# EXHIBIT H

# ENCYCLOPEDIA OF COMPUTER SCIENCE AND ENGINEERING

SECOND EDITION

ANTHONY RALSTON, EDITOR
EDWIN D. REILLY, JR., ASSOCIATE EDITOR

VNR VAN NOSTRAND REINHOLD COMPANY
NEW YORK CINCINNATI TORONTO LONDON MELBOURNE

Library of Congress Catalog Card Number: 82-2700
ISBN: 0-442-24496-7

Manufactured in the United States of America

Published by Van Nostrand Reinhold Company Inc.
135 West 50th Street
New York, New York 10020

Van Nostrand Reinhold Company Limited
Molly Millars Lane
Wokingham, Berkshire RG11 2PY, England

Van Nostrand Reinhold
480 Latrobe Street
Melbourne, Victoria 3000, Australia

Macmillan of Canada
Division of Gage Publishing Limited
164 Commander Boulevard
Agincourt, Ontario M1S 3C7, Canada

15 14 13 12 11 10 9 8 7 6 5 4

QA76.15
.E48
1983

**Library of Congress Cataloging in Publication Data**
Main entry under title

Encyclopedia of computer science and engineering.

Includes index.
1. Computers—Dictionaries. 2. Electronic data
processing—Dictionaries. 3. Information science—
Dictionaries. I. Ralston, Anthony. II. Reilly, Edwin D.
QA76.15.E48 1982 001.64'03'21 82-2700
ISBN 0-442-24496-7 AACR2

In addition to refining the basic functions of gathering data, current development in hardware monitors shows a trend toward using microprocessors for processing data during collection and for allowing the host computer and the monitor to alter each other's measurement functions during operation. From the user's viewpoint, the principal differences between hardware and software monitors are:

1. Software monitors can provide more information on cause and effect by relating measured data to the program steps being executed; however, care must be exercised to avoid disruption of time relationships caused by the addition of the measurement programs.

2. Hardware monitors only measure electrical events at predetermined physical points; hence, it is more difficult to relate measurements to program activity. However, with reasonable care, data may be gathered without interfering with the system being measured.

REFERENCES

1978. Ferrari, D. *Computer Systems Performance Evaluation.* Englewood Cliffs, NJ: Prentice-Hall, pp. 32–40.

1979. Borovits, I. and Neumann, S. *Computer Systems Performance Evaluation.* Lexington, MA: Lexington Books, D. C. Heath and Co., pp. 39–56.

1981. Plattner, B. and Nievergelt, J. "Monitoring Program Execution: A Survey." *IEEE Computer* **14,** No. 11 (November).

J. D. NOE

## HASHING

For articles on related subjects *see* SORTING; and TABLE LOOKUP.

Hashing (or *hash coding*) is a word coined by computer programmers to describe a general class of operations done to transform one or more fields (usually a *key*) into a different (usually more compact) arrangement. Probably, "hashing" was first coined because it seemed that "hash" was being made out of integral pieces of data. The rationale for hashing is developed more fully in the article on table lookup, dealing with key transformation. The justification for hashing derives from being able to convert naturally occurring, diverse, ill-structured, scattered key fields into compact, easily manipulated fields—usually some numeric, computer-oriented field such as a word or double word, or a computer memory address to facilitate subsequent references. The transformation from the natural field to the hash address is only a one-way process, however; the natural field cannot be decoded or reconstructed from the hash. Also, the hashed field may not represent only one unique natural field; many natural fields could hash into the same value.

For example, suppose there is a table of automobile part numbers that are ten numeric characters in length, but there may be no more than 10,000 unique part numbers. In order to contain every possible number, the table would have to allow 10 billion ($10^{10}$) positions to handle only $10^4$ possible keys. A scheme can be contrived to transform the original ten-digit key to an integer that will represent the position of that part in a much more compact table.

One simple scheme for hashing is the division-remainder method: Choose a number close to the number of table positions needed. Use that number as a divisor to extract a quotient and a remainder from the dividend (which is the original key). The remainder so obtained is the transformed key. Using 10,000 as the divisor, the transformed key becomes the original key modulo 10,-000. Some examples follow.

| Original Key (Part Number) | Transformed Key |
|---|---|
| 00 0000 1000 | 1000 |
| 00 0001 0000 | 0 |
| 00 0001 0001 | 1 |
| 00 0001 0099 | 99 |
| 10 0001 0099 | 99 |
| 22 3333 4444 | 4444 |
| 90 0020 0110 | 110 |
| 99 0020 0112 | 112 |

The examples in this table were constructed to illustrate the occurrence of duplicate transformed keys. In such schemes, prime divisors are normally used in practice.

Ideally, the hashing scheme would convert the original keys to transformed keys with no duplicates. While schemes can be constructed to minimize *collisions* (*hash clash*) their possibility cannot be eliminated completely and, because of this, the original key must be stored in the table. Further, some scheme must be used to handle duplicate transformed keys.

The examples and discussion in the article on table lookup will further describe methodology and rationale for hashing. Some other techniques in addition to division-remainder are: (1) folding, (2) radix transformation, and (3) digit rearrangement.

*Folding* consists of splitting the original key into two or more parts, then adding the parts together (or, sometimes, using the exclusive OR operator). This sum, or some part of it, is then used as the transformed key. For example:

Original key = 20 2152 9396
Splitting and adding: 20 + 2152 + 9396 = 11568
Discard high-order digit to obtain four-digit transformed key of 1568.

*Radix transformation* involves changing the radix or base of the original key and either discarding excess high-order digits (i.e., digits in excess of the number desired in the key) or extracting some part of the transformed number. For example, an original key of 12345 (base 10) could be considered a base-16 number, and would be transformed as follows:

$$(1 \times 16^4) + (2 \times 16^3) + (3 \times 16^2) + (4 \times 16^1) + (5 \times 16^0) = 74565.$$

The four-digit key would be 4565 by discarding the high-order excess digit(s).

*Digit rearrangement* consists simply of selecting and shifting digits of the original key. For example, an original key of 1234567 could be transformed to a four-digit key of 6543 by selecting digit positions 3 through 6 and reversing their order.

No one technique is necessarily superior to another in general; however, for specific applications, some may work better than others. The selection of a technique should involve consideration of which technique results in fewest duplicate hash keys.

*Hash* totals are sometimes used for purposes of checking or verification; in this context, hashing has a different purpose than key transformation, inasmuch as the totals may not necessarily be hashed or scrambled. The use of hash totals is for a purpose much like the use of parity bits or self-checking codes for representing characters in digital form on media such as magnetic tapes or punched paper tapes. When such data is written (or recorded), hash totals are generated and written along with (usually after) the data. Then, when the data is read, the hash totals are recomputed, using the same algorithm, and checked against the ones recorded. If they agree, one can be more certain that the recorded data read is identical to that written and that no bits have been lost or misread.

For example, if a hash total is taken after every five numbers, that hash total could be recorded (written) after the five numbers, thus:

| | |
|---|---|
| Five data numbers | 12345 |
| | 37654 |
| | 89701 |
| | 00378 |
| | 42270 |
| Total | 182348 |
| Discard excess to obtain "hash" total | 82348 |

Now, upon reading this data and its hash total during some subsequent process, one could recompute the hash total in the same way and thus verify that it matched the one originally recorded.

C. E. PRICE

## HEURISTICS

For articles on related subjects *see* ALGORITHM; ARTIFICIAL INTELLIGENCE.

The ancient Greek word *heuriskein* means "to find out, to discover." The English adjective "heuristic" and the more recently coined noun "heuristics" came into being via the Latin adjective *heuristicus*. According to the *Random House Dictionary:*

> *heuristic* adj. *1.* serving to indicate or point out, stimulating interest as a means of furthering investigation. *2.* (of a teaching method) encouraging the student to discover for himself.—n. *3.* a heuristic method or argument.

In the general sense, we talk of the "heuristic power" of a technique, the "heuristics in somebody's reasoning," and so on. Pólya (1954) has written several most entertaining books that do not teach, but do make one realize how to approach problems in mathematics and geometry via heuristic ideas. Also, Hadamard's essay (*The Psychology of Invention in the Mathematical Field*, Dover, 1974) on discovery in mathematics yields an interesting insight—a much too rare phenomenon—into how one of the great mathematicians of all time tackles problems.

How does all this concern us in computing? The reason is simple but its application leads to an area that is completely open-ended. Let us consider, for example, a standard task in programming. We wish to find the roots of a higher-order algebraic equation. There are several methods of approximation that yield the solution with estimatable error bounds. We have the formulas to follow, step by step, and eventually we obtain the results. This is the *algorithmic approach*.

Let us now consider a so-called ill-defined problem, and we have many of them in everyday life. For example, say we want to balance our household budget by following a program. Although our basic needs are reasonably well known (food, shelter, clothing, medical items, transportation, entertainment, etc.), neither the relative weight of the components nor their unit prices are determinable completely. Also, our needs, desires, and tastes