Simon Light

Unit 6

Assignment 1

Contents

[Variables 2](#_Toc444506865)

[Arrays 3](#_Toc444506866)

[Input/output 4](#_Toc444506867)

[Logical operators 5](#_Toc444506870)

[If/Else 6](#_Toc444506871)

[Loops 7](#_Toc444506872)

[Benefits of a variety of data types 8](#_Toc444506873)

[The System Life Cycle 9](#_Toc444506874)

[Structures in programming 9](#_Toc444506875)

[Functions 9](#_Toc444506876)

[Procedures 10](#_Toc444506877)

[Classes and Objects 10](#_Toc444506878)

[Attributes 10](#_Toc444506879)

[Methods 10](#_Toc444506880)

[The importance of quality code 10](#_Toc444506881)

[Compatibility 10](#_Toc444506882)

[Encouraging good structure 10](#_Toc444506883)

[A sign of quality 11](#_Toc444506884)

[How to implement quality code. 11](#_Toc444506885)

[Factors of code readability 11](#_Toc444506886)

[Using a language/library feature 11](#_Toc444506887)

[Using white space 11](#_Toc444506888)

[Comments 11](#_Toc444506889)

P3.

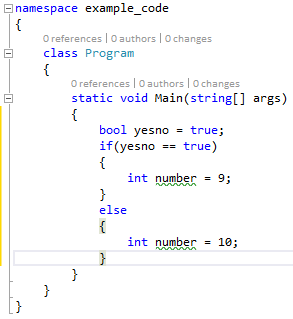
I am going to explain to you some basic programming knowledge and give you some examples of them. I am going to give all of my examples in C#.

# Variables

Variables are just letters or words that represent a value of data. They are called variables as they can be changed or edited by the program. It is very similar to algebra. In algebra x could equal 9. The same can be done in programming. The difference is, in programming a variable can take on multiple data types. You will learn about this in the next chapter.

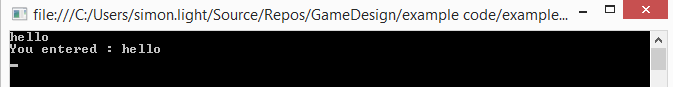
Here is a piece of code:

Main is the part of code that runs every time the application starts.



This then says if ‘yesno’ has the value of true (which it does) create an integer (whole number) variable called number and assign the value of 9 to it.

This statement creates a Boolean (true/false) variable called ‘yesno’ and assigns the value of true to it.



# Arrays

An array is a group of variables that have the same data type. Array positions always start at 0 and count up. The alphabet is an example of an array, therefore, alphabet[0] (the value in the array alphabet at position 0) is ‘a’.

This first line is the creation of an array. The “char[]” tells the computer that it is creating an array with the data type ‘char’ (characters). It then has the name of the array ‘alphabet’ and then the contents of the array, separated by commas.



The 3rd line contains an output. In this case it will display the value in the position 7 in the array alphabet (this is the 8th letter in the alphabet as the position starts from 0). This will then display ‘h’.

# Input/output

Input simply put is the user entering information into the computer. It can be done in many ways but I will show you how to do the most common. I will also be showing you how to output and one of the many tricks you can do while outputting.

This first line create a string (sentence) variable called input and assigns it the value of a ‘console.readline’. This means whatever the user enters into the terminal window it will be assigned to input.

The next line then is a console.writeline. This creates a new line in the terminal window and writes whatever you put into the brackets. If you want literal text you must surround it with speech marks to tell the code it is text. The plus then says after that put this. I then have the variable input. This means the console will take and input, then on the next line write “You entered : “and then whatever the input is.

# 

# 

As you can see above it worked.

It is common to end your code with a readline. This is so that the console must wait until you enter a value (press enter) to continue. Because there is no more code after this the console would just close. This would result in the console flashing up and disappearing before you could read the output.

# Logical operators

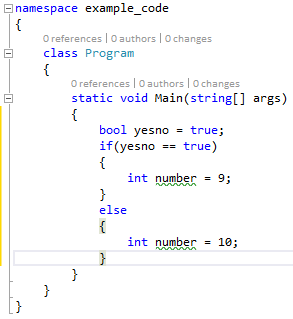
Logic operators ate very simple. They are just the stuff you learnt in maths plus a few more.

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Effect | Example of how to use | Outcome |
| + | Addition | 5+6 | 11 |
| - | Subtraction | 6-5 | 1 |
| \* | Multiplication | 5\*6 | 30 |
| / | Division | 30/6 | 5 |
| ++ | Add 1 | I++ | The value of I add 1 e.g. I = 7, i++, I = 8. |
| = | Assignment | Output = 7; | Output now has the value of 7. |
| == | Comparison | If(output == 7) | If output does equal 7 then the statement will be true. |
| && | And | If((output == 7) && (input == 7)) | If both output and input and input equal 7 then the statement will be true. Else it will be false. |
| || | Or | If((output == 7) || (input == 7)) | If either input or output equal 7 then it will be true. If both equal 7 or neither do then it will be false. |
| < | Less than | If(5<7) | This will return true |
| > | Greater than | If(5>7) | False |
| <= >= | Less/greater than or equals | If (5 <= 5) | This will be true |
|  |  |  |  |

# If/Else

If statements are statements that run what is in the ‘{ }’ if the statement inside the brackets are true.

After the code has run/not the code continues from after the final ‘}’



This if statement will always return true (and so will make number = 9.

Else statements run if the if statements above don’t. In this case obviously the else wont run as the if did.

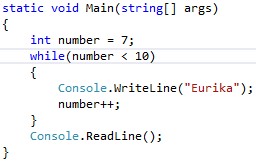
# Loops

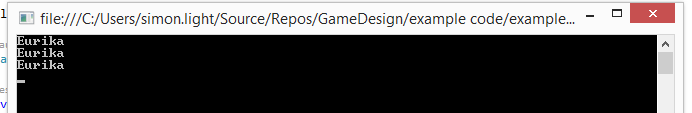
Loops are very similar to if statements except after the code has run the code jumps back to the line of the loop and tries to run it again.

While is the most commonly used loop and ill show that here.

Example:

In this example the code makes a new integer called number and assigns it the value of 7. It then checks if the number is less than 7, which it is, so there for writes “Eurika”. It then adds one to the value of number. It then checks again, passes, and runs the code. The code will keep running until number is not less than 7. In this instance the code should write eurika 3 times.





Which it does!

P4.

I am going to explain to you the different data types and you should be able to figure which you need to use.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data type** | **Example** | **Space occupied** | **Range** | **Comments** |
| Byte | Byte myByte = 255; | 8 bit | 0 to 255 |  |
| Sbyte | sbyte sByte1 = 127; | 8 bit | -128 to 127 |  |
| Char | Char letter = ‘a’; | 16 bit | U+0000 to U+ffff | Is declared with ‘ ‘ |
| Int | Int age = -72; | 32 bit | -2,147,483,648 to 2,147,483,647 |  |
| Uint | Uint age = 72; | 32 bit | 0 to 4,294,967,295 |  |
| Long | Long year = -2015; | 64 bit | -9,223,372  ,036,854,775,808 to 9,223,372,036,854,775,807 |  |
| ULong | Ulong year = 2015; | 64 bit | 0 to 18,446,744,073,709,551,615 |  |
| Short | Short day = -4; | 16 bit | -32768 to 32767 |  |
| UShort | Ushort day = 4; | 16 bit | 0 to 65535 |  |
| String | String today = “Tues”; | - | - | Is declared with “ “ |

## Benefits of a variety of data types

A programmer needs different data types to store different types of data. All data could be stored as one standard form (e.g. numbers) however this would be very confusing for a programmer which will result in mistakes.

Programmers also need different data types is to be efficient with memory. When you declare a new variable, the MMU (Memory Management Unit) in the CPU reserves a section of memory that the variable can use. If every time the programmer declares a new variable he reserves a massive piece of memory, it may cause the program to slow the computer or even crash. If the programmer reserves too little memory then he may over run the slot available and receive a ‘memory overload error’ causing the program to crash. This is why you have different sized data types that do the same things.

Another reason that a programmer may choose one data type over another is its use, this again relates to memory management. If in a program you want to declare a variable to represent time (in seconds), rather than using an integer (a very large data type that is positive and negative) you could use a ushort (a much smaller data type that is only positive). If you used the integer you would only be using half of the data type (the positive side) therefore wasting half of the memory that was reserved. This will not happen with the ushort as it is only positive (the same as time) and is much smaller than an integer (closer to the size that you are trying to represent).

P5.

# The System Life Cycle

The system life cycle is the method that you should use to avoid common problems when creating a new program. The cycle is:

* Definition – define what you are going to make
* Investigation and analysis – find out what can help you and what other products are on the market
* Design – keeping previous steps in mind design the program
* Implementation – put your plan into action and make the program to the previous requirements
* Testing – test the application to make sure it works and fits the previous requirements.
* Installation – move the new application into place and remove the old one
* Documentation – write up about what you have done and errors you have occurred when testing
* Evaluation – is it fit for purpose?
* Maintenance – keep doing updates and making sure that your program fits the requirements

Although you should always start with definition and end with maintenance it is not always a linear process. It may be necessary to finish a stage and then redo an earlier stage to a higher standard.

This systems life cycle above has been adapted for the life cycle of a piece of software rather than an IT system. The only major difference between the two life cycles is the installation process, the others have a direct comparison to each other.

As a piece of software can be distributed much more widely, the installation method can be a lot more difficult. Rather than removing one system and replacing it with another, software needs to be distributed. This is mainly done over the internet from a URL that people can access if they want to install the software. This is often accompanied with documentation on how to correctly install and configure the software. That way everyone has a piece of software that fits their requirements.

Software can also be distributed via hard copies such as installation disks or flash drives. This again has accompanying documentation on how to correctly install and configure the software. This method has the added bonus of the device not having to be connected to the internet. This is useful for companies setting up a large amount of devices as they don’t have to browse the internet each time they want install the software.

# Structures in programming

## Functions

Functions are pieces of code that rum when called. They can also be fed values through the brackets after the name of the function. Functions can come with the framework of the language or be created by the user for tasks that are being repeated multiple times in the program. An example of this is in C#.net there is a toString() function. This allows the user to convert the value of a variable to the form of a string e.g. 7.toString() = “7”;

## Procedures

Procedures are objects that the code will create for you. They are run when an event happens (making them event driven). These could include clicking a button in a form application or pressing a key. An example of this is the main procedure which is created for you in C# and always runs first in your code.

## Classes and Objects

An object is basically a function that has data linked to it. These objects are then linked together by calling each other. This then makes functioning code. Object are usually linked to what they are needed for in the finished application. E.G. in a system for a library, an object could include books members loans .etc.

Classes are basically templates for an object. For example: in the class ‘cars’, it might include: ford, Renault and Vauxhall, as they are all instances of cars.

## Attributes

Attributes are the before mentioned data that is put into a class. This is basically information about the object which can be used to interact with other parts of the code. If a person was the object, their attributes could be name, age, height, weight, hometown .etc.

## Methods

Methods are pretty much functions that classes can call. This mainly interacts with the attributes of the objects. An example could be ‘changeOfAdress()’ for the example given above. It would probably also take an input such as name and new address so that it can update the attributes correctly.

M1.

# The importance of quality code

## Compatibility

If speak with lots of slang and colloquialisms then it may become confusing. Your immediate peers may understand you but people half way across the world probably won’t. The same is with code. If you create a program around no real structure then trying to integrate it into other programs will be near impossible. If you follow some pre-set rules, it will be very easy to integrate. One place that this is very evident is creating browser add-on’s. If you use code that is very unstructured then it simply won’t work. If you follow the browsers developer guidelines then it will be very easy to get it to work.

## Encouraging good structure

When you follow someone else’s structure it is usually going to force you to use the language you are, in the way it was intended to be used. This means your program is going to run much better and more efficiently than first expected. It is likely to force you to use code structures you would not usually. Over time this will make you a much better programmer and it will make your code look much better to your employer, which takes me to the next point.

## A sign of quality

Any company or organisation employing a developer will have a competent 3rd party to check both the end program and the code used to create it. If this person sees very sloppy and unstructured code with very few comments, he will suggest to the employer that the code is substandard. This may be untrue of the application but it gives people these impressions. You wouldn’t publish a writer if you read a book of theirs where they had cut corners and taken the easy route. This is much the same for developers.

# How to implement quality code.

Good practise for code is to split your code up into separate objects. You then have a main loop that runs constantly and calls each class to do its job. This is mainly use in OOP however it can be translated to any paradigm.

Another way to do this is to comment your work. This can include commenting out code that doesn’t work so that you know what you have tried and also commenting a description of what your code is doing.

It may be necessary to document parts of your work to have full understanding when you come to edit/update it.

D1.

# Factors of code readability

The person who is going to be maintaining your code is almost always going to be less competent than the person who wrote it. You need to remember that someone else is going have to know what the code does on each line. Here are a few factors to make sure of this.

## Using a language/library feature

Just because a function in one language/library is available it doesn’t mean you should use it. The code maintainer may not have come across some of the functions before and it will be quite difficult to work out what it does. You should remember that the function of the code is to do the job it was asked to, not do it in as few lines as possible. If the function is very common to a lot of languages (e.g.array.reverse), then the maintainer will either know what it is or there will be a lot of documentation about how to use it. This is a good choice to use.

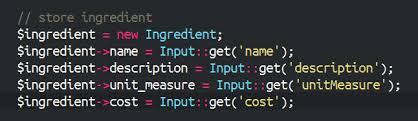
## Using white space

Using white space (blank lines) can split up chunks of code. It is a good idea to split up code into chunks that do different things. This allows for easy navigation when trying to edit a singular piece of code.

## Comments

There are 2 rules to follow when commenting code; comment well and comment often. You need to comment exactly what you want your code sections to do and what each part of the code chunk does. You also need to explain any complex features you have used. Not all comments need to be made in the code. It is common to include additional information in an appendix style document. This can be then be referred to like so: //see appendix 3.1. This makes commenting a lot easier as you can copy information about language specific code to refer to. It also means you don’t have to keep writing out the same information (possibly making mistakes). This means that the code won’t be overrun with comments rather than actual code.

Here is some code that is hard to read:



By implying all of the factors above I get this code:



As you can see the code is nicely aligned and very easy to read. This means that you can read the code much quicker and it is easier to debug without making mistakes.

Bibliography

<https://msdn.microsoft.com/en-us/library/364x0z75.aspx>

<https://msdn.microsoft.com/en-us/library/s1ax56ch.aspx>

<http://www.teach-ict.com/as_a2_ict_new/ocr/A2_G063/331_systems_cycle/slc_stages/miniweb/pg17.htm>

## 