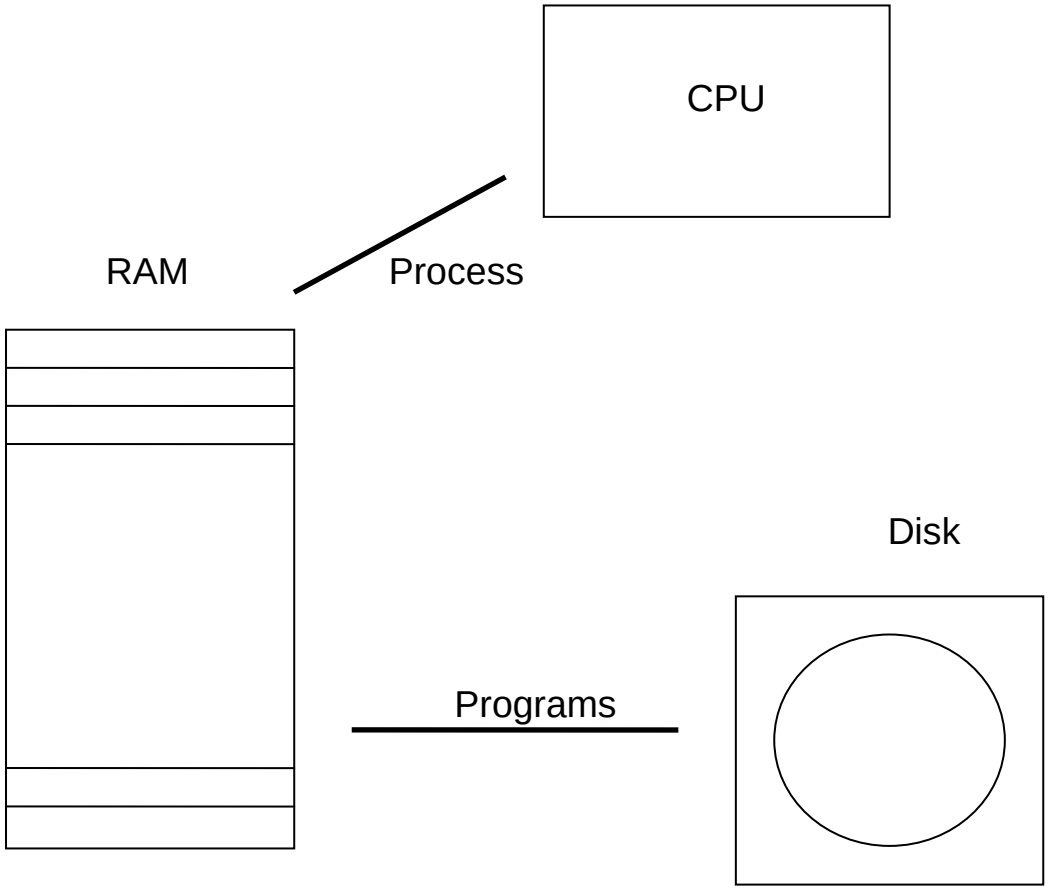
INTRODUCTION TO COMPUTER PROGRAMMING

Richard Pierse

Class 1:
Introduction to Computers and Programming Languages

How does a computer work?

Here is a simple schematic diagram of the structure of a computer.



The ***CPU*** (Central Processing Unit) is the 'brain' of a computer and handles all the calculations.
RAM (Random Access Memory) is the 'memory bank' of the computer and holds programs and data that are being processed by the CPU. All programs and data need to be loaded into memory from disk before the CPU can use them.
Modern computers often have several available disk sources. These include floppy disks, hard disks, CD and DVD.

What is a computer program?

The CPU does the computer's calculations but it needs detailed instructions as to what calculations to do and in what order. It is the computer program that tells the computer what to do. Without a program, the computer would not be able to do anything at all.

When you switch on a computer, several computer programs automatically start running. These programs are part of the *operating system*. The operating system controls the operation of the computer and allow other programs to run such as Word Processors, Spreadsheets, web browsers, computer games etc. Examples of operating systems are the various versions of Microsoft Windows (95, 98, NT, XP etc.), UNIX, DOS, Mac OS.

Computer ***hardware*** comprises the physical parts of the computer: the CPU, memory chips, disk drives, disks, monitor, printers etc.

Computer ***software*** comprises the programs and data that run on the hardware. Although, software may come on a disk, it is not the disk itself but the programs on the disk that are software.

When do we need to program?

Your computer comes already equipped with many programs like Word Processors, spreadsheets, and web browsers that allow you to do many things. Some, such as *Excel*, allow you to write your own small programs in the form of macros (as you have seen in a previous module). In addition, special-purpose programs exist to solve specific economic problems in areas such as econometrics (*Eviews*), model solution (*WinSolve*), Operations Research (*Lindo*).

If a program already exists to solve your problem, it will almost always be better to use it rather than write a program yourself. However, there are occasions when no program does exist to solve your problem so you will need to write your own program. Your first decision will be what language to use. Choices include general purpose languages such as *BASIC*, *Java*, or *C*, or special-purpose languages like *Matlab*, *Gauss*, *Mathematica* or *Maple*. Programming can be a lot of fun but it can also be hard work so it should not be undertaken unnecessarily.

Machine Code and Compilers

The computer can understand instructions only if they are written in a special code called ***machine code***. Machine code is specific to the particular type of computer. For example, here is the machine code for an IBM PC (Intel 80386) to add up three numbers and compute their average.

Machine code	Assembly instructions	Meaning
D945FC	`fld dword ptr[ebp-0x04]`	load number from RAM
D845F8	`fadd dword ptr[ebp-0x08]`	add second number
D845F4	`fadd dword ptr[ebp-0x0c]`	add third number
D95DF0	`fstp dword ptr[ebp-0x10]`	store sum in RAM
DB2DBC104000	`fld tbyte ptr[0x4010bc]`	load value 1/3
D84DF0	`fmul dword ptr[ebp-0x10]`	multiply sum by 1/3
D95DF0	`fstp dword ptr[ebp-0x10]`	store result in RAM

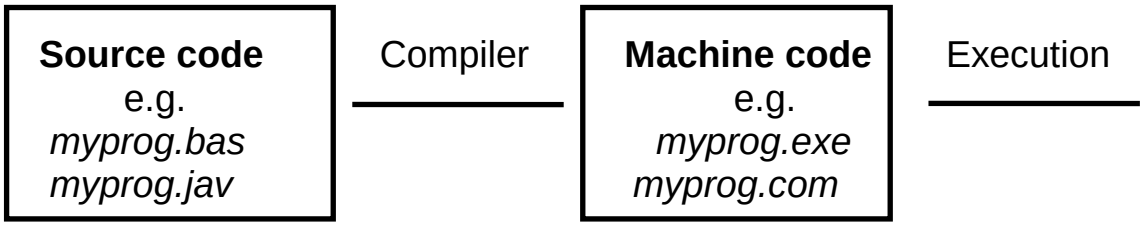
- The first column lists the machine code in hexadecimal. (Hexadecimal is a compact way of writing binary numbers the computer understands using the sixteen 'digits' 0-9 and A-F).
- The second column lists the same machine code in ***assembler language***. Assembler is a simple language that makes it slightly easier for a programmer to write machine code. However, it is still not very easy to understand!
- The third column explains the meaning of the various machine code instructions in English.

In the early days of computing, all programs had to be written directly in machine code. This was extremely tedious and prone to error. Also, a machine code program written for one machine would need to be rewritten to run on a different type of machine.
The solution to this was the invention of ***compiled*** computer languages. With a compiled language, the programmer writes a program as a set of instructions in a high-level language designed to be easy to use. This program is called the ***source code***. Then a computer program, called a ***compiler***, translates the source code into machine code, which can then be run or ***executed*** on the computer. With an appropriate compiler, the same source code can be translated to run on any machine.

The main advantages of compiled languages are:
- ease of use - the language is designed for people not machines
- portability - the same source code will execute on any computer

Compilers

The following schema shows the separate stages of compilation and execution.



Files of source code generally have a suffix to identify the language e.g. `.bas` for ***BASIC***, `.jav` for ***Java***, `.c` or `.cpp` for ***C*** or ***C++***. Source code files are human-readable and can be viewed in any text editor. Machine code files that can be executed by the computer are indicated by extensions such as `.exe`, `.com`, `.dll`. These files are not human-readable.

Examples of Computer Languages

Consider again the task of adding three numbers, A1, A2 and A3 and computing their average (as variable A4). Let us see how code to do this might be written in some different computer languages:

BASIC: `a4 = (a1+a2+a3)/3.0`
Java: `a4 = (a1+a2+a3)/3.0;`
C: `a4 = (a1+a2+a3)/3.0;`
Gauss: `a4 = meanc(a1|a2|a3);`

Firstly, note how much simpler, more compact and closer to familiar algebra are all these examples than the machine code we looked at earlier. Secondly, note the similarity between the source code in the first three examples. This illustrates how knowing one computer language can often help in understanding other languages. Thirdly, note the last example. *Gauss* is a special-purpose language designed for dealing with vectors and matrices. Here the average is calculated as the mean of a vector formed of the three variables, using a built-in *Gauss* function, `meanc`. This illustrates how a special-purpose language can be useful for dealing with special problems.

Code Optimisation

It used to be true that programs written in compiled languages were generally less efficient than those written directly in machine code. This was because good machine code programmers would know the detailed working of a computer and would tailor their code to take advantage of the quirks of a particular machine. They would know for example that addition is much faster on a computer than multiplication, and that multiplication is faster than division. As a result, most critical software such as operating systems and games software used to be written directly in machine code.

Modern compilers can automatically optimise source code to increase execution speed and/or to reduce the size of the executing program. Optimisation increases the time it takes to compile a program but will reduce the time it takes to execute. Since a program is normally compiled once but may execute many times, optimising code is generally a good idea. As a result of automatic optimisation, programmers don't need to worry about the details of how the computer operates and can concentrate on writing source code that is clear and easy to understand.

These days compilers are very good at optimising code to run efficiently and high-level languages like *C* are now used to write both operating systems like *Windows* and major applications such as *Excel*. In fact, the *C* language was originally invented in order to write the operating system of *UNIX* machines.

Compilers versus Interpreters

A ***compiler*** translates source code into machine code and produces as output an executable file. This executable file can then be run on any computer without the need for the compiler to be present. In contrast, an ***interpreter*** is a program that translates the source code and then executes it without going through the intermediary stage of producing an executable file. As a result, the interpreter needs to be present every time that the program is run. The analogy is with language translation: a compiler is like a written translation of a text, an interpreter is like a human real-time translator. Some languages such as *BASIC*, traditionally have an interpreter rather than a compiler. Interpreted languages are generally not optimised because they are compiled each time they are executed.

Choosing a Language

There are a great many different computer languages and new ones are continually being invented. Some languages are better than others for particular tasks. However, people have different preferences and most computer programmers only know a small number of languages. It is easier to continue to program in a language you already know than to learn a new one.

Languages divide into two main categories: general purpose languages which are designed for general programming, and special purpose languages which are designed for particular tasks. For example *FORTRAN* is an early example of a general purpose language, first developed in the 1950s. Although it has been largely superseded by more modern languages such as *C*, it continues to be used for scientific applications, mainly because of its familiarity. *HTML* is a special purpose language used for developing web sites (we will be looking at *HTML* in class 8 of this module). *Java* (which we will be learning later in this module) is an example of a language originally developed for web-based applications, which has become a general purpose language that is rapidly taking over from *C* and *C++* (on which it is closely modelled) as the most widely used general purpose language.

General purpose languages:

Language	***Compilers***	***Comments***
BASIC	*Visual, Liberty*	Originally developed for teaching
FORTRAN	*Lahey, Salford*	Early high-level language
Pascal	*Turbo*	Structured language, based on *Algol*
C	*Visual, Borland*	Very popular and efficient language
C++	*Visual, Borland*	Object-oriented version of C
Java	*Sun, Forte*	Becoming popular, especially on the web

Special purpose languages useful for Economics:

Language	***Author***	***Comments***
Gauss	*Aptech*	Matrix language widely used in econometrics
Matlab	*Math Works*	Similar to Gauss, more widely used in science
Mathematica	*Wolfram*	Mathematical language, including algebra
Maple	*Waterloo*	Similar to Mathematica, good for graphics
HTML	*W3C*	Special language for writing web pages

It is best to use special purpose languages wherever possible. For econometric applications, *Gauss* or *Matlab* are ideal and pre-existing code is available for many problems.

Installing *Liberty BASIC*

We are now ready to start programming using *Liberty BASIC*. *Liberty BASIC* is a shareware version of a the *BASIC* language that is similar to Microsoft's *Visual BASIC*. A copy of a compiler for *Liberty BASIC* comes on the CD that accompanies Greg Perry's textbook for this module.

Getting started with *Liberty BASIC*

Loading a Sample program

Before, writing our first program, let us take a look at an existing example program that comes with *Liberty BASIC*.

1. Select ***File, Open***.
2. Select `draw1.bas` and click OK. The source code in the `draw1.bas` file will appear in the *Liberty BASIC* editor.
3. Select Run from the Run menu (or click on the run button which has a picture of a man running). *Liberty BASIC* compiles the program and runs it in one step. A drawing window will open.
4. To draw a line, circle or square, click on the appropriate icon. Click somewhere in the window, hold down your mouse button and drag the mouse. When you release your mouse button, the line, circle or square will appear.
5. Clicking on a colour icon before drawing changes the colour of the shape.
6. To stop the program, click on the Close button.

This program illustrates the power of *Liberty BASIC* to create a simple Windows program. We will be learning about how to write Windows and graphical programs in class 5. However, for now, our first program will be a little less ambitious.

Running our First Program

1. Select ***File, New*** to clear the source code window and prepare the editor for a new program.
2. Type the following lines into the editor:

```
' prints a countdown from 10 to 1
for i = 10 to 1 step -1
     print i
next i
print "Blast Off!"
end
```

3. Select Run from the Run menu (or click on the running man icon) to run the program.

The program should print the numbers 10 down to 1 followed by the message “Blast Off!”.

Analysing the Program

Let us look in detail at the program we have just typed in.

- The first line (beginning with a single quote character) is a ***comment***, called a ***remark*** in *BASIC*. Any line starting with a single quote is simply ignored by BASIC so we can use it to add any comment that we like to our program
- The second line is the start of a loop. All the commands between the `for i` and the `next i` commands will be repeated for values of the variable `i` going from 10 down to 1 in steps of -1. In this case the only command in the loop is the third line `print i` and so the loop is equivalent to the 10 commands:
  `print 10`, `print 9`, and so on until `print 1`.

This will simply "print" these numbers to the output window.

- The program ends by printing the text ***Blast Off!*** to the output window. Note that the text to be printed needs to be surrounded by double quotes.
- The final `End` command tells the compiler that this is the last line.

It might seem as if this program took a lot of effort for little result. However, by simply changing the `10` in the second line to `10000`, we can get the program to print numbers from 10000 down to 1!

Alternatively, by inserting the line

```
input n
```

before the second line and changing the `10` in the second line to `n`, we get a program that will print a countdown from any number n, that we supply. The program waits for us to type in a value for n, showing a question mark as a prompt.

What would happen if we typed in the value 0? Since 0 is less than 1, the loop will never be executed and the program will immediately print the text ***Blast Off!*** and exit.