**Software**

**What is Software?**

* A Computer program in the form of machine-readable instructions that directs computer to perform particular operations.
* Generally divided into application software and system software.

**Application Software**

* **Application software** consists of programs designed to make users more productive and/or assist with personal tasks
* Available in a variety of forms

**System Software**

* System software serves as the interface between the user, the application software, and the computer’s hardware
  + Utility programs

****

**Programming Languages**

Computers work purely using binary – everything is converted into binary code for a computer to read and follow instructions. So, if we want a computer to follow a simple instruction ***x + y*** then this needs to be converted into binary. If ‘***x***‘ was represented in binary by ***0011*** and ‘***y***‘ by ***0100***, the ‘**+**‘ might be represented by ***1111*** the instruction might be converted into ***machine language*** looking like ***001111110100***. Fortunately we don’t need to write programs in machine language in order to instruct a computer as this would be very error prone!

|  |  |
| --- | --- |
| language | As humans, we use Natural Language to express our thoughts. Unfortunately it isn’t possible to instruct a computer using Natural Language, as it is prone to ambiguities in meaning (S*emantics*) which computers can’t identify – consider how you could define the word ‘*Spring*‘.  Natural Language can also have problems with grammar (*Syntax*) - it wouldn’t be correct to say in English *“Eat I now food”* although we would probably understand. Therefore, ***Programming Languages*** have been developed in between ***Natural Language*** and ***Machine Language***as neither are suitable. They are designed to have a more precise (unambiguous) syntax and semantics, to ensure the computer responds as intended, without using the error prone binary code.  Early programming languages were ‘***low level***‘ or assembly language – an example could be;  ADD X Y Z  In Natural Language this could be “add the value in memory location x to the value in memory location y and store the result in memory location z”. This would then be converted by the computer into a binary instruction similar to the one we encountered before. ‘***High level***‘ languages (such as C++) use much more sophisticated instructions, and then use a ‘***compiler***‘ to translate the instructions into machine language. |

**Problem Solving**

A computer programmer is responsible for writing computer programs that is a series of instructions written in a programming language, which a computer can follow to perform a task. When learning their first programming language, many students believe that learning how to translate their ideas into the language a computer can understand is the difficult part of problem solving… …***this is NOT true***! Learning to use a programming language, is ***not*** like learning to use a different natural language, so you shouldn’t approach it in the same way that you did when learning English, (or Thai, Japanese, etc.) A good programmer is one who understands how to solve a problem - one who has a *method of solution*. If you understand how to solve a problem, it is then relatively straight forward to translate your solution into a series of instructions, in C++, or any other programming language.

|  |
| --- |
| ***Software Engineering Lifecycle*** |
| ***The Waterfall Model***  waterfallmodel |

|  |
| --- |
| ***The Spiral Model***  spiral |

The first step when considering developing a computer program is to ensure that you understand the nature of the problem (*Requirements Engineering*). This may sound obvious, but many software projects have failed due to incomplete problem understanding. Once you understand what your program needs to do you can consider how you could solve the problem(s) you are faced with.

|  |  |
| --- | --- |
| ***BREAK IT DOWN!!!*** | |
| Yoda | “Break your problem down into smaller more manageable chunks!” |

Often problem solving involves developing an ***algorithm***, or sequence of precise instructions, which can be followed to produce the intended result. This is breaking the problem down – taking the whole problem, and breaking it down into various parts which can be more easily managed.

**Exercise 1**

|  |  |
| --- | --- |
| code | Consider the problem;  “Go To Bangkok”  Let’s break this down and develop an algorithm to solve this problem. |