different set of marks can be produced merely by changing the assignment to p.

If permutations of the natural numbers are required, all three new procedures could be improved slightly. The variable *temp*, is redundant, its value being known. For example, in the recursive procedures on entry to permute at level k, $p[k] = k$.

To return to the general case, the specification of the procedure could have been altered to include p as a parameter. This has not been done because it is not clear that either structure is the best.

The family of procedures described here have a common structure. A procedure is called once and generates all the permutations, these permutations being processed by code embedded within the procedure or by subroutine. The procedures described by Ord-Smith have all been cast in a form in which the permutation procedure is called $n!$ times as a subroutine of the procedure processing the permutations. This suggests that all the procedures might better be recast as co-routines.

Conclusions

In this paper we report the results obtained by accepting an algorithm and taking a programmer's view of the procedure implementing this algorithm. Gains of between 40 to 55 per cent have been obtained. However, when we compare Fig. 4 with Fig. 7 we see that the improvements are very much machine and compiler dependent. If we rank the five procedures for the two systems considered, we produce two different rankings. On the 1906A, the best procedure, marginally, is a recursive one: on the 7600 it is, by over 30 per cent, a non-recursive one.

That the recursive procedure on the 1906A should prove the fastest is of significance in that it supports Fike's observations about his original algorithms on IBM 360. The figures suggest that the much quoted assertion that recursion is inefficient (which the figures for 7600, on their own, would support) is an assertion more about the compiler used than about recursion itself. If procedure entry and exit is handled sensibly, as it is on the 1906A compiler used, then recursion is not expensive, and, where the algorithm involved is essentially recursive, can be used to advantage.

Acknowledgements

I should like to record my thanks to C. T. Fike who, on reading the first draft, pointed out a serious weakness in the procedures, and to Mrs. E. M. J. Chadwick who tested and timed them all.

References

FIKE, C. T. (1975). A permutation generation method, *The Computer Journal*, Vol. 18, pp. 21-22.
ORD-SMITH, R. J. (1970). Generation of permutation sequences: part 1, *The Computer Journal*, Vol. 13, pp. 152-155.
ORD-SMITH, R. J. (1971). Generation of permutation sequences: part 2, *The Computer Journal*, Vol. 14, pp. 136-139.

Book review

The Principles of Systems Programming, by Robert M. Graham, 1975; 422 pages. (*John Wiley*, £8·25)

In writing this book, the author has tackled a formidable task. The phrase 'systems programming' is taken, particularly in the USA, to include almost all programs except those specifically directed towards applications. The book is correspondingly long, (about 180,000 words) and contains several major sections:

An introduction to systems	(30 pages)
Machine and assembly languages, assemblers, macroprocessors and loaders	(92 pages)
Programming languages and compilers	(113 pages)
Operating systems	(130 pages)
Appendices	(40 pages)

The book has many appealing features. A strong unifying thread is the consistent use of INSTRAN, a 'private' high-level language somewhat similar to Pascal or ALGOL 60, throughout most of the main sections. INSTRAN serves to describe all the algorithms used for assembly, loading, compilation, scheduling and peripheral operation, and itself forms the example for an extensive discussion on language translation techniques.

The text is always readable, and markedly fresher at the end than at the beginning of the book, where many of the sentences are cumbersome and repetitious. The best part of the book is the last section on Operating Systems. My guess is that this is what really interests the author. The section begins with an introduction which singles out and describes some of the important characteristics of a modern multi-access system such as its response ratio, file storage, arrangements for privacy and protection, and adaptability to change. Next, there is a good chapter on process control and communication. The writer describes both the classical P and V operations and the alternative 'block', 'wake-up' and message passing mechanisms. The chapter on memory management covers segmentation and paging (without confusing the two) and the organisation of file stores. The next chapter deals with input and output of all kinds, giving several device driver routines in INSTRAN. The section ends with a short chapter on the sharing of information and protection against unauthorised access.

The rest of the book filled me with somewhat less enthusiasm. For all that the text is described as 'machine independent', it is clear that most of the writer's experience was gained on the IBM 360 series and on the various incarnations of MULTICS. Thus, although very little is said about job control languages, those samples which are presented are derived from OS 360. The author sees nothing wrong with the seven job control statements needed to compile and run a simple program in PL/I, and nowhere does he suggest that a JCL might be a genuine programming language instead of a series of rigid primitive commands. The lack of any discussion on this point, and of any suggestion that the user interface might be an important aspect of an operating system, are two unfortunate omissions from the text.

The section on assemblers and loaders again seems to lack breadth. No mention is made of a one-pass assembler, and the various details, which are exceedingly precise, refer quite specifically to the IBM 360 computer. Similarly, the section on compilers to a specific (and not generally known) language, and such general information as it contains is better presented in other books such as those by Gries or Rohl.

The book gives only 19 references, and they are scattered throughout the text instead of being collected together as is normal. However, the names of the authors given—Dijkstra, Wirth, Gries, Brown, Knuth, Denning, etc. are reassuring.

In conclusion, the section on Operating Systems will make good background reading for Computer Science students, and those who teach assembly code programming in the context of the IBM 360 might find that section of the book a possible alternative to the standard IBM documentation as source material. The book will be a useful addition to a college library, but—in view of the price—the student would do well to spend his book allowance elsewhere.

A. J. T. COLIN (Strathclyde)

Downloaded from https://academic.oup.com/comjnl/article/19/2/159/408728 by guest on 19 April 2024