Note on the addressing of lists by their source-language names

By E. W. Elcock*

This paper gives an informal description of a scheme for implementing the use of arithmetized source-language names for addressing list structures.

1. Introduction

Methods for direct addressing of data structures by simple arithmetization of their source-language names are of interest because of the much faster access to the data structure than, for example, table look-up procedures. List processing lends itself to a particularly simple scheme for implementing the use of arithmetized source-language names for addressing list structures. The scheme is described informally below.

2. Assumptions and conventions

The scheme assumes that each list source-language name can be arithmetized to generate an address in a given area of store. It is not assumed that the mapping from source-language names to addresses is one to one. In describing the scheme a machine-independent formalism will be used. The conventions of the formalized system are:

2.1. Word in the list area have the format:

[G][F][P1][P2]

P1 and P2 are obligatory address fields. P2 *must* contain either a pointer to a word in the list area or the terminator.

P1 *may* contain a pointer to a word in the list area.

G is an obligatory 2-bit field used for a code group in the arithmetized source-name addressing system (ASAS).

F is an implementation optional field. It is usually for a code group specifying the format of the word or data structure pointed to by P1.

2.2. The list area is a block of contiguous main storage which is initialized as the freespace stack (*freestack*) chained through P2 with G = 0 throughout.

2.3. Lists are formed from data or program by acquiring storage from the freestack in the usual way (see 2.5).

2.4. Each word in a user list-structure must have G = 1.

2.5. The ASAS automatically gives lists with a source language name a "head word" of the form

⟶ [1] [ ] [NAMEP] [LISTP]

where: NAMEP is a pointer to the internal representation of the source–language name (a user data structure: e.g. a simple chained list of packed characters);

LISTP is a pointer to the named list.

2.6. In standard processing of a *located* list, only those listwords for which the code group G = 1 (0) are to be regarded as part of the located list (freestack). On encountering a listword with G = 2 or 3 the processor action should be to "skip" to the tail (located by the pointer in the P2 field of the listword).

3. Filing of named lists

It is assumed that the list name and list have been formed and that HDWORD is the pointer to the head-word of the list (see 2.3 to 2.5 above). It is also assumed that a user (or separate ASAS routine) has arithmetized the source–language name and that AA is the resulting address in the list area. The following possibilities arise:

3.1. C(AA) = [0] [ ] [ ] [X]

The AA has not been generated previously in the ASAS, and AA points to an unused word on the free stack.

The code group is changed and HDWORD inserted in the P1 field so that now

C(AA) = [2] [ ] [HDWORD] [X]

Note that the word is left on the freestack. By 2.6 this will be skipped when acquisition of storage from the freestack reaches this point.

3.2. C(AA) = [1] [ ] [X] [Y]

The AA has not been generated previously but the list-word to which it points has already been used in a previously stored list, L say. C(AA) is restored in the first available word on the freestack. Let the pointer to this word be P.

C(AA) is now reset to

C(AA) = [2] [ ] [HDWORD] [P]

By 2.6, the presence of this word on L will be ignored in any standard processing of L.

3.3. C(AA) = [2] [ ] [X] [Y]

* *Department of Natural Philosophy, University of Aberdeen, Aberdeen.*

The AA has been generated previously. The substructure

[1] [ ] [HDWORD] [·—]—▶ [1] [ ] [X] [0]

is stored. Let the pointer to this substructure be P. C(AA) is now reset to

C(AA) = [3] [ ] [P] [Y]

3.4. C(AA) = [3] [ ] [X] [Y]

The AA has been generated at least twice already. The substructure

[1] [ ] [HDWORD] [X]

is stored. Let the pointer to this substructure be P. C(AA) is now reset to

C(AA) = [3] [ ] [P] [Y]

4. Retrieval of named lists

The source–language name is stored and arithmetized. The following possibilities arise:

4.1. C(AA) = [2] [ ] [X] [Y]

'X' is the required pointer to the headword of the list. The headword of the list and the pointer to the list itself can be obtained by standard processing (remember 2.6).

4.2. C(AA) = [3] [ ] [X] [Y]

The AA is not unique. The table look-up list pointed to by X, of the intrinsic (2.6) form

X——▶ [1] [ ] [HDWORD]
[·—]—▶ [1] [ ] [HDWORD] [·—]—▶

is now processed (2.6) to find the headword whose name matches the given source language name, etc., etc.

5. Final comment

The system is effective provided only that the density of headwords in the list area is low enough that lack of uniqueness of AA's is the exception rather than the rule. This is so for most applications of list processing.

The system can be incorporated readily into a general list-processing system by a trivial modification of basic list-processing routines to take into account the requirements 2.4 to 2.6. The system need not interfere in any way with the characteristic dynamic nature of list processing.

The ASAS system has been implemented for a general list processor on the Elliott 803 in the course of a project supported in part by the S.R.C.

THE COMPUTER JOURNAL

Published Quarterly by

The British Computer Society, Finsbury Court, Finsbury Pavement, LONDON, E.C.2, England.

Subscription price per volume £3 10s. (U.S. $10). Single Copies £1.

All inquiries should be sent to the Assistant Secretary at the above address.