

High Level Languages

What is a high level language ?

High level programming languages are used to allow people to communicate with computers. They are at the top of a hierarchy of languages which allow users to give instructions to a computer :

1. High Level Languages
2. Assembly Language
3. Machine Code

Ultimately, any information given to the Central Processing Unit (CPU) of a computer must be in binary form (machine code).

Machine code and assembly language are processor specific, ie they can only be used to give instructions to a particular type of processor. Also, they are difficult for people to use because they are very different from the ways in which people normally communicate. For example, to instruct a computer to add two numbers together and store the result, the following section of code would be required for the Z80 processor :

```
LD A , 0H10      ; Load Accumulator with Hex 10
ADD A, 0H10       ; Add Hex 10 to contents of accumulator
LD (0H1000), A    ; Save contents of accumulator at memory location
                  Hex 1000
HALT              ; Stop program
```

The corresponding machine code would be even more complicated and difficult to understand.

High level programming languages are much closer to natural languages such as English, and are therefore much easier for people to understand. They have a number of advantages over assembly language :

- 1) easier to program.
- 2) easier to understand, and therefore modify.
- 3) more portable (move from one machine to another).

The disadvantage of using a high level language is that the information must be translated to a form that the computer can understand before the program can be executed. Two methods of translation are used :

- 1) compilation
- 2) interpretation

When a compiler is used the complete source code (computer program) is translated and the object code is produced. The object code can then be linked with any required libraries and executed, and is independent of the source code. A program only needs to be re-compiled if it is modified.

When an interpreter is used the source code is taken one line at a time. Each line is translated and executed in turn. The program is not translated in its entirety, and is translated again each time it is executed.

Compilers produce code which runs more quickly and requires less memory than an interpreted program. The debugging of a program (identifying and correcting errors) is simpler when the language is interpreted.

A large number of high level languages are available, most of which are compiled languages. Most languages were developed with a particular application in mind.

Language	Derivation	Application
COBOL	C ommon B usiness O riented L anguage	Business and Commercial
FORTTRAN	FOR mula TRAN slation	Scientific and Engineering
BASIC	B eginners A ll-purpose S ymbolic I nstruction C ode	Teaching
Pascal	After the French mathematician	Teaching of structured programming techniques
C		Systems programming

These days, most applications are written in a high level language. Assembly language is only used in special cases where speed or memory requirements are critical.

An Approach to Writing Software

When the need for a particular program is identified, the software lifecycle begins. This lifecycle is made up of four main phases,

- 1) Analysis
- 2) Design
- 3) Implementation
- 4) Maintenance

The analysis phase is mainly involved in communication between the customer and the system analyst to decide what exactly is required of the computer program.

During the design phase, the results of the system analysis are used to decide how the requirements are going to be achieved.

The implementation phase involves taking the proposed design and writing the code so that the program can run on a computer. During this phase, the code is also tested and debugged. The final stage of the lifecycle is the maintenance phase. Maintenance is required for one of three main reasons,

- 1) removing bugs which were only discovered after the software had been delivered to the customer.
- 2) modifications are requested by the customer, e.g. the addition of new features to the program.
- 3) system changes, e.g. a new computer is to be used.

Software Design

The two main approaches which can be taken for the design of software are,

- 1) functional,
- 2) object-oriented.

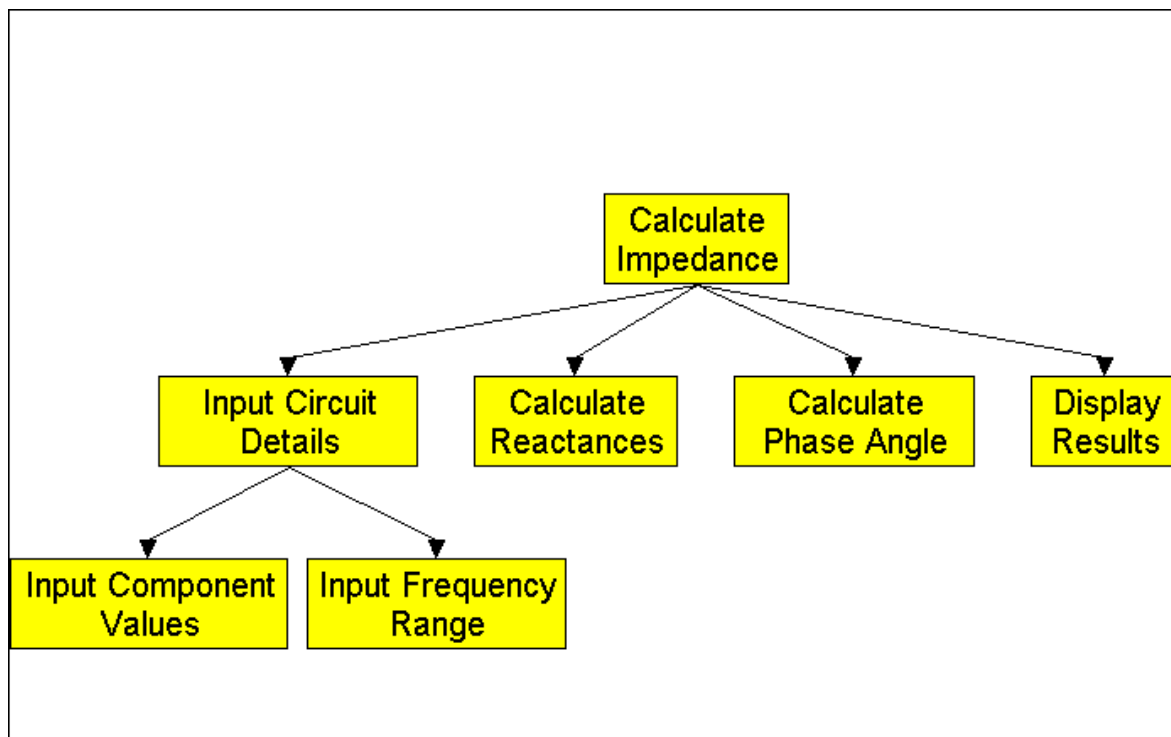
These approaches are not entirely independent, and are often considered to be complementary. The functional approach involves considering the system from the point of view of its function (what it is supposed to do). The object-oriented approach considers the system as a collection of objects which carry out particular tasks.

The design phase can be split into three components,

- 1) initial design,
- 2) detailed design,
- 3) data structure design.

The initial design is based on a top-down approach and functional decomposition. Firstly the overall purpose of the software is established. The next step is to sub-divide this into smaller units, which when combined perform the required tasks. Each of these smaller units can be

further sub-divided until each unit can be easily and thoroughly understood. At this stage consideration should also be given as to how the individual units are going to interact. The initial design can be represented by a hierarchy chart. An example is given below for a program which is to calculate the impedance of an RLC circuit across a range of frequencies. The hierarchy chart shown below would be further developed, adding more detail to each of the first level boxes, and possibly to some of the second level boxes.



This approach encourages the development of software as separate, well-defined modules. Well-written software is broken down into separate sections for a number of reasons :

- 1) easier to write.
- 2) easier to test.
- 3) easier to maintain.