THE UNIVERSITY OF NOTTINGHAM

QUAERENTI OSTIUM

SAPIENTIA URBS CONDITUR

FACULTY OF APPLIED SCIENCE

Applications of Computers

LECTURE 16.

"TECHNICAL PROBLEMS"

(2) AUTO-CODING

by

J. HOWLETT, B.Sc., Ph.D.

March 1958

AUTOCODES AND OTHER AIDS TO PROGRAMMING

INTRODUCTION

Although electronic computers are now becoming almost common, there is still a fairly wide acceptance of the view that programming is a black art, a practice too esoteric for the plain working scientist. Complementary to this is the view that only the largest-scale problems are suitable for machine computation, anything smaller being not worth the effort of programming. My experience has led me to very different views, and I feel strongly that these machines, which have most remarkable powers, should be made to work for us whenever possible; programming should be made easy enough for anyone of reasonable mathematical competence to learn quickly, and to make it worth while to use the machine for small problems. Just how far down in the scale of problem size one should go can be left for discussion, but I would say that it turns out very often indeed that what was started as "just a trivial little calculation" develops into an extensive survey, admirably suited to a machine. In any even moderately large organisation possessing a computer there will be many people, not machine experts, who need to make use of the machine and who therefore must either wait until a professional programmer has time to take up the work or write their own programmes; the professionals in every organisation are always under considerable pressure from all sides, so the second course may be the better one. However, writing a programme in conventional machine code, using the absolute address system of the machine, demands an intimate knowledge of the machine's logical structure and is likely to call for more concentrated care and attention to details than the occasional user can be expected to exert. Thus it seems to me essential that some simplification of computer programming is made, if only to make the machine easily accessible to many legitimate if not expert users. In fact, equally important benefits conferred by a well-designed scheme for simplification are the much increased speed with which even the experts can write programmes and the much greater ease of finding and correcting mistakes; both these increase the rate at which problems can be put to the machine and thus help to exploit its flexibility. Since one never gets anything for nothing one must expect to pay for these benefits and the cost is usually in computing speed; the likely cost is discussed later.

SOME APPROACHES TO PROGRAMME SIMPLIFICATION

It would be vastly inconvenient to have to write an entire programme in precise machine language and in fact this is never done, even by the most dedicated purist. Every machine installation is equipped with some at least of these aids to programming:- decimal conversion
- synthetic addresses

(i) an input scheme;

(ii) a sub-routine library, including standard programmes for reading and printing numbers;

(iii) a standard process for organising the entry to and exit from sub-routines.

The sub-routine library enables one to call in standard functions, such as Sin x, e^x, log x, without giving a thought to the processes by which they are produced. In a machine with a two-level store it is usually necessary to keep the sub-routines in the auxiliary store and bring them into the working store as required; this can be a quite elaborate process, and machines of the Manchester school use a "Routine Changing Sequence", a permanently-stored programme which organises all necessary link-setting and transfers between the two stores, on being given the necessary cue.

The input scheme can be of any degree of elaboration, and really defines the whole programming system so far as the machine user is concerned. In earlier machines quite simple input schemes were used, resulting in a very close correspondence between the written programme and what was actually stored in the machine: this has been preserved in the DEUCE computer. In general, input schemes have increased in complexity as time has passed, so a greater amount of "processing" is done on the instructions as the programme is read into the machine and the programme stored in the machine has become increasingly different from the written form. This has all been done to simplify the programmer's task, as for example by allowing him to use a simpler language for the writing of the programme which is translated into machine language by the input programme. One of the most important advances in this field is the idea of "floating addresses", due to Wilkes (Ref. 1, p. 127): briefly, this enables one to label strategically-important instructions in a programme, and refer only to these in control-jumps; the effect is that it is then easy to add or delete instructions between these control points without affecting the structure of the calculation - in contrast to programming in what is called "absolute address", in which the location of every instruction must be known. This idea of labelling is of extreme importance in Autocode programming.

In what follows I refer to "simplified", "automatic" or "Autocode" programming schemes. By such a scheme I imply a master programme which operates on the programme written in the simplified language to produce a programme in true machine language. Thus the method of working with such a scheme is this. The master programme is first put into the machine; the simplified-language programme is then read in and, as it goes in, is translated by the master into a machine-language programme which is stored; when this process is complete the working programme can be made to start the calculation at once or can be punched out on tape or cards (or written on to magnetic tape) for later use.

REQUIREMENTS FOR A SIMPLIFIED PROGRAMMING SCHEME

These are:-

(i) The programming language should be easily learned, readily intelligible and as close as possible to ordinary mathematics.

(ii) There should be as few ~~corrections~~ as possible.

(iii) There should be no restrictions on the sizes of numbers used anywhere in the calculation.

(iv) All the functions and operations one can reasonably expect to need should be provided.

(v) The scheme itself should detect the commonest types of mistake and should indicate what is wrong.

(vi) It should be efficient: that is, should not reduce the machine's computing speed unreasonably.

These imply that the machine itself should have these features:

(a) built-in floating-point arithmetic;

(b) a large rapid access store;

(c) a comprehensive set of instructions;

(d) a good complement of 'index' or 'B'-registers.

It is a truism that any machine can be programmed to do any computation; equally a programme can be written to cause any machine to produce a working programme from one written in any simplified language, but the loss in efficiency may be too high a price to pay for the increased ease in programming if the machine is basically too simple.

It is the trend towards fast, large-scale machines with all the above features that is making automatic programming an increasingly attractive idea.

EXAMPLES OF AUTOMATIC PROGRAMMING SCHEMES

Innumerable schemes of this kind have been devised: expert programmers are a race of individuals and the facts show that many would rather design a programming scheme of their own than accept one produced by another programmer: there is clearly a great fascination in this work. - I shall describe three systems which at least have the merit of being in regular use: two only briefly, the "Alphacode" for the English Electric DEUCE, and the matrix interpretation scheme for the Ferranti Pegasus; in more detail, the system I have used most, the Autocode written by R. A. BROOKER (Ref. 2) for the Ferranti Mercury.

I have left out what is almost certainly the most elaborate scheme in existence, FORTRAN (Ref. 3) developed by IBM for use with their Type 704 and 709 computers; an adequate account of this would take more time and space than I wish to use.

(1) DEUCE "Alphacode"

This was developed by the English Electric Company and is described in full in Reference 4. The introductory paragraph in this account states that the code "aims at providing a means of writing instructions for DEUCE in a simple and quick way, as near as possible to plain English; one which can be learnt in a short time and used without much practice. . . . "

The general principle is that all quantities are written as variables X_1, X_2, X_3 . . . ., up to X_{2500}, or integers $N_1 N_2$. . . . . . up to N_{63}. The computation is laid out as a series of statements of the type

$$X_p = X_r \text{ FUNCTION } X_s$$

or $$X_p = \text{ FUNCTION } X_s$$

The function can be any of the arithmetical operations + - x ÷ or any of the common algebraic or transcendental function, such as ROOT (for square root), COS, LOG, EXP etc.

For example to construct

$$y = e^{-ax} \cos bx/\sqrt{a^2 + b^2}$$

one would first replace y, x, a, b by, say X_1, X_2, X_3, X_4 and then write:

X_5	=	$X_3 \times X_2$	ax
X_5	=	$-X_5$	-ax
X_5	=	EXP X_5	e^{-ax}
X_6	=	$X_4 \times X_2$	bx
X_6	=	X_5 COS X_6	$e^{ax}\cos bx$
X_7	=	$X_3 \times X_3$	a^2
X_8	=	$X_4 \times X_4$	b^2
X_8	=	$X_7 + X_8$	$a^2 + b^2$
X_8	=	ROOT X_8	
X_1	=	$X_6 \div X_8$	

Notice that some of these instructions overwrite a variable with a new value, thus in effect increasing the number of variables available.

There are instructions for reading numbers into the machine and for printing results: for example, the instruction

3 DATA X_2

will read three numbers (punched on cards) and assign them to X_2, X_3, X_4 in order, thus setting values for x, a, b. If the last instruction is written

X_1 = $X_6 \div X_8$ P

FINISH

the machine will print (or punch) the value of X_1 and then stop.

The system works in floating-point arithmetic (programmed, because DEUCE is a fixed-point machine) and therefore requires no attention to scale on the part of the programmer. It provides a complete programming system, including instructions for organising cycles of operations and performing, on a single order, such elaborate processes as the integration of a set of differential equations.

2. The Pegasus Matrix Interpretation Scheme

Matrix operations are very commonly required in numerical mathematical work, the most familiar process being the solution of a set of linear simultaneous algebraic equations; all computer installations have found it essential to equip themselves at least with standard programmes for the routine performance of such tasks. Ferranti have developed a master programme for their Pegasus computer (Ref. 5), which enables such operations to be called in with very few orders, using floating-point arithmetic to take care of all scaling. The programme provides for input and output of matrices, addition, subtraction, multiplication, division and transposition. As an example, suppose we wish to solve a set of 20 simultaneous equations: if A is the matrix of coefficients and B the set of right-hand sides, the solution is C, where

$$C = A^{-1} B$$

A is a 20 x 20 matrix. B and C are 20 x 1 matrices.

In the Ferranti scheme each matrix is represented by an index number and its dimensions: thus if we call A matrix 1, it is described by (1, 20 x 20) - in general, (N, m x n) means "matrix number N, having m rows and n columns".

We call A, B, C matrices 1, 2, 3 respectively; the label 0 always refers to the input or output equipment. The programme is as follows, the instructions being punched on tape exactly as written here:-

(0, 20 x 20) $\rightarrow$ 1	Reads in A
(0, 20 x 1) $\rightarrow$ 2	B
(1, 20 x 20), (2, 20 x 1) $\rightarrow$ 3	C = A^{-1} B
(3, 20 x 1)(6) $\rightarrow$ 0	Output C in floating-point form to 6 digits.

These few orders organise the whole set of processes which are needed to solve the system. With such a scheme the solution of linear equations become an almost trivial operation.

3. Autocode for the Ferranti "Mercury"

A very complete scheme for use with this machine has been written by R.A. BROOKER of the Computing Laboratory at Manchester University; it is in large-scale use at Manchester and at Harwell. The design of this machine makes it very well suited to automatic programming on quite an elaborate scale.

The basic principle, as in the DEUCE scheme, is that a programme is written as a series of equations involving numbers, variables and functional operations; variables are represented by letters, with or without suffixes.

Thus $y = a_1x/b_1 + a_2y/b_2 + a_3z/b_3$

is an admissible instruction, meaning just what one would expect. Also is

$$y = \phi \sin (2\pi x/a)$$

where the symbol ϕ is needed as a warning to the master programme that the group of letters following, and up to the first bracket, defines a functional operation: the programme then examines these letters to find which sub-routine has to be called in.

The instruction

$$x = \phi \exp(-ax)$$

is interpreted as replacing the variable x by the new quantity e^{-ax}.

The Autocode vocabulary consists of the letters a b c d e f g h u v w x y z π (the last automatically set to 3.14159 ..... unless specifically altered) with or without suffixes, called variables and used for quantities of virtually unlimited size; i j k l m n o p q r s t called indices and denoting integers in the range 0-511; a number of functional orders; and a number of directives and control instructions for organising the layout and flow of the calculation.

To take a very simple example, suppose we wish to produce a table of cos θ and sin θ at 10^o intervals from 0^o to 360^o, with 6 decimals; and to record the values of $\sqrt{\cos^2 \theta + \sin^2 \theta}$; the programme is:

```
          r     =     0(10)180
          u     =     φ cos(rπ/180)
          v     =     φ sin(rπ/180)
          w     =     φ sq rt (uu + vv)
          print (r, 3, 0)
          space
          print (u, 1, 6)
          print (v, 1, 6)
          space
          space
          print (w, 1, 6)
          new line
          repeat
                      stop
```

Most of this needs no explanation. The instructions

```
          r     =     0(10)180
          repeat
```

are control instructions, and cause all the intervening instructions to be obeyed with r set to 0, 10, 20 - 180 in turn. The print instructions are all of the form

```
print (x, m, n)
```

meaning "print the value of x with m figures before the decimal point and n after; follow with 2 spaces". "Space" puts in an extra space and "new line" causes the next printing to start at the beginning of a new line.

As a variation, we might decide to examine the value of $\cos^2\theta + \sin^2\theta$ at each step and print only if it differs from 1 by, say, 5×10^{-7}; after the first three instructions the programme then becomes -

```
         w    =    ϕ mod (uu + vv - 1)
         print (r, 3, 0)
         space
         print (u, 1, 6)
         print (v, 1, 6)
    ┌──  jump 1,  0.000 0005 > w
    │    space
    │    space
    │    w    =    uu + vv
    │    print (v, 1, 7)
    └─> 1) new line
         repeat
         stop
```

The symbol 1) is a label; jumps, that is, departures from the sequential obeying of instructions, are always made to labelled instructions. Here we have a conditional jump, the instructions to print w being skipped over if $| u^2 + v^2 - 1 | < 5 \times 10^{-7}$. The general form of a conditional jump is:

```
jump (n),  a  =  b
      or   a  ≠  b
      or   a  >  b
      or   a  ≥  b
```

where a, b can be numbers, variables or indices.

All that it needed to make this set of instructions from a complete programme is the addition of some directives. To aid the organisation of large calculations, Brooker divides each programme into blocks of up to about 100 instructions, called Chapters and, so far as possible, coinciding with logical divisions of the calculation. Each chapter is headed, thus -

Chapter 7

and terminated by the directive close; the autocode instructions are translated into machine code as the programme-tape is read into the machine and when this final directive is received the assembly is completed and the working programme of the Chapter written on to the magnetic drum store, ready to be called down when required. The master programme then continues to read in more programme tape. Transfers of control from one chapter to another are made by instructions of the form

across m/n

meaning "read chapter n from the drum and start to obey it at the instruction labelled m". To start a calculation, the programme having been read in, one needs only an instruction directing control to the first instruction to be obeyed; this is obtained by following the "close" directive of the last chapter to be read in by the instructions

across m/n

close

This final "close" causes the "across" instruction to be obeyed immediately, and hence starts the calculation.

The present example has only one chapter; hence the complete programme is:

```
                    Chapter 1
          2)        r  =  0(10)180
                    u  =     cos(rπ/180)
                            .  etc
                            .
                            .
                    stop
                    close
                    across  2/1
                    close
```

If this is typed exactly as written on a standard Mercury-code tape perforator, a machine holding the master programme will accept it, perform the calculation, print out the results and then stop.

Most of the important features of the scheme are illustrated in this example. Numbers can be read from tape by instructions of the form

```
                    read (x)
```

A string of numbers, say 50, can be read in by a cycle:

```
                    i  =  1(1)50
                    read ($x_i$)
                    repeat
```

which sets x_1 = first number on the tape

x_2 = second " " " "

etc.

Numbers can of course be processed on the way in: if one wanted only the reciprocals of these numbers one could write

```
                    i  =  1(1)50
                    read ($x_i$)
                    $x_i$  =  1/$x_i$
                    repeat
```

If suffixed variables are to be used in any chapter, some information about these must be given at the head of the chapter; thus if one wanted to use $f_o, f_i \ldots$ up to f_{25} and $x_o, x_i \ldots x_{100}$, the chapter would start:

Chapter 3

$f \rightarrow 25$

$x \rightarrow 100$

and these directives would cause space in the fast store to be allocated to the variables.

As in the DEUCE scheme, there is an instruction which enables one to integrate any set of ordinary differential equations; behind the scenes this uses the Runge-Kutta process, in a form which is correct to the fourth order in the interval of integration.

There is a more elaborate example in Appendix I.

It will be clear that this system is simple, flexible and powerful; it is fast in operation because Mercury has the required built-in floating point arithmetic and other desirable features, so that loss of speed comes only from the use of general programme techniques in all cases, when in a particular case a technique tailored to fit the problem would be faster. Brooker estimates the reduction in computing speed is never as bad as a factor of 2, usually more like 1.2-1.5; the reduction in programme writing and development time is very great, probably a factor of at least 10. The main limitation on the use of the scheme - at least in its present form - is that it sacrifices some of the machine's flexibility; one has not the same control over the store as in standard programming, and it is not easy to perform logical operations.

The scheme has been used extensively at Harwell with wholly excellent results; at the moment at least half our machine time is taken up by autocode programmes, for problems of a very wide range both in type and in scale. Many scientists who in other circumstances would not have considered using the machine have become enthusiastic programmers and we have been able to have quite junior staff doing work which they themselves would have imagined beyond their powers. The result has been a very rapid progress of problems through the machine.

In spite of this eulogy I must finish with a warning; an automatic programming system, however excellent, does not remove the need for skilled programmers and people who really understand the machine - and the working of the automatic system. There always comes a time when things go wrong, and then chaos can develop unless there is someone to diagnose the trouble and put things right.

APPENDIX

The programme given here will evaluate the function

$$\psi(x,u) = \frac{1}{\sqrt{4\pi u}} \int_{-\infty}^{\infty} e^{-\frac{1}{4u}(x-y)^2} \frac{dy}{1+y^2}$$

for any given value of the argument x, u and to any stated accuracy (limited only by the machine's ability to carry not more than about 9 significant decimal digits). This function plays an important part in certain branches of theoretical nuclear physics.

The method used is direct numerical integration, using Simpson's rule, in the form

$$\int_{0}^{2h} f(x)dx = \frac{h}{3}\left[f_0 + 4f_1 + f_2\right] - \frac{h}{90}\delta^4 f_1$$

The programme adjusts the interval h at each step, always trying to increase it but keeping the fourth-difference term less than some stated quantity: it is assumed that the higher-order correction terms will then be negligible. It replaces the infinite limits by finite ones, determined so that the exponential term does not get less than e^{-25}, i.e. about 10^{-10}.

As given here, the programme reads in the values of x, u and the limit placed on the fourth-difference term; it computes the value of the integral, prints this and stops; it can be made to continue with another case by pressing the prepulse button on the Mercury console. It would be simple to alter the programme to make it produce a set of values of $\psi(x,u)$ for stated ranges of x and u.

A single calculation takes about 1.0 seconds.

```
    Chapter 1                         Notes

    y → 10                            Variable directions
    f → 10
    π → 10
                                      Restarted by prepulse
1)  stop
    read (x)
    read (u)
    read (e)                          Limit for  hδ^4/90
    new line
    print (x, 2, 4)
    print (u, 2, 4)
    new line
    new line
    z = 0                             Starting value for integral

    h = ϕ sq rt (u)
    2 = ϕ sq rt (4πu)
    2 = 1/π2                          1/√4πu
    3 = 0.25/u                        1/4u

    a = x - 10h            )          Effective limits of
    b = x + 10h            )            integration

    π1 = a
2)  i = 0(1)4
    yi = π1 + ih - h
    y = yi
    n) = 3                 )          Arranges entry into sub-rputine
    jump 20                )          for computing integral, and return

3)  fi = f
    repeat

    c = fo - 4f1 + 6f2 - 4f3 + f4        δ^4 f2
    d = ϕ mod (hc/90)
    jump 4,  e > d
    h = 0.5h               )          Halves interval if  hδ^4/90
    jump 2                 )            exceeds limit

4)  v = f1 + 4f2 + f3      )          Integrates over interval 2h
    z = z + hv/3-hc/90     )

    y = y3 + 2h            )          Tests if upper limit of integral
    jump 6,  b > y         )            is within 2 h

    h = 0.5b - 0.5y3                  Last interval;  link for return
    n) = 5                              from sub-routine
    i = 4(1)5
    yi = y3 + ih - 3h
    y = yi
    jump 20

5)  fi = f
    repeat
    v = f3 + 4f4 + f5
```

```
        z = z + hv/3
        print (z, 2, 6)                 )    Prints result and
        new line                        )     stops
        jump 1

   6)   y = y3 + 4h                     )    Tests if doubling
        jump 7,  b > y                  )    interval would over-
        jump 8                          )    run upper limit.

   7)   h = 2h                          )    Doubles interval
   8)   π1 = y3                         )    and continues.
        jump 2

  20)   w = x - y                       )    Subtraction for
        w = ϕ exp (-π3wv)               )    evaluation integral.
        f = ϕ divide (π2v,  1 + yy)
        jump (n)

        close                                End of chapter.

        across 1/1                           Starts calculation
        close
```

<u>Data:</u> e.g.

1.0	x
0.5	u
0.0000005	e

<u>N.B.</u> The first order obeyed (1/1) stops the machine, so the operator has time to insert a data tape.

REFERENCES

1.	WILKES, M.V.	Automatic Digital Computers Methuen (1956).
2.	BROOKER, R.A.	Computer Journal 1 (1) April 1958 pp. 15-22.
3.	I.B.M. CORPORATION	FORTRAN: Programmer's Reference Manual.
4.	ENGLISH ELECTRIC CO. LTD.	Report NSy 87 (January, 1958).
5.	FERRANTI LTD.	List CS 135 (March, 1957).