| **\*** | Total = num1 \* num2 | Multiplies two values together |
| **/** | Total = num1 / num2 | Divides two values |
| **-** | Total = num1 - num2 | Subtracts one value from another |
| **%** | Total = num1 % num2 | Finds the remainder from a division |
| **++** | num++ | Increments by 1 (so here, num would have 1 added to it) |
| **--** | num-- | Decrements by 1 (so here, num would have 1 taken away from its value) |

### Arithmetic operators and precedence

When calculating using variables or literal values, the rules of mathematical precedence (often referred to as **BODMAS**) apply. This means that operations are performed in the following order:

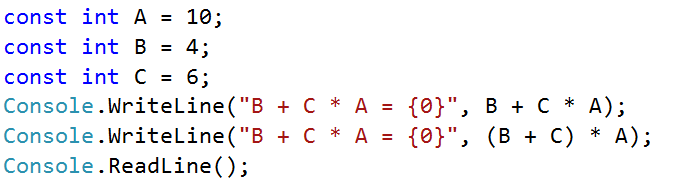
1. Brackets (parts of a calculation inside brackets always come first)
2. Orders (numbers involving powers or square roots come next)
3. Division
4. Multiplication
5. Addition



1. Subtraction

Written Task

Build the program below and answer the questions that follow.



1. What is the difference in the output from the two WriteLine statements?
2. Why is there a difference?
3. Find out what the rules of precedence are that are applied to calculations with arithmetic operators.

## The VAR keyword in C#

**Why have these two mathematical operations got different results?**

Generally, it is considered good practice to declare a variable with its data type before it is used. However, C# does provide a facility that enables the compiler to determine the type of the variable automatically, based on the value or expression assigned to it.

**var** amount = 23;

**var** surname = "Smith";

In the above examples, since the value assigned is an integer and string respectively, these variables will be declared as those types automatically. You cannot use the **var** keyword without assigning a value to it first, so the following statement would produce a compiler error:

**var** amount;

amount = 23;

## Constants

These are used when the user does not provide the value and you don’t want the value to change. Therefore, the value of a constant does not change during execution and cannot be changed from the value it is initialised with during execution of the program.

A constant must be declared and initialised at the same time.

****

## Variable scope

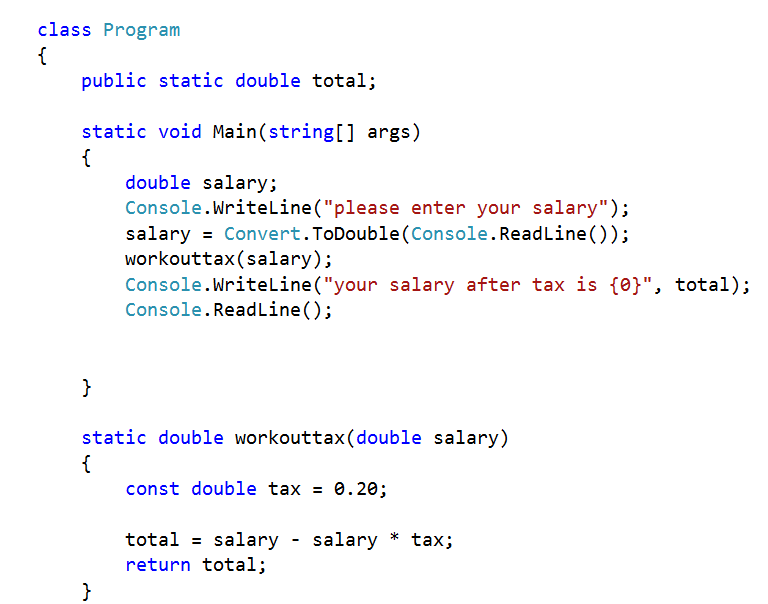
Scope is the term used to show where a variable can be used or seen. If a variable is declared, a memory location is used for that variable. When the block of code in which the variable was declared stops executing, the memory is released and that variable is no longer available to use.

**Global variable**

These types of variable are accessible from within any function or procedure. The reality is that in C# there really is no such thing as a true global, although it is possible to use a static variable declared within a class that will act in a similar way. A static variable is one where the lifetime of the variable is the entire run of the program.

**Local variable**

It is initialised and used only within the block of code in which it is declared. This may be within a procedure or function, or may be inside an IF block or loop block. When variables are declared within a procedure or function, that local variable may be passed as a parameter to another procedure being called (see more in the chapter dealing with procedures and functions).



salary can only be accessed by workouttax because it has been passed as a parameter when workouttax is called by Main procedure

total can be accessed and used by any procedure or function within class Program

tax can only be accessed and used within the workouttax function

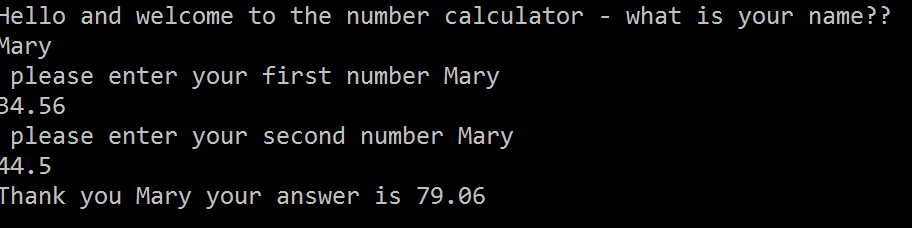


**{ }**

Coding Tasks

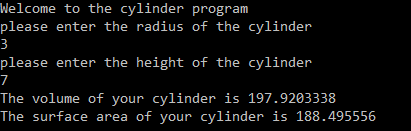
Program 2.1: *Adding Two Numbers Together*

Write a simple program that will add two numbers together. The output should look like the screen shown below. It should work for any name and any input numbers, including those with a fractional part.

****

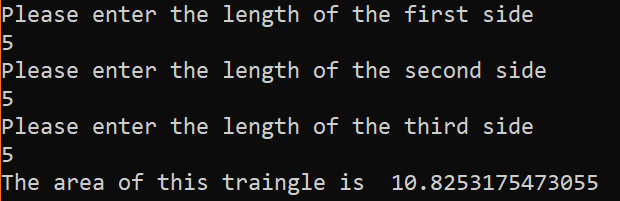
Program 2.2: *Cylinder Program*

Write a simple program that will find the volume and surface area of a cylinder. Pi should be set as a constant. The image below shows the expected output to the console window.



Program 2.3: *Working Out the Area of a Triangle*

Write a program to work out the area of a triangle when given the length of each side using Heron’s formula. The function Math.Sqrt() can be used to find the square root of a number.



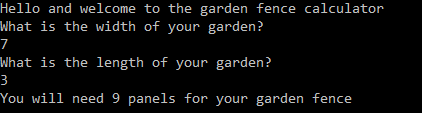
## Chapter 2 – Consolidation Tasks

Program 2.4: *Fence Panels*

Write a program that will work out how many fence panels are required for a rectangular back garden.

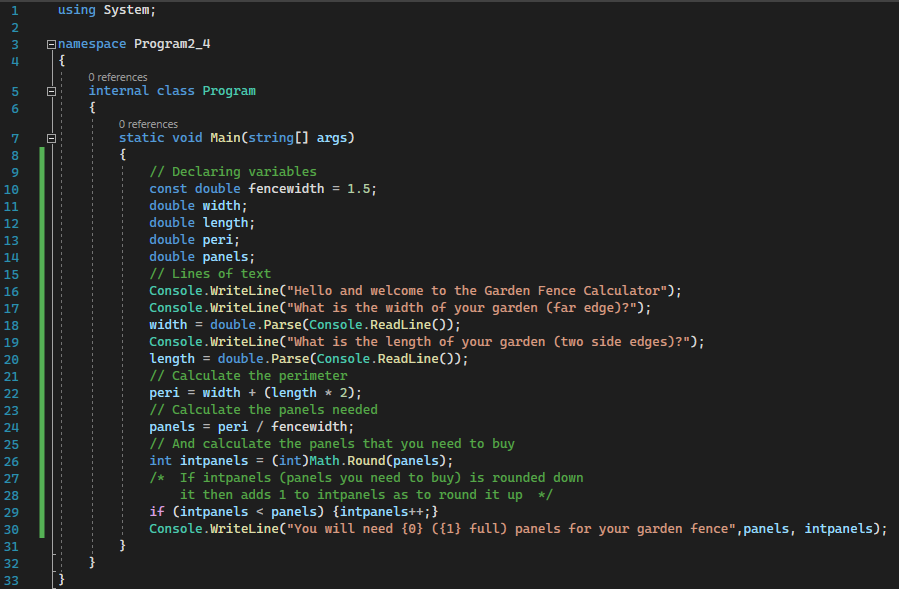
For the purposes of this program, garden fence panels come in a standard size of 1.5 metres. The program will first need to work out the perimeter of the garden that needs fencing (three sides) before calculating how many panels are required.

The image below shows the required output (although your program should work for a garden of any size).

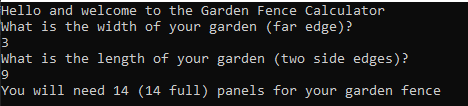


**Evidence required:**

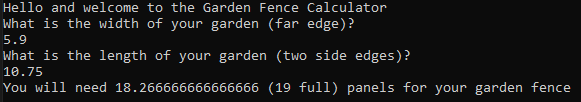
1. Annotated code listing for the above **(Code will also be sent via teams)**



1. Screenshots of the following tests:
   1. Where the garden length is 9 and the garden width is 3



* 1. Where the garden length is 10.75 and the garden width is 5.9



Now answer the following questions.

* 1. Why might it be appropriate to use a constant for the fence panel width? **[1 mark]**

The length of the fence panel width won’t change while in the code’s running so you can use a constant for it. Also, if it does change then it is easier to change it than to write “1.5” every time

* 1. How are constants different from variables? **[2 marks]**

Variables can have the value inside them change while the program is running/executed, while constants cannot. Also, constants have to have their values declared at the same time as declaring its data type while variables don’t

* 1. Why is it good programming practice to use meaningful identifier names for variables and   
     constants? **[2 marks]**

Because if someone were to add to the code, they would have a better time understanding what the program would do if the variables had meaningful identifiers instead of “a, b, c, d” etc.

* 1. Name three operations that might occur with variables during the course of a program. **[3 marks]**

1. addition

2. subtraction

3. multiplication

* 1. Why is it necessary to declare variables? **[2 marks]**

If you don’t declare a variable, the program won’t know what to do with it. If you were to store “12” in the variable “length”, the program can’t look at the variable name and assume it’s a number. It could think it’s a string. Also, it doesn’t know whether it’s meant to be an int, float or double

* 1. What impact does scope have on a variable? **[3 marks]**

If you declare it in the class instead of the “main” subroutine, every subroutine will be able to access it. However, if you store a variable in a subroutine, you won’t be able to access it in other subroutines unless you use parameter passing

* 1. A programmer has been asked to write a program to check the validity of an entered password to a computer system. The user will have only three attempts before being locked out of the system. The system will check the entered password against the stored password. What would be appropriate identifiers and data types for the following variables:
     1. A variable to store the entered password **[1 mark]**

passGuess (string)

* + 1. A variable to store the saved password **[1 mark]**

passCorrect (string)

* + 1. A variable to store the number of unsuccessful attempts **[1 mark]**

guesses (int)

* + 1. A variable to store whether or not the entered password matches the saved   
       password **[1 mark]**

guessCorrect? (bool)