thus

$$y = (S_1 + S_2)x^3 - (2S_1 + S_2)x^2 + S_1x$$

x is allowed to take on a series of values between 0 and 1 and the corresponding values of y are calculated from the equation of the cubic. x and y are multiplied by r to convert to actual distances.

Consider the point P_c thus found. From the geometry of the figure

$$Y_c = Y_1 + rx_c \sin\theta + ry_c \cos\theta$$

$$\sin\theta = \frac{Y_2 - Y_1}{r}\,;\ \cos\theta = \frac{X_2 - X_1}{r}$$

$$\therefore\ Y_c = Y_1 + x_c(Y_2 - Y_1) + y_c(X_2 - X_1)$$

similarly

$$X_c = X_1 + x_c(X_2 - X_1) - y_c(Y_2 - Y_1)$$

X_c and Y_c are the co-ordinates of the point in the practical XY system.

To the Editor
The Computer Journal

Sir

Reading variable length records using FORTRAN IV

The letter by Messrs Hathaway and Van Viliet (page 190, May 1974), led me to reconsider the method I had been using to read variable length records from disc, using FORTRAN IV. The records consist of a major segment, with two types of minor segment, each of which may occur a variable number of times. The major segment includes two-byte count fields for each minor segment type. Maximum record length is about 3,400 bytes and average lengths about 1,300 bytes.

The method of reading, which has been in use in this recently developed program, is the most obvious, that is, an indexed READ, viz.:

```
READ (9, 110, END = 500) MAJOR, COUNT1, COUNT2,
    SPARE, ((COUNTR(I, J), I = 1, 91), J = 1, COUNT1),
    ((COMNTS(I, J), I = 1, 78), J = 1, COUNT2)
```

A formatted READ was used for no better reason than convenience of handling the data once read in.

Since this FORTRAN program produced CPU times of the order of 30 secs, whereas reports from the same file using Informatics MARK IV produced times of the order of a few seconds only, I was concerned at its apparent inefficiency.

By way of experiment, I wrote four small programs to test the various possible methods of reading variable length records in FORTRAN. Two of the methods used an indexed READ of the kind referred to above, one formatted and one unformatted. The other two methods were based on the letter of Hathaway and Van Viliet, and used a FORTRAN subroutine with run-time dimensioning of a buffer array, viz:

```
      SUBROUTINE READ (ARRAY, SIZE)
      DIMENSION ARRAY (SIZE)
      READ (9, 20) ARRAY
   20 FORMAT etc.
```

Again, these two methods used formatted and unformatted READ statements in the subroutines.

The main problem in using this latter method is that the record size is unknown until the READ is performed. This was solved by reading through the file once, to get the two count fields from each record, which were stored in arrays. The file was rewound and read again, using the stored count fields to calculate the size of each record before reading it using the subroutine described.

The four programs were used to read the same file of 100 of the variable length records referred to, from a Type 3330 disc on an IBM 370 Model 155 using the G-level FORTRAN compiler. The CPU times were as follows:

	CPU Time (secs)	
	Formatted	*Unformatted*
Indexed READ	20·3	12·3
Subroutine	11·8	0·7

The result for the unformatted READ in the subroutine was so striking that the operation of the program was tested and the run repeated, with the same result. Clearly it must be possible to get below even 0·7 secs, because that program contains the preliminary read of the file to establish the record sizes.

To remove the need for this preliminary read, a subroutine was written in Assembler to read the next record from the file into a specified array. Using this subroutine in a small FORTRAN program a CPU time of 0·4 secs was achieved for the same file of 100 records.

It is clear that very worthwhile savings of CPU time can be made, at the expense of slightly more complicated coding, and that variable length records can be handled quite efficiently by FORTRAN, even without specially written subroutines.

I would like to thank Dr. K. M. Howell for drawing my attention to the original letter and Mr. D. J. Wilson who wrote the Assembler subroutine.

Yours faithfully,

D. F. ARTHUR

Management Services Department
ICI Fibres
Hookstone Road
Harrogate
Yorkshire HG2 8QN
29 November 1974

To the Editor
The Computer Journal

Sir

Check digits and error correction

The letter from Dr. A. M. Andrew (*The Computer Journal*, Vol. 17, p. 382) was of interest to me because I have been looking at this subject in connection with a specific problem, namely how to reduce the number of computer runs needed to achieve 100 per cent accuracy in updating a set of files. If even one error is detected, a re-run is needed. In the case I have investigated, there is a record key of six digits and I tried using two check digits together with an error-correcting facility if any of the resulting eight-digit input numbers failed to check.

The results were encouraging though they failed to achieve a 100 per cent correction rate even for single-digit errors. The check digits were defined by

$$\sum_{i=1}^{8} d_i f_i \equiv 0 \text{ (modulo 100)}$$

where the d_i are the digits of the eight-digit number and the f_i are weighting factors.

The optimum values of the f_i were found semi-empirically. They are: 43, 47, 9, 31, 39, 37, 10, 1. This set of factors leads to at most two possible corrections to be tested for any non-zero check sum. Even when there are two possibilities, one may be ruled out if it would lead to a corrected digit which is either negative or greater than 9.

The order of these factors was determined by optimising the chance of correcting other types of errors if single-digit errors happened not to satisfy the particular check sum found. These are:

1. Any number of consecutive digits in error by the same amount,
2. Interchange of adjacent digits,
3. Interchange of the outer digits in a group of three.

In a test of the method, a set of 1,000 eight-digit numbers was generated from a pseudo-random sequence of six-digit numbers with two check digits attached. These were given to two punch operators, one experienced and the other a novice. The first girl made only five errors, all of which were detected and corrected automatically.

The second girl had errors of several different kinds which were difficult to analyse statistically, notably 'offsets' where the correct number is punched in the wrong columns. The test corrected 62 out of 82 wrong eight-digit numbers but also mis-corrected 18 including 1 out of 63 single-digit errors. (308 188 60, punched 301 188 60, mis-corrected 301 187 60).

My conclusion is that, although not 100 per cent effective, this method can be very useful in reducing the number of computer re-runs caused by punching errors as long as the error rates are low to start with and 'offset' errors are unlikely.

The economics of using the technique depend partly on the cost of