Since we know $\frac{p}{q-p}$ and can estimate β, a decision may be made that is based entirely on the value of α, the load factor of the open table. Thus, each assembler or compiler may have critical load factors, α_L and α_u, which may be computed and stored as constants, and the decision to chain may be made when $\alpha_L < \alpha < \alpha_u$. (Note that it is possible to find that $\alpha_L \geqslant \alpha_u$ in which case chaining would never be profitable.) For most practical cases, we would consider utilising this technique when we had free bits available for the link field within each symbol table entry, that is,

$$\frac{p}{q-p} = 0.$$

Here we would chain whenever α_L becomes greater than the critical value shown in **Fig. 10** (α_u is, of course, 1·0).

β	α_L
1·0	·90
1·5	·85
2·0	·80
2·5	·77
3·0	·74
4·0	·70

Fig. 10

Any applications which require the use of both open hash tables and list structures can be optimally structured using the techniques described in 4.1. Such applications include string processors and multilist or ring-structured information retrieval systems. Similar applications are suggested which employ the techniques described in 4.2. In fact, the author is aware of an efficient sorting algorithm that utilises both calculated address tables and tree structures, which are embedded directly onto each other in the sort area. In this technique the tree roots form the head nodes in a chained hash table, and the other entries in the trees are considered as further entries within the hash table. Thus, all entries hashing to the same location are grouped together in a sorted tree whose non-root entries may be moved around to make room for newly created roots. Insertion is somewhat more complicated than algorithm *G*, since it involves searching down a tree, but otherwise, the techniques given in 4.2 are employed. The utility of this sort is that for n random entries, the sort time is linearly proportional to n, even when the entries (and their links) completely fill storage.

Only a few applications have been mentioned here, but it is obvious that useful and innovative processes can be developed by carefully utilising the techniques described within this paper.

References

BAYS, C. (1973). A Note on When to Chain Overflow Items within a Direct-access Table. *CACM*, Vol. 16, No. 1, p. 47.

BELL, J. R. and KAMAN, C. H. (1970). The Linear Quotient Hash Code. *CACM*, Vol. 13, No. 11, pp. 675-677.

BLACKE, L. F., LAWSON, R. E., YUILLE, I. M. (1970). A Ring Processing Package for use with FORTRAN or a Similar High-Level Language. *The Computer Journal*, Vol. 13, No. 1, pp. 40-47.

DODD, G. G. (1969). Elements of Data Management Systems. *Computing Surveys*, Vol. 1, No. 2.

HIGGINS, L. D. and SMITH, F. J. (1971). Disc Access Algorithms. *The Computer Journal*, Vol. 14, No. 3, pp. 249-253.

HOPGOOD, F. R. A. (1969). *Compiling Techniques*. American Elsevier Publishing Company, New York.

JOHNSON, L. R. (1961). An Indirect Chaining Method for Addressing on Secondary Keys. *CACM*, Vol. 5, No. 8, pp. 218-222.

KNUTH, D. E. (1968). *The Art of Computer Programming* (Volume 1 *Fundamental Algorithms*). Addison-Wesley, Reading, Massachusetts.

MAURER, W. D. (1968). An Improved Hash Code for Scatter Storage. *CACM*, Vol. 11, No. 1, pp. 35-36.

MCILROY, M. D. (1963). A Variant Method of File Searching. *CACM*, Vol. 6, No. 1, p. 101.

MORRIS, R. (1968). Scatter Storage Techniques. *CACM*, Vol. 11, No. 1, pp. 38-42.

Book review

Introduction to Computers and Computer Programming, by Samuel Bergman and Steven Bruckner, 1972; 433 pages. (*Addison-Wesley*, £4·90)

This book was developed from the notes for a series of introductory programming courses given at the University of Pennsylvania and Temple University. It is unlikely that this book will be of great interest to British universities as it is too long for an introductory text and has insufficient depth for a computer science course.

The first eight chapters, about 230 pages, are devoted to teaching the basic principles of machine code and assembly language by using an imaginary computer FACET (FAcility for Computer Education and Training). This is a small (1K) simulated decimal machine with a single accumulator and index register and instructions for handling integers, characters and floating point numbers. The instruction set is well constructed and a variety of basic concepts are skilfully illustrated within the limited framework chosen.

A chapter entitled 'A Look at Other Computers' links the FACET part of the book to the section dealing with FORTRAN. This deals with the representation of numbers (particularly binary), multiple register machines, paging and peripherals.

The authors' foreword states that 'the FORTRAN presentation is more thorough than the standard approach because students . . . can already program'. I found their 'thorough' approach unpalatable and lacking a sense of purpose. Instructions were introduced in quick succession without any thought to orderly development and illustrated by quoting comparable sequences of FACET assembly language instructions.

The general presentation of the book is good, each chapter containing a review section emphasising the main points of the chapter. A large number of problems are provided ranging from well disguised trivia to several problems which could (and did!) prove difficult to an experienced programmer. Appendices cover the FACET assembly language and control cards, FORTRAN syntax and FORMAT examples and debugging, which draws together ideas from various chapters.

The FACET simulator (written in FORTRAN) and an instructors' manual are available from the authors. However, I was unable to obtain these in time to include comments on them in this review.

R. C. WELLAND (Reading)