THE USE OF SUB-ROUTINES IN PROGRAMMES

D. J. Wheeler

Cambridge & Illinois Universities

A sub-routine may perhaps best be described as a self-contained part of a programme, which is capable of being used in different programmes. It is an entity of its own within a programme. There is no necessity to compose a programme of a set of distinct sub-routines; for the programme can be written as a complete unit, with no divisions into smaller parts. However it is usually advantageous to arrange that a programme is comprised of a set of sub-routines, some of which have been made specially for the particular programme while others are available from a 'library' of standard sub-routines. The reasons for this will be discussed below.

When a programme has been made from a set of sub-routines the breakdown of the code is more complete than it would otherwise be. This allows the coder to concentrate on one section of a programme at a time without the overall detailed programme continually intruding. Thus the sub-routines can be more easily coded and the tested in isolation from the rest of the programme. When the entire programme has to be tested it is with the foreknowledge that the incidence of mistakes in the sub-routines is zero (or at least one order of magnitude below that of the untested portions of the programme!)

If library sub-routines exist for the major part of a code then the task of constructing the remaining part of the programme is naturally very much less than if the code had to be written from the very beginning. However, one will rarely have available sub-routines to do exactly what is required and thus a certain amount of manipulation may be necessary before a given sub-routine can be used. Even so, it is usually far easier to use a sub-routine which will meet the specifications with a small amount of manipulation than to make one specially for the purpose.

It should be pointed out that the preparation of a library sub-routine requires a considerable amount of work. This is much greater than the effort merely required to code the sub-routine in its simplest possible form. It will usually be necessary to code it in the library standard form and this may detract from its efficiency in time and space. It may be desirable to code it in such a manner that the operation is generalized to some extent. However, even after it has been coded and tested there still remains the considerable task of writing a description so that people not acquainted with the interior coding can nevertheless use it easily. This last task may be the most difficult.

Besides the organization of the individual sub-routines there remains the method of the general organization of the library. How are the sub-routines going to be stored? Are they going to be stored on punched paper tape or are they going to be available in the auxiliary store of the machine? Usually it will be found that it is not possible to write the sub-routines such that they may be put into arbitrary positions in the store- although in certain machines this is now possible. Usually some translation process will have to be arranged so that an invariant form of sub-routine stored on some medium such as paper tape can be translated to the form required in a particular application. This translation is possible because fixed rules can be set up for adjusting a sub-routine so that it becomes correct in the set of locations in which it is put and used.

One next considers the methods by which sub-routines can be used. There are a number of different ways of transferring control to sub-routines and arranging that control is returned to the appropriate point to which it is required. One of the simpler methods was that used for the closed sub-routines of the EDSAC in which it was arranged that when the sub-routine had performed its part of the computation then control was returned to a point in the main programme immediately after the orders which had called it into use. This has been described in detail by Goldstine. This perhaps facilitates thinking of a sub-routine as an 'order' of the machine although it is usually of a more complicated kind than that wired in the circuits of the machine.

A second more interesting type of sub-routine is an interpretive routine. In this type of routine it is arranged that a sequence of operations is performed each time the sub-routine is called into action, each operation being determined by one parameter or 'order' in a list of such 'orders'. This type of sub-routine is particularly useful for coding certain special types of arithmetic for the machine,for example, floating point arithmetic in which numbers are expressed as $X \times 10^p$. Thus the sub-routine executes the 'orders' in the list in a similar fashion to the way that the machine obeys ordinary orders. However, the orders that it does are determined by the parts of the sub-routine, and so can be made to do any kind of operation or arithmetic.

One extension of an interpretive routine is a checking routine which is so arranged that the 'orders' that are obeyed are identical with those of the machine. However, the interpretive routine retains control and so it is possible to print out extra information about the course of the programme. This extra information makes it possible to follow the meanderings of the program in detail thus helping to locate the errors of a programme. This is not a good method of finding errors in programmes as it takes a long time and the programmers knowledge of the programme is not utilized - as it should be - in tracing the fault. However, it is a useful last resort and can quite often give out information about a code which would be difficult to find in any other way.

Sub-routines seem to have two distinct uses in programmes. The first and most obvious use is for the evaluation of functions, a simple example being the evaluation of sine x given x. The second use is for the organization of processes such as the integration of a function given f(x). This second type requires more consideration to make it useful and general. For instance how should f(x) be specified for the sub-routine? One obvious and useful way is to allow the integrating sub-routine access to an auxiliary sub-routine which is capable of evaluating f(x).

The above remarks may be summarized by saying sub-routines are very useful-although not absolutely necessary-and that the prime objectives to be born in mind when constructing them are simplicity of use, correctness of codes and accuracy of description. All complexities should-if possible -be buried out of sight.