omitting the forms and pointing to the first of these two triples. For Variation 2, only the form itself need to be stored.

Denote by v the number of memory accesses required for locating one item in a hash table using a specific hash function. This includes the additional accesses required for resolving hash collisions by methods such as chaining or double hashing. Then the number of memory accesses for retrieving w is $t = (2\,|w| + 1)v$ for Variation 1, $(|w| + 1)v$ for Variation 2, $3\,|w|\,v$ for Variation 3, and $4\,|w|\,v$ for Variation 4. The hash dictionary can be stored in an almost full hash table with a good average and worst case v by using a method such as that proposed by Schmidt and Shamir [6]. Since the same operators are calculated for every word, assembly language routines or even microcoding of them can be prepared, thereby reducing the CPU cost. On the other hand, more "collisions" than in normal hashing can be expected: whenever two distinct dictionary words are transformed into the same string by our operators, both of them are stored, since they are induced by different dictionary words. The problem of locating them is of course taken care of automatically by the collision handling mechanism associated with the hash function, but the number of collisions increases. We have not investigated this effect; instead we wish to thank the referee for pointing out the desirability of doing so.

The RED method can also be extended to detect other types of errors, which are not single errors but occur frequently in optical character recognition, such as changing one character into two other characters (horizontal splitting); changing two characters into one other character (catenation); changing two characters into two other characters (crowding). (The terms are from [5].) This can be achieved, for example, by storing $D_i^2(x)$ $(1 \le i \le |x| - 1)$ in the hash table for every significant word x.

Received 4/81; revised 8/81; accepted 3/82

References

1. Bratley, P., and Choueka, Y. Processing terms in document retrieval systems. To appear.
2. Damerau, F.J. A technique for computer detection and correction of spelling errors. Comm. ACM 7, 3 (Mar. 1964), 171–176.
3. Mor, M., and Fraenkel, A.S. Retrieval in an environment of faulty texts or faulty queries. In *Proc. 2nd Int. Conf. Databases—Improving Usability and Responsiveness*, Jerusalem, June 1982, Academic Press, New York.
4. Peterson, J.L. Computer programs for detecting and correcting spelling errors. *Comm. ACM 23*, 12 (Dec. 1980), 676–687.
5. Rosenbaum, W.S., and Hilliard, J.J. Multifont OCR postprocessing system. *IBM J. Res. Dev. 19*, 4 (July 1975), 398–421.
6. Schmidt, J., and Shamir, E. An improved program for constructing open hash tables. In J.W. de Bakker and J. van Leeuwen (Eds.), *7th Colloquium on Automata, Languages and Programming*, July 14–18, 1980, Springer-Verlag, Berlin, pp. 569–581.
7. Shiloach, Y. Fast canonization of circular strings. *J. of Algorithms 2* (1981) 107–121.

Technical Note:
Human Aspects of Computing

Henry Ledgard
Editor

A Comment on English Neologisms and Programming Language Keywords

C. M. Eastman
Florida State University

The choice of keywords in the design of programming languages is compared to the formation of neologisms, or new words, in natural languages. Examination of keywords in high-level programming languages shows that they are formed using mechanisms analogous to those observed in English. The use of mirror words as closing keywords is a conspicuous exception.

CR Categories and Subject Descriptors: D.3.3. [**Programming Languages**]: Language Constructs—*control structures*
General Terms: Design, Human Factors, Languages
Additional Key Words and Phrases: keywords, natural language, neologisms

Introduction

High-level programming languages are not natural languages. They are artificial languages created in order to convey a sequence of instructions to a machine. They are written, not spoken. They have a well-defined and compact syntax. Their vocabularies are restricted.

Yet the designers and users of such programming languages learn and use natural languages all of their lives. It would be surprising if this experience with natural languages did not carry over into the use of programming languages in many ways [3]. For example, the choice of keywords by language designers is similar to neologism formation in English.

English Neologisms

A natural language such as English is not static. Changes occur in pronunciation, grammar, and vocab-

Author's Present Address: Caroline M. Eastman, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275.

ulary. New words are introduced, old words fall into disuse, the meanings of words change, pronunciation shifts, and grammatical changes occur. If there is a written version of the language, that also changes. The process of language change has been extensively studied by linguists. Overviews may be found in many linguistics books, including Samuels [5] and Sturtevant [6].

It seems reasonable that examining the mechanisms of change in natural languages can provide insights into the design and use of programming languages. Consider the example of English neologisms. A neologism is a new word; it may be either a newly created word or an existing word whose meaning has changed. The formation of English neologisms shows parallels to similar processes related to programming languages, the choice of keywords by the designer, and the choice of identifiers by the programmer.

Neologisms in English are rarely created out of thin air. They have roots in preexisting languages and are often very closely related to existing words. There are several common mechanisms by which new words or new versions of old words are formed. Such words are usually borrowed from other languages, formed in some way from preexisting words, or changed in form. On rare occasions a word may be made up with no obvious relation to preexisting words. The classification of neologism formation given here follows that of Samuels [5].

Words that are taken from other languages are *loan words*; they may be changed somewhat to fit the patterns of the new language. English has a large number of such loan words. The use of English words as programming language keywords may be viewed as parallel to such borrowings in natural languages.

Samuels classifies the main mechanisms by which words are formed from preexisting language elements. *Compound words* may be formed by concatenating two or more other words, as in "blackbird." *Phrases*, such as "give up," may acquire a meaning beyond that of the separate words. Words may be formed from *combinations of Greek and Latin roots*, e.g., "photograph." *Acronyms* such as "radar" may be formed from initial letters. *Blends* result from combining parts of other words, for example, "smog" from "smoke" and "fog." *Suffixes* and *prefixes* may be used. *Phonaesthemes*, phonemes that have a traditional association of meanings, may be used in creating new words. For example, "gr" can carry an unpleasant connotation, as in "grim," "greedy," "gruesome," "grumble," and "gross."

Some of these processes have parallels in programming languages. Compounds are frequently used for keywords and identifiers. Keywords made up of parts of existing words can be regarded as blends intermediate between compounds and acronyms. Suffixes, prefixes, and acronyms are occasionally used. Classical roots and phonaesthemes are rarely, if ever, used.

Another way in which a "new" word can arise is through a *change in meaning* of an old word. For example, "silly" at one time meant "blessed." English words which are used in programming languages are often changed somewhat in meaning. A word may also change form to such an extent that it may be regarded as a different word. Many such changes are purely *phonetic* and are not considered here. One of the more common changes reflected in a written language is that of *shortening or abbreviating* a long word so that it is easier to handle, for example, "phone" may be used instead of "telephone." Keyword abbreviation is quite common in programming language, for example, "var" is used in Pascal instead of "variable."

Closing Keywords

An important class of keywords that have been formed by a mechanism not normally used in English is the mirror words that are used as closing keywords in some languages. There have been regular, although far from universal, negative reactions to such words. Since these keywords do not relate to English in a conventional way, these negative reactions are understandable.

Three such mirror words, "fi," "od," and "esac," were introduced in Algol 68 [7]. They are used to terminate the scope of the associated keyword. For example, "fi" terminates the scope of an "if." The reasons for introducing such keywords are commendable. The use of these keywords avoids potential ambiguities by clearly indicating the end of a group of statements. The words used are shorter than the customarily suggested alternatives, such as "endcase." These keywords and others formed in a similar manner have also been used in the design of other programming languages.

The formation of new words by spelling old ones backwards does not correspond to common English practice. The use of such a mechanism depends upon a written language, and English is primarily spoken. Similar situations may be found in some word games such as palindromes and Pig Latin, and in cryptography. Examples that are in common use are rare, however. One such example is the name of a brand of laxative; "serutan" is "natures" spelled backwards.

Reaction to these mirror words has been mixed. Their use in Algol has been generally accepted. The continuing proposal of new mirror words can only be taken as an endorsement. On the other hand, disgruntled grumblings can occasionally be detected. Knuth [1] and Richard and Ledgard [4] both express dissatisfaction.

Kovats [2] argues that the problem with these keywords is that users regard them as nonsense. ". . . readers react rather negatively to the occurrence in programs of words which appear to be nonsense; empirically, of course, abbreviations (like **proc**) and concatenations (like **goto**) do not seem to be regarded as nonsense and are therefore not automatically prescribed." Kovats recommends that one criterion for selecting closing keywords is that they not "appear to be nonsensical"; this criterion

could, of course, be generalized to apply to the selection of any keyword. He remarks that this criterion is highly subjective.

As can be seen from the previous discussion, there is a good reason why mirror words are seen as nonsense, and abbreviations and concatenations are not. Native speakers of English make heavy use of abbreviations and concatenations, both in speaking and writing. They very rarely use mirror words. Keywords like "proc" and "goto" have been formed using standard English language mechanisms; keywords like "od" have not. So the recommendation that such keywords be avoided is not, in fact, as subjective as it seems.

Acknowledgment. I would like to thank Robin Carter and Henry Ledgard for assistance and encouragement.

Received 2/82; revised 4/82; accepted 5/82

References

1. Knuth, D.E. Structured programming with goto statements. *Computing Surveys, 6*, 4, (Dec. 1974) 261–301.
2. Kovats, T.A. Program readability, closing keywords and prefix-style intermediate keywords. *SIGPLAN Notices, 13*, 11, (Nov. 1978) 30–42.
3. Naur, Peter. Programming languages, natural languages, and mathematics. *Comm. ACM, 18*, 12, (Dec.1975), 676–682.
4. Richard, Frederic and Ledgard, Henry F. A reminder for language designers. *SIGPLAN Notices, 12*, 12 (Dec. 1977), 73–82.
5. Samuels, M.L. *Linguistic Evolution With Special Reference to English.* Cambridge University Press, London, England, 1972.
6. Sturtevant, E.H. *An Introduction to Linguistic Science.* Yale University Press, New Haven, Conn., 1947.
7. van Wijngaarden, A., Mailloux, B.J., Peck, J.E.L., Koster, C.H.A., Sintzoff, M., Lindsey, C.H., Meertens, L.G.L.T., and Fisker, R.G. Revised report on the algorithmic language ALGOL 68. *Acta Informatica, 5*, Nos. 1, 2, and 3, 1975, 1–236.

ABSTRACTS from other ACM Publications

In the ACM Transactions on Database Systems/December Issue

A Unifying Model of Physical Databases by D.S. Batory and C.C. Gotlieb

A unifying model for the study of database performance is proposed. Applications of the model are shown to relate and extend important work concerning batched searching, transposed files, index selection, dynamic hash, based files, generalized access path structures, differential files, network databases, and multifile query processing

Categories and Subject Descriptors: H.2.2 [**Database Management**]: Physical Design

General Terms: Design, Performance

Additional Key Words and Phrases: Unifying model, decomposition, simple files, linksets

For Correspondence: D.S. Batory, Computer and Information Sciences Dept., University of Florida, Gainesville, FL 32611.

A Practical Guide to the Design of Differential File for Recovery of On-line Databases by Houtan Aghili and Dennis G. Severance

The concept of a differential file has previously been proposed as an efficient means of collecting database updates for on-line systems. This paper studies the problem of datababse backup and recovery for such systems, and presents an analytic model of their operation. Five key design decisions are identified and an optimization procedure for each is developed. A design algorithm that quickly provides parameters for a near-optimal differential file architecture is provided.

Categories and Subject Descriptors: E.2 [**Data**]: Data Storage Representations; G.1 [**Mathematics of Computing**]: Numerical Analysis; H.2.2 [**Database Management**]: Physical Design; H.2.7 [**Database Management**]: Database Administration

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Backup and recovery, database maintenance, differential files, hashing functions, numerical methods, optimization, reorganization

For Correspondence: H. Aghili, IBM Research Laboratory, San Jose, CA 95193.

Performance Analysis of Linear Hashing with Partial Expansions by Per-Åke Larson

Linear hashing with partial expansions is a new file organization primarily intended for files which grow and shrink dynamically. This paper presents a mathematical analysis of the expected performance of the new scheme. The following performance measures are considered: length of successful and unsuccessful searches, accesses required to insert or delete a record, and the size of the overflow area. The performance is cyclical. For all performance measures, the necessary formulas are derived for computing the expected performance at any point of a cycle and the average over a cycle. Furthermore, the expected worst case in connection with searching is analyzed. The overall performance depends on several file parameters. The numerical results show that for many realistic parameter combinations the performance is expected to be extremely good. Even the longest search is expected to be of quite reasonable length.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sorting and searching*; H.2.2 [**Database Management**]: Physical Design—*Access methods*; H.3.2 [**Information Storage and Retrieval**]: Information Storage—*File organization*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Hashing, dynamic hashing schemes, linear hashing, extendible hashing

For Correspondence: P-Å. Larson, Dept. of Computer Science, University of Waterloo, Waterloo, Ont., Canada N2L 3G1.

Deadlock Freedom Using Edge Locks by Henry F. Korth

We define a series of locking protocols for database systems that all have three main features: freedom from deadlock, multiple granularity, and support for general collections of locking primitives. A rooted directed acyclic graph is used to represent multiple granularities, as in System R. Deadlock freedom is guaranteed by extending the System R protocol to require locks on edges of the graph in addition to the locks required on nodes.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Transaction processing*

General Terms: Algorithms; Theory

Additional Key Words and Phrases: Concurrency control, locking, serializability

For Correspondence: H.F. Korth, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.