Introduction to Computers and Programming

What a computer does

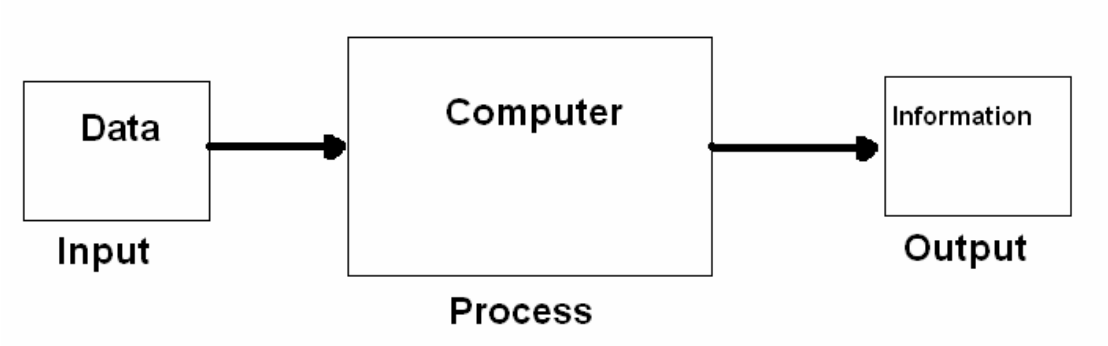
From a programmer's viewpoint, at its simplest level, a computer processes data into meaningful information.

Although you may not think there is a difference to a computer programmer there is a quite a difference.

Data consists of raw facts and figures.
Information is processed data; information is meaningful to the end user.

The information need not be thought of as purely mathematics, financial or such non-entertainment based business. Information can be visual or audio. Displaying the graphics incorrectly or mapping the textures wrongly and your game isn't going to look to meaningful.

Data its self isn't meaningful to the end user, the computer game player or the manager looking at payrolls.



A computer with an effective program can turn the data it needs into information meaningful to the end user.

The program is the driving force of the behind any job (or process) that the computer undertakes.

*We'll be covering what exactly a computer program is in more depth later one, for now I hopefully want to dispel some of the myths surrounding computer programming.

Only maths experts can program computers
Well thank goodness this is a myth as thousands of programmers would be out of work, me included.
Certain aspects of games programming you need some mathematics knowledge, matrix multiplication, linear algebra, but the maths used isn't rocket science and the computer handles the actual calculations. Plus those aspects of game programming are for a different course.

Computer programs make mistakes
Your may have heard the adage "To err is human, but to really foul things up takes a computer". This may be accurate but only in the sense that it duplicates any error caused by a human very rapidly.
Computers do not make mistakes, people make mistakes. People program computers, people run them, and people enter data.
Computers are finite machines; when given the same input, under the same conditions, they always produce the same output.

Programming is difficult
Well computer programming isn't always easy and some people will take to it more naturally than others. As long as you are willing to work and learn what the computer expects then you should get through it.
Computer programming can be rewarding, both in terms of solving problems and financially. Computer (games) programmers get paid a good salary
This course also aims to give you an introduction to programming, so isn't that difficult (we hope).

The programmer's life

Writing a program can bring about a sense of accomplishment, similar to what an artist or craftsman feels when completing projects. Writing the program though can be at times tedious, large programs require a lot of detail. The tedium is not like other jobs, such as factory work, here the computers quick feed back on your mistakes quickly enables you to try and correct the problem.

So other than a little patience as a programmer you'll need to learn that proper design is critical to a successful program. Some of you may have heard the term *systems analysis and design*. Sounds clever, but essentially it is the practice of analysing the problem to be programmed and then designing the program from that analysis.

Programming as directions

When you sit down at your desk as a programmer, your computer is a blind dumb machine waiting for you to give it instructions. When you do give the computer instructions it follows them, without second guessing them.
If you tell a computer to do something incorrect it will do it.

Essentially a program is a list of detailed instructions that the computer carries out.

The problem comes when giving the computer these instructions. The solution is that the instructions have to be explicit.

To give you an example of some of the thought necessary in programming I'll use a car example shameless taken from somewhere else.
If we had someone learning to drive from our society and in our age, someone much like you or me and we wanted them to start the car we could give them the following instructions:

1) Use this key
2) Start the car

However if we had an example of middle ages man, called Burt, and we gave him those instructions he may not get very far – we would have assumed to much knowledge on the part of Burt.
Burt would know nothing about cars or automobiles in general. You would have to give Burt a much greater set of detailed instructions:

1) Attached is the key to the car. You need it to start the car
2) With the key in hand go to the car door that is closest the front door of the house
3) Under the door handle you will see a round silver coin-sized metal part, in which you can insert the key
4) After putting the key in the hole as far as it goes, turn right until you hear a click
5) Turn the key back left until it faces the same way as when you inserted it and remove the key

This could go one, but as you can see if already have five instructions and we haven't even sat in the car yet, let alone go on to describe using the accelerator while he turns the key in the ignition.

This is an analogy for programming give an example of how explicit we need to be when telling the computer what we need to do.

A game consists of more instructions than:

1) take data
2) display in a coherent fashion
3) take user input
4) alter the data so the display is updated and still coherent

Speaking the computers language

The instruction you give to computers must be in a language the computer understands.
At its lowest level a computer is nothing more than a series of switches flipping on and off. In fact early computers had to be given data input by flipping switches. This on-off state is represented by 0's and 1's or binary as it is known.

Nowadays we have a variety of options to program computers, these are known as programming languages.

What are Programming Languages

Programming languages are the languages you write your programs in.

Computers only understand binary codes so the first programs were written using this notation. This form of programming was soon seen to be extremely complex and error prone so assembler languages were developed. Soon it was realised that even assembler languages could be improved on. Today, a good programming language must be:

- Totally unambiguous (unlike natural languages, for example, English -- `old women and men suck eggs', does this mean that men or old men suck eggs?)
- Expressive -- it must be fairly easy to program common tasks
- Practical -- it must be an easy language for the compiler to translate,
- Simple to use.

All programming languages have a very precise syntax (or grammar). This will ensure that a syntactically-correct program only has a single meaning.

Why do we need programming languages, why can't we use English?

- Computers are too stupid to understand English
- English would make a lousy programming language:
  English as a language is full of ambiguities.

The 'old women and men suck eggs' is just one example. Consider the following sentences:
"Time flies like an arrow"
Time = noun, flies = verb "like an arrow" = action description

"Fruit flies like an orange"
flies = noun, like = verb, "an orange" = object

If we get the computer to except the first sentence it would have trouble adjusting to the second.
Understanding simple sentences like this, which most of us take for granted, posses tremendous problems to the computer. Languages like English are not explicit by there nature.

Language Levels:
There are three main language levels:

Machine Language (very low level):

- Computer's native language
- Consists of 0's and 1's (bits), binary encoded
- Different for each machine (machine dependant, different computers understand different sequences)
- All programs must be written in machine language (code) or be translated in to machine code before execution

Assembly Language (low to intermediate Level):

- A mnemonic is used to represent each of the machine language instructions
- Typical instructions for addition and subtraction might look like this:

Assembly Language	Machine Language
ADD	100101
SUB	010011

- Assembler: program that translates an assembly language into machine code
- Still hard for humans to understand

High-level languages:

- Machine independent
- Must be translated before execution (compiled or interpreted)
- Closer to English and algebra
- Easier to read and understand:

```
hypotenuse = Math.sqrt(opposite * opposite + adjacent * adjacent);
```

- Fortran, Pascal, Smalltalk, C, C++, Cobol, Ada, Java, etc.

What Programming Language are we using:
We will be using C++

What is C++:
"The C++ language was developed at AT&T Bell Laboratories as a general-purpose programming language based on C. In addition to the features provided by C, C++ provides data abstraction and support for object-oriented design and programming through classes, multiple inheritance, virtual functions and templates. C++ also provides operator and function name overloading, reference types, memory management operators and inline functions. Features of C++ such as constant types and function argument checking and type conversions have become part of the ANSI C programming language".

"C++ is an object-oriented programming language created by Bjarne Stroustrup. C++ maintains almost all aspects of the C language, while simplifying memory management and adding several features - including a new data type known as a class (you will learn more about these later) - to allow object-oriented programming. C++ maintains the features of C that allowed for low-level memory access but also gives the programmer new tools that simplify memory management".

Standard C++

- C++ was formally adopted by the American National Standards Institute (ANSI) and International Organization for Standardization (ISO) to provide a common base in 1998.
- It is unambiguous and machine independent
- C++ implementations for a specific machine or operating system usually provide "extensions" to the standard. The use of the extensions reduces the portability of the resulting C++ code
- Is not yet fully supported by any widely used compiler

Why C++:

No one fantastic good for all language, different game programmers use different languages, often C or C++.
C++ is a superset of C.
Using OO provides a better way of modelling data in games.

Getting Started:

C++ is a programming language of many different dialects, just as each spoken language has many different dialects. In C++, dialects are not because the speakers live in the North or South; it is because there are several different compilers. There are four major compilers: Borland C++, Microsoft Visual C++, Watcom C/386, GCC and DJGPP. You can download DJGPP.

It is not just C++ that has all these dialects, other languages such as basic and C, do as well.

Obtaining a complier will be of use when completing the course assessments.

As with C, application programs written in C++ can usually be transported to a variety of machines. It is important to remember, however, that C++ was standardized more recently than C, and as a result different C++ compilers still may differ in how closely they track the standard. Each one will support the ANSI/ISO standard C++ functions, but each compiler will also have non-standard functions Sometimes the use of non-standard functions will cause problems when you attempt to compile source code (the actual C++ written by a programmer). Implementations may also differ in various anachronisms of the language allowed. Users are discouraged from using anachronisms unless they occur in existing code that is difficult to change.

Program Translation

We have already mentioned compilation above, but what is a compiler?

Compiler

- A program that translates a high-level language into machine code
- Source program: a program in a high-level language
- Object program: the machine language version of a source program
- High-level languages allow us to write portable (or machine-independent) programs

I have motioned that we use a language because a computer cannot understand English and now you're telling me that the computer can't understand C++ (or any other high level language).
That's right, but it is easy for a program to translate C++ into a form that the computer does.
A compiler takes programs written in a high-level program language and translates them into a form readable by the computer.