Programming primer

Pay attention at the back! Eleanor Turton-Hill provides an overview of programming principles and common structures.

A computer program is nothing more than a sequence of instructions which are designed to make your computer behave in a certain way. There are literally hundreds, if not thousands, of programming languages and they are all different. Some have only superficial differences, while others differ hugely. Nevertheless, all programming languages have *some* things in common: they all attempt to describe the processing of data, and because of this they all share certain basic facilities.

Here, I'll be giving an overview of programming principles and common structures (and if you're about to embark on a programming course, this will give you a good head start). Once you've understood the basic principles of programming, you should be able to get to grips with practically any language.

What is a computer program?

Computers are not very intelligent things. They can't philosophise on the meaning of life, but they are very good at performing boring, repetitive tasks very quickly and very accurately.

Deprived of a program written by a human being, a computer can do nothing. If a program is badly written, it can very obediently turn perfectly good data into complete gibberish… just one small error can alter the behaviour of an entire application.

There are three basic facilities which all programming languages must have. Firstly, they must have some way of representing data and performing operations on it. Most provide some form of primitive data and structured data, and some allow you to create your own data types.

Secondly, programming languages must provide some kind of evaluation mechanism; some way of describing the way in which you would like the data to be transformed.

Thirdly, every programming language must have a set of naming and declaration rules. These rules state when you can and cannot refer to other elements of a program.

Constants and variables

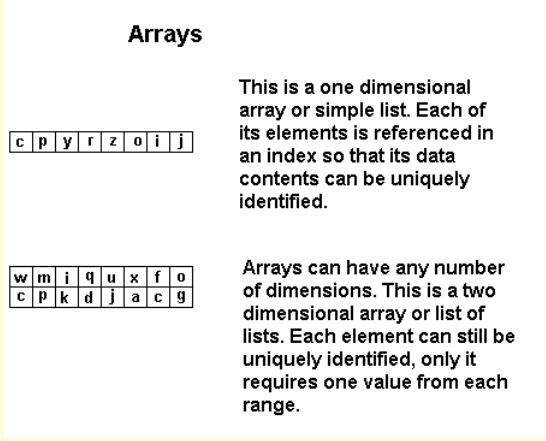
Most computer programs use some numbers which do not change their value throughout the duration of the program. These values are called constants and are usually declared at the beginning of the code.

The rate of VAT, for instance, is a figure which may be referred to many times in a program but always as the same value. This can be declared at the beginning of the code (e.g. VAT = 17.5). Then, every time you need to use the figure, you can simply refer to "VAT". What's more, when the rate of VAT alters, all you need to change is the constant declaration at the top of the code.

A variable, as you've probably guessed, is a value which changes throughout the program. The value of a variable can be altered and manipulated by the program, and certain operations may be performed, depending on its value. All constants and variables have names and values.

Data types

Before a variable can be referred to in code, it must first be declared and given a data type; the program needs to know the type of thing with which it is dealing.



Most programming languages acknowledge several data types, the major distinction being between text and numbers. In Pascal, there are four basic data types: char (for character strings), int (for integer values), and real (for double precision real numbers).

There is also a Boolean type which is used as a flagging mechanism. A Boolean variable can hold just one of two values like Yes/No, or On/Off, or 1/0.

Another data type used in most programming languages is the array type. This is used for managing lists. When an array is defined, it is given an index which enables you to uniquely identify any one of its elements *(see Arrays, above)*.

Controlling program flow

Every language has different conventions for beginning a program. In both C and Pascal, programs start with the reserved word "main". This makes it clear where execution should begin *(Fig 1)*.

The following text shows three basic control structures which are universal in procedural programming languages. Each structure has variations and each is written slightly differently, depending on the syntax rules of the language you're using. Here, I've made the examples simple and used pseudo-code to illustrate their structure.

Fig 1 A 'main' point

```
C                          Pascal
main () {                  PROGRAM main (input,output);
definitions;                       definitions
statements;                BEGIN
}                                   statements
                           END.
```

● IF THEN ELSE

The IF statement is probably the easiest programming structure to understand. It

What is object-orientated programming?

There's been much confusion recently about the meaning of the term "object-orientated". This is largely because it has been bandied about by all and sundry to mean a multitude of different things. Put simply, object-orientated programming is a collection of design principles for writing code. It is only supported by some languages and aims to break programs down into manageable units called "objects". The core idea behind this is to make components which are sufficiently general purpose as to enable them to be re-used in other programs.

This method of designing code yields many advantages. Firstly, a program which is divided into independent chunks is easier to understand, easier to debug and generally easier to maintain. Secondly, if many of those chunks are re-usable, time will be saved in future projects. Thirdly, an application made out of many independent parts can be more easily created by teams, thus increasing productivity.

The first object-orientated programming languages (Simula and Smalltalk) were conceived more than 20 years ago, but it's only recently that people have started taking its principles seriously. C++ is now the most popular object-orientated language. Objects within C++ can correspond to real-world entities such as bank accounts, employees or customers. But they can also correspond to computer hardware and software components such as communications ports, or video display windows, or data structures such as stacks or lists.

What are classes?

Many of the objects that a program uses have the same structure. A program which simulates the operations of a bank, for example, will need many account objects and many customer objects. Once the structure of an object has been set up, it is possible to produce many copies of it. This is done by using "classes"; each contains a complete description of one kind of object. Truly object-orientated code must have the three essential characteristics of inheritance, encapsulation and polymorphism. This may sound frighteningly technical, but in fact the whole thing rests on three fairly simple concepts.

One: classes can be defined from scratch, or they can be created by modifying an existing class. Derived classes take on all of the characteristics of the existing class, plus any modifications. This is called inheritance, and can save you an incredible amount of time and effort in code writing. Two: objects are available to the programmer through an interface which responds to a limited number of different kinds of message. The internal structure of individual objects is hidden from the programmer and this data hiding, or encapsulation, simplifies the use of objects. And three: a major attribute of an object-orientated language is that all the objects of the derived classes of a parent class are type compatible. This means that a derived class can be used anywhere that the parent class is expected. This is called inheritance polymorphism and enables clients of a family to see a simple uniform interface.

will execute one or other group of statements depending on the value of a condition. We use this structure in normal language all the time: "If it's sunny, we'll go out, otherwise we'll stay at home".

In code, it looks more like this:

```
IF condition true THEN
  instructions
ELSE
  instructions
END IF
```

● WHILE DO

The WHILE statement is iterative rather than conditional. It will execute a statement continually until a condition no longer holds. This translates to normal language something like this: "While John is well, he will keep working. If he is unwell, he will stop".

In code, it looks something like this:

```
WHILE Condition is true
  DO Instructions
WEND
```

● FOR..NEXT

The FOR..NEXT control structure is also a repeating routine. It is used to execute a single statement, a specified number of times: "For the next five days, I'll be going to work".

In code, it looks rather like this:

```
FOR
 n=1 TO 5
Instructions
NEXT
```

Structure

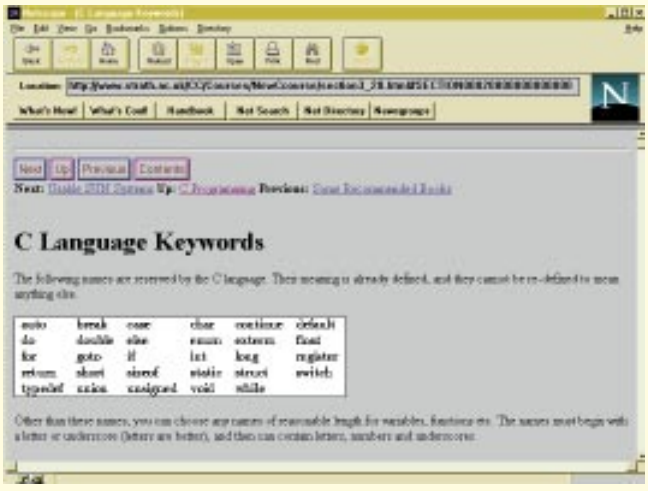
Once you've got used to the idea of variables, data types, and basic control structures, you're ready to start writing simple programs. But when your code starts to become more complex, then you'll have to learn about scope rules and structure.

It's easy to turn a perfectly good working program into complete garbage if you don't follow a few design principles. Your program may still work quite well but will gradually become unreadable and, worst of all, unmaintainable.

Over the years there have been many theories about how programs should be designed. The idea of the procedure emerged in the seventies with C and Pascal. It attempted to break code down into manageable and well-specified chunks, making it easier to write and maintain, especially by large teams. This "modular" style of programming, which is based on the idea of packaging data and functions, developed into what is now known as "object-orientated" code (*see the box, above*).

If you are thinking of learning a programming language, there are plenty of ways to get started. Turbo Pascal and Turbo C++ are both available from Borland, in DOS and Windows versions. Microsoft offers a Visual Basic Pro and Visual C++ Student Pack which you can get for a street price of about £80.

There are plenty of good programming tutorial pages on the internet. Check out this one, written by Steve Holmes of the University of Strathclyde, for some lessons in C: http://www.strath.ac.uk/CC/Courses/NewCcourse/ccourse.html



PCW Contacts

Eleanor Turton-Hill welcomes any feedback and suggestions from readers. She is at **ellie@pcw.ccmail.compuserve.com**

Borland 0990 561281
Microsoft 01734 270000