1970

FORTH - A Language for Interactive Computing

Charles H.Moore

Geoffrey C. Leach

Mohasco Industries, Inc.
57 Lyon Street
Amsterdam, New York 12010

Abstract

FORTH is a program that interfaces keyboards with computer. It provides all the software necessary to time-share users and manage core and disk memory. Its key is a dictionary that divides memory into entries that identify character strings, code and data. The resulting language is sufficiently powerful to describe FORTH itself, and sufficiently flexible to make inquiries with. It may be readily extended to handle as many, and as complex, applications as hardware permits. On the B-5500 FORTH uses 2K of core and can express a complex application in each of the 30 1K regions of core that remain.

What is FORTH?

FORTH is a computer program. It provides a software interface between terminal and computer. It provides a complete Software interface with a large computer. Including a language in which the user can describe his application. And in which FORTH is written.

The software provided with large conmputers supplies a heirarchy of languages; The assembler defines the language for describing the compiler and supervisor; the supervisor the language for job control; the compiler the language for application programs; the application program the language for its input. The user may not know, or know of, all these languages; but they are there. They stand between him and the computer, imposing their restrictions on what he can do and what it will cost.

And cost it does, for this vast heirarchy of languages reguires a huge investment of man and machine time to produce, and an equally large effort to maintain. The cost of documenting these programs and of reading the documentation is enormous. And after all this effort the programs are still full of bugs, awkward to use and satisfying to no one.

We maintain that this multi-level nightmare is precisely that. We place the blame upon the lack of a suitable language. FORTH provides a suitable language, and by using it an order of magnitude improvement in the cost, effort and efficiency of providing a terminal interface.

We introduce a language with which a man at a keyboard can talk to a computer - man-machine communication. The reverse problem does not arise. The computer handles machine-man communication in rote fashion-typing specified replies.

We insert FORTH between man and machine and define 2 dictionaraies: man-FORTH and FORTH-machine. The man-FORTH dictionary is a collection of documentation - of which this is a part - that teaches the man how to express his thoughts in FORTH.

The FORTH-machine dictionary is the subject of this paper, for it is the key to the dramatic sucess of FORTH.

The Dictionary

A dictionary is an association of words with meanings. FORTH's dictionary is a list of words together with their definitions (Fig. 1). Almost the entire program is contained in the dictionary, excepting only certain control information.

FORTH is an interpretative program, as are all language processors. It uses a scanner to read character string, identify individual words and find them in the dictionary. The FORTH dictionary differs from a conventional symbol table. Each entry specifies code that is to be executed when that word is encountered. Parameters used by this code are also part of the entry.

Certain words construct dictionary entries for the words they preceed - they declare their successors. Each such word indentifies the code to be executed when its entry is encountered. For example, a : declares the next word to be a definition.

It is helpful to distinguish 3 classes of words: nouns, verbs and definitions. Illustrating each of these may clarify the structure of the dictionary (Fig. 2).

A noun is a word that names storage. The code it specifies causes either an address or a value to be placed on the stack. (This stack is a last-in/first-out store used to hold arguments - keep them in mind.)

```
3 VECTOR X
```

Will reserve 3 consecutive cells for X. When X is seen by the scan-ner the address of the first cell is placed on the stack.

The phrase

```
7 CONSTANT N
```

will assign 1 cell to hold the constant 7. When N is seen 7 will be placed on the stack.

A . declares a verb and points to the parameter part as the code to be executed. Subsequent words will generate instructions and place them in this region.

A : declares a definition and stores the following character string (until the first ;) in the parameter part. The code specified directs the scanner to interpret this string. The words it finds there may in turn be nouns, verbs, or definitions. Definitions may be referenced within a definition (nested) but may not be declared there.

The process of making a dictionary entry and filling the parameter part is core allocation. A noun fills core with data, a verb with code and a definition with a character string. Clearly different entries will require different amounts of core, which complicates the dictinoary search. However the variable length of entries poses no real problem.

Just as making dictionary entries uses core, so deleting them releases core for re-use. To avoid awkward gaps in the dictionary - and because it seems the natural thing to do - only the latest entries my be deleted. The dictionary thus provides the user with direct and effective control of the core assigned him.

The Program

The dictionary organizes core and thus forms a major component of FORTH. However most words do not modify the dictionary - they merely reside there. Fig. 3 is a functional diagram of FORTH. The major blocks besides the dictionary are the scanner and the queue. Notice that although FORTH implements a supervisor, compiler and loader these aspects cannot be localized.

The Scanner:

The heart of FORTH is the scanner. A simple loop that reads the next word, finds it in the dictionary and executes its associated code.

What does the scanner recognize as a word? A word starts with any charac-ter and ends at the first special character. Thus

```
X@J+
```

would be broken into

```
X @J +
```

In practice, most words are separated by spaces - but this is not always convenient. Words have arbitrary length. The leading characters are stored in the dictionary and the rest discarded.

The convention the words may start with special characters increases the set of words with mnemonic value. For example

```
+ +1 +FIX +VECTOR
```

may represent different addition operators. And

```
IF .IF #IF
```

different forms of a conditional operator

In the first case the primary mnemonic value is carried by the special character; in the second by the alphabetic word.

Period (.) is considered alphabetic so that it may serve as a break character and decimal point. Thus

```
3.14 COSMIC.RAY
```

are both words.

The scanner depends upon a character pointer to tell it where to read next. A word may change this pointer and thus alter the sequence of input words. This is the way definitions and backward branches are implemented. The user is free to establish other conventions if he wishes. Manipulated the character pointer permits a remarkably simple and independent scanner.

For example, imput is initially expected from the keyboard. The first word would probably cause input to be read from disk - expanding the dictinary. Other words might come from definitions stored in core. The scanner itself is independent of the source of its input, and freely intermingles all 3 sources (Fig. 4).

Dictionary Search:

Finding a word in the dictionary involves more than the structure of the dictionary. The search proceeds from later to earlier entries. This is a convenient way to search since a word may be redefined and the latest definition found.

However the search does not always begin with the latest entry. To some extent this is under the user's control, but the main variation involves definitions. A definition is an entry that contains other words that must be interpreted. We specify that the search for these words begin at the defintion (Fig 5). This has 2 desirable effets: Words must be defined before they are used - a valuable diagnostic aid. And when a word is redefined, though its old meaning is not directly accessible, old definitions can still reference the old meaning. The only restriction on redefinition is exactly that - that the user need no longer directly access the old meaning.

The net effect of this convention is that words may be defined, redefined and referenced in a completely natural fashion. The user rarely need trouble about the search mechanism; it follows the context so closely as to seem to be aware of his intention.

Of course the redefinition ability means that there are no reserved words. In fact there are no reserved concepts. The user is completely free to redefine any aspect of FORTH he wishes. Our system definitions are available to save him trouble, and have been selected for maximum utility. But we do not force our choices upon him.

In addition to word, address and parameter parts, each entry has a link and the dictionary becomes a linked list. Since the dictionary is large, several hundred entries, search time is important. So we scramble each word to determine which of 32 chains it will be found in and then follow the links of this chain through memory (Fig. 6). Thus with 300 entries, only 5 comparisons should produce a match. This complicates the problems of deleting entries and starting the search at definitions, but the advantage is overwhelming.

There is a large class of words that are defined but not in the dictionary - numbers. Numbers may be placed in the dictionary, and often are. But if a number is not found, it is assumed to define itself and its value is placed on the stack.

If a word cannot be found in the dictionary and is not a number, it is undefined and in error. The scanner will type the word and return to the keyboard for input - the current input string being abandoned. The user must correct the error, restore the dictionary and restart his program.

The Queue:

The purpose of the queue is to share the computer among users. When a user enters the system he is allotted a region in memory for his dictionary and for the buffers and indicators required by his presence. This includes notifying the queue of his arrival and of his priority and linking him into the existing dictionary at the appropriate place, which may be the basis system definitions or some resident application (Fig. 7).

When a user requests a system resource (including I/O) he is enqueued until it becomes available. Thus, the queue is used to resolve conflicts among users. Timesharing the computer is implicit in this treatment of the user. Once a user is established - and this requires only that core be available - he becomes a natural part of the system, indistinguishable from preexisting parts. User swapping may take place every time there is an interrupt or the current user enqueues and requires only that a very small amount of system information be changed.

No attempt is made to overlap or otherwise optimize I/O. A user's requests are satisfied as expedient with due regard for priority. Although this may be inefficient from the standpoint of his elapsed time, there are other uses and to expedite one is to penalize others. Every effort is made to minimize the impact of any user upon the system. For example, sorting algorithim are usually designed to minimize elapsed time. FORTH's sorting algorithim mini-mizes core and disk, and is free to run as long as necessary. The problem of cpu-bound verbs is solved by handling the timer interrupt in the same way as an I/O finish.

Sheets:

The definitions we have discussed to far have referenced character strings in core. Another kind of definition is stored on disk. The phrase

```
1017 SHEET FIT
```

declares FIT as a sheet with disk address 1017. When FIT is encountered, the scanner is directed to disk for further input.

FIT identifies a region of disk that contains source language. We call it a sheet because typically such definitions consist of about 50 lines of 40 characters, which fit nicely on a sheet of paper.

An application typically uses several sheets to describe the dictionary entries it requires. In effect, referencing a sheet moves its entries from disk to core, compiling them on the way.

A sheet may reference other sheets and thus direct the retrieval of information. For example, to generate FORTH we access ROOT, a sheet that acts like an index: it declares a number of other sheets. One of these sheets is SYSGEN, which references other sheets in a particular order and thereby assembles the system dictionary.

Sheets may be edited and definitions added or changed using definitions on the seet EDIT. Each user is assigned disk space and provided tools to maintain it.

The Language

We have been speaking of FORTH as a program that implements a language and we have described the dictionary that explains the language to the computer. But we have not described the language beyond mentioning the basic kinds of words: nouns, verbs, and definitions.

There are two reasons for this. FORTH, to an honesty significant extent, only supplies the tools whereby a user can construct a language that looks as he wants it to. Second, there are 200 words in the dictionary, all of which are intended to be used - and therefore must be explained.

Moreover, although we claim significant compactness of FORTH programs, such compactness is not apparent in small examples. It arises through the economies of tailoring definitions to a specific application. It is the case that our small programs are smaller than your small program, but our large programs are much smaller than your large programs.

Nevertheless, we can try to give you an impression of FORTH's language by some brief discussion and a simple example.

Verbs vs. Definitions:

It is useful to distinguish 2 classes of words: those that cause instructions to be generated and those that do not. The former are used only to describe the code for verbs, the latter are used everywhere.

An instruction generating word is *DUP. It says: Generate the instructions that will duplicate the top of the stack when executed. It is used by verbs that wish to manipulate the arguments they find on the stack.

The word DUP is interpretive. It says: Duplicate the top of the stack.

It is used when the stack is to be changed at once, it contrast to *DUP.

There is clearly a similarity between the two, in fact DUP is defined in terms of *DUP:

```
. DUP *DUP RETURN
```

is a FORTH statement that effects a dictionary entry. The . declares the next word (DUP) to be a verb. The words following it generate the code for the verb. *DUP generates its instructions, RETURN generates a branch to the scanner.

In general, instruction generating words are prefixed by one of several special characters, for ready identification. For example, *IF.....*THEN represent a pair of related words that generate a condition forward branch. *IF generates instructions to test the top of the stack for true (1) or false (0) and branch on the false condition. *THEN fills in the branch address. Thus,

```
*IF  *DUP  *THEN
```

is a FORTH statement that will generate code to test the top of the stack for true or false and duplicate on true.

Analogous to the above is the statement:

```
IF  DUP  THEN
```

which will test and duplicate the top of the stack on true. This statement actually tests the stack when encountered, in contrast to the previous example, which generates instruction to do the same thing.

Commonly used words fall into 3 categories:

Operators (arguments on the stack)

+ - x / MOD OR GREATER

Conditional statements:

IF ELSE THEN

Iterative statements:

DO CONTINUE

These words are all verbs. In fact, most system words are verbs.

There is a certain equivalence between defintions and verbs. In fact, most operations could be written as either. Definitions are easier to code, but their execution, being interpretive, is slow. Verbs are harder to code, bu are most versatile as they may use the full hardware instruction set and are 100 times faster in execution. It seems that a good balance between ease of coding and execution time is achieved by coding the inner loop as a verb and the remainder as a definition.

An Example

We illustrate the language of FORTH by summing the squares of integers between 2 limits. The user types

```
17 63 SUMSQ
```

To request $\sum_{I=17}^{62} I^2$

Fig. 8 shows the source language required. We will attempt a running commentary:

Since the numbers 17 and 63 are not in the dictionary, the scanner places them on the stack. It then finds SUMSQ in the dictionary and executes the associated code. The colon preceeding SUMSQ indicates that it is a definition, so the scanner is directed to the character string following it. The next word read is thus SHIFT.

SHIFT mist also be interpreted. At this point the stack contains 17 63. It will place a zero before the numbers on the stack (interchanging them in the process). -1 @T are 2 words that form a phrase. -1 is a number that is placed on the stack. @T uses it as an address relative to the top of the stack and fetches the first number - 17. 0 is a number and is placed on the stack. -3 =T is another phrase that will store this 0 3 places deep in the stack - where the 17 was. At this point the stack contains 0 63 17.

The ; marking the end of SHIFT directs the scanner back to SUMSQ where the next word is DO. DO marks the beginning of a loop.

Status

FORTH was developed in the fall of 1968 on an IBM 1130. We used it primarily to generate pictures on the 2250 display scope, a task it handles with ease. We also developed a report generator that would select, sort and print records from sequential files with an ease and versitility beyond the range of a conventional approach.

In February 1969, we generated FORTH fo the Burroughs B-5500. At the time of writing, we are developing B-5500 applications, one of which will be the 1130 report generator.

FORTH encourages compact programs. Definitions make the programmers job so much easier that he is strongly motivated to isolate the significant features of his problem and to factor out those constructs that occur repeatedly. Proof of the effectiveness of this technique is the compactness of FORTH itself.

FORTH is a small program, composed mostly of dictionary. FORTH is written in FORTH and is its most complex application. FORTH requires 16 sheets to describe FORTH. FORTH requires 20 seconds to generate FORTH. FORTH uses 4K 16 bit words on the IBM 1130 and 2K 48 bit words on the B-5500. We estimate 8K bytes on any of the System 360 models. These figures are for a complete software package capable of supporting an independent application for every 4-8K bytes (depending on word size and hardware) of core available.

Both core and source program size are orders of magnitude less than the conventional systems they replace. This is a dramatic refutation of the conventional approach to software, and we presume to offer FORTH as a standard against which to measure system software.

Any application within the capacity of the hardware can be described in FORTH: easily, compactly and in a manner as self documenting as the programmer wishes. There are certain restrictions on how FORTH may be used: It requires keyboard communication. Jobs may not be batched in background, however they require only as much attention after starting as the programmer has designed into the application. FORTH has no automatic error-handling facilities. These may be provided the the user if he wishes, but otherwise FORTH merely reports an error to the user and awaits instructions. The user must assume responsibility for his source language and the integrity of his data. He must maintain his dictionary in the appropriate fashion. In brief, FORTH words with the user to solve his problems: it makes no attempt to solve them for him.

For example, FORTH can process a payroll. It requires a responsible person from the payroll department to edit the personnel file, see that the checks are in his remote printer and respond to errors as they are detected by FORTH. If you prefer to batch your change cards, merge them into a master tape and print the checks off line, stick with COBOL.

FORTH is a complete software package and a closed software package. FORTH programs must be written in FORTH. There are no provisions for compiling or executing programs written in other languages. To provide such facilities would be to force upon FORTH the very problems it is designed to avoid.

In Summary

FORTH is a program that interfaces keyboards with computer. It is a small program, though not intended for small computers. FORTH requires about 4K of core and each application (keyboard) requires an additional 1K. It also requires perhaps 100K characters of disk storage for source language. Modest enough requirements, but beyond the scope of desk-top computers.

FORTH implements a language that gives the user the complete access to hardware capabilities, and complete freedom to design a compact POL. Interestingly enough, FORTH applications do not use subscripts, counters and such house-keeping variables. They require very little data storage - usually in the form of arrays. They are composed mostly of definitions - seemingly endless nests of definitions - which are simple to write and simple to use. FORTH applications make efficient use of computer resources, and share these readily with other applications. In a word, FORTH is THE computer language.

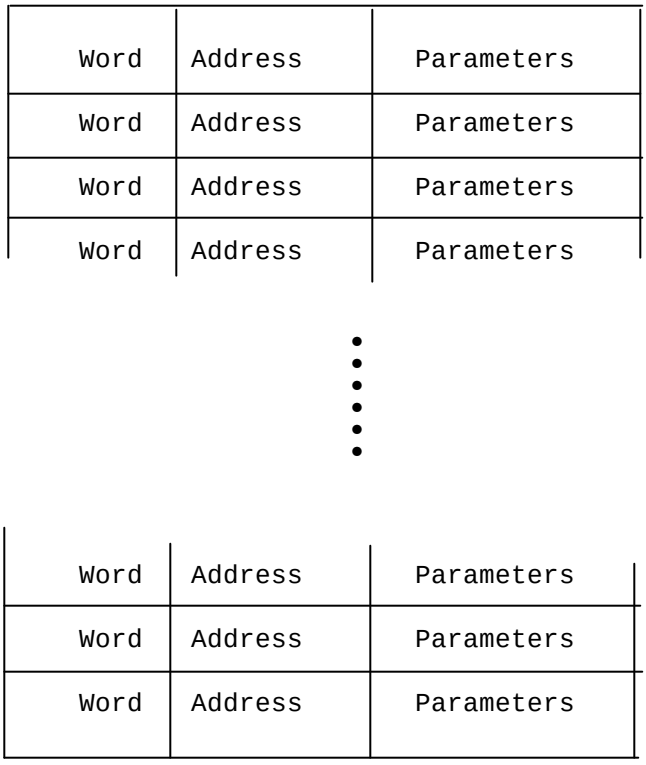

Fig. 1 - The Dictionary:

A word may be 4 to 8 characters long, depending on implementation;

Its address identifies code to be executed - distinguishes kinds of words;

Its parameters may be absent, numbers, code or characters - they distinguish words of the same kind.

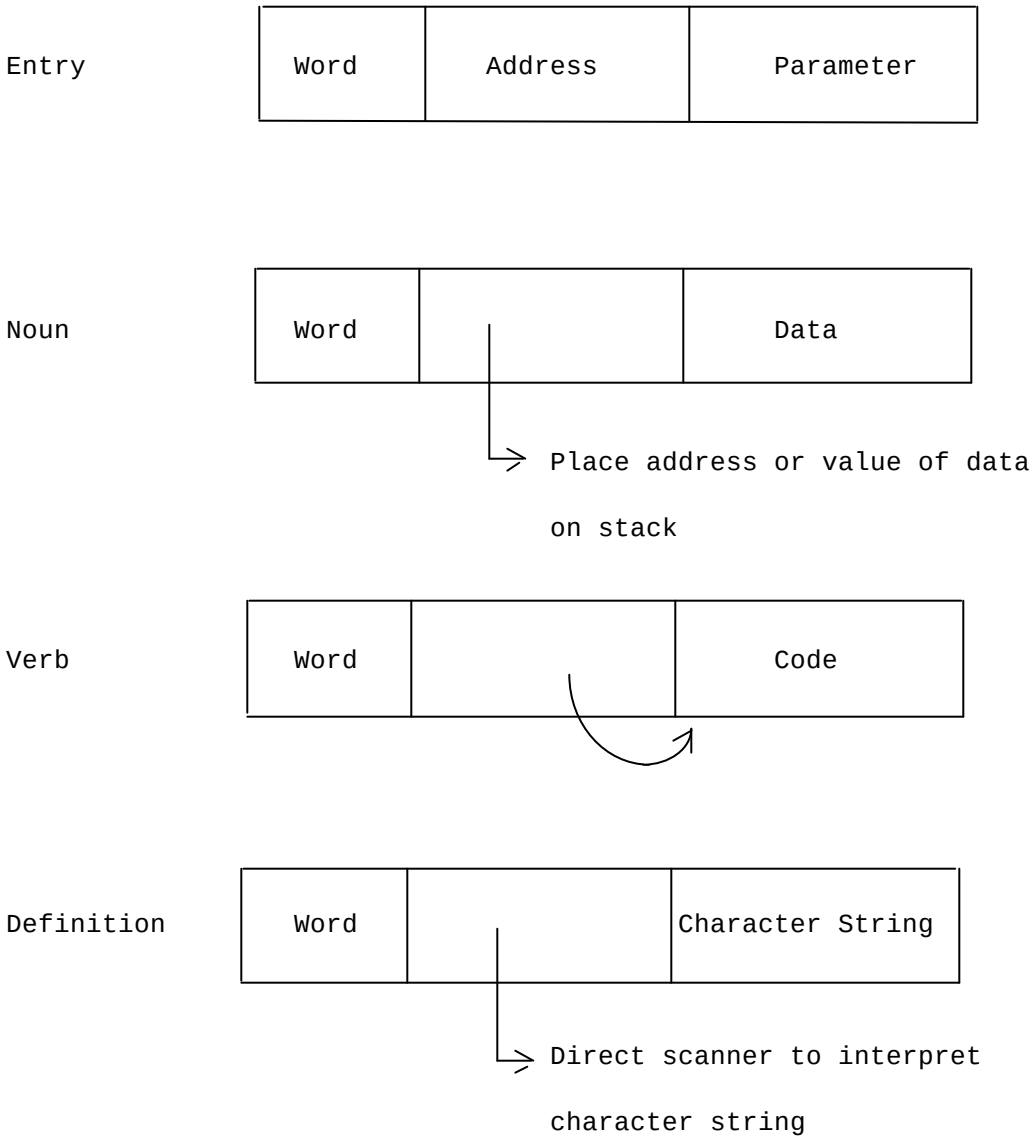

Figure 2

Dictionary entries are distinguished by the code executed when they are seen and the parameters used by that code.

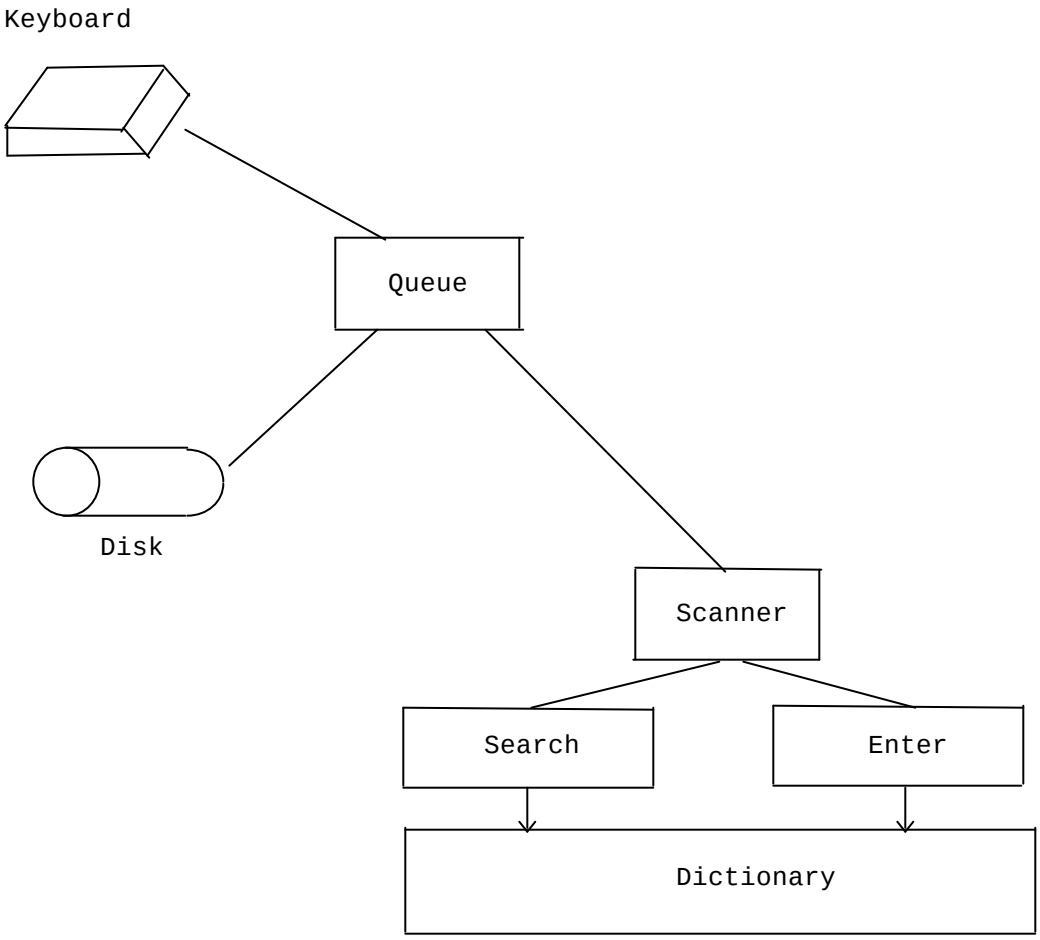

Figure 3.

Components of FORTH.

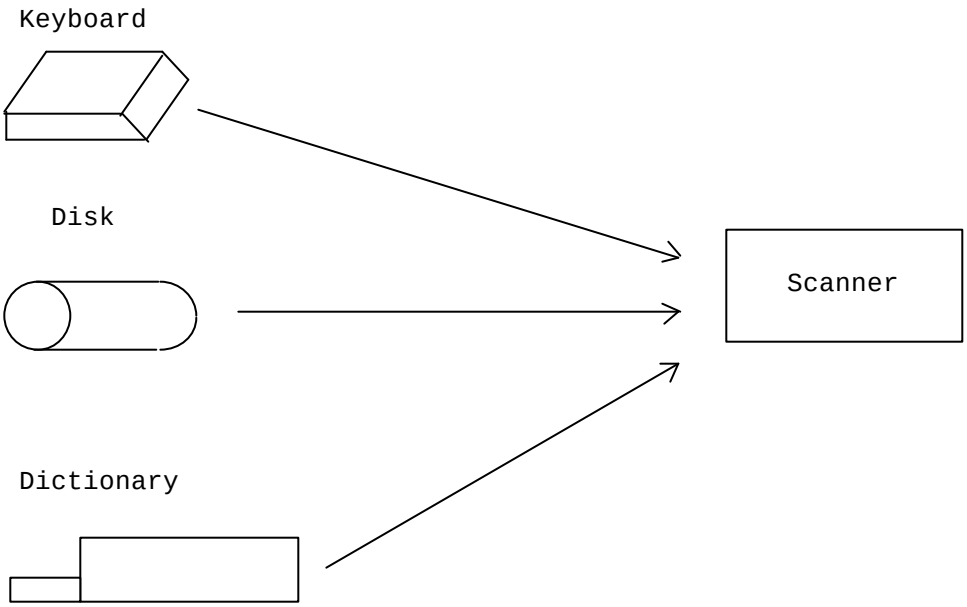

Figure 4

The scanner accepts input indiscriminantly from keyboard, disk and dictionary.

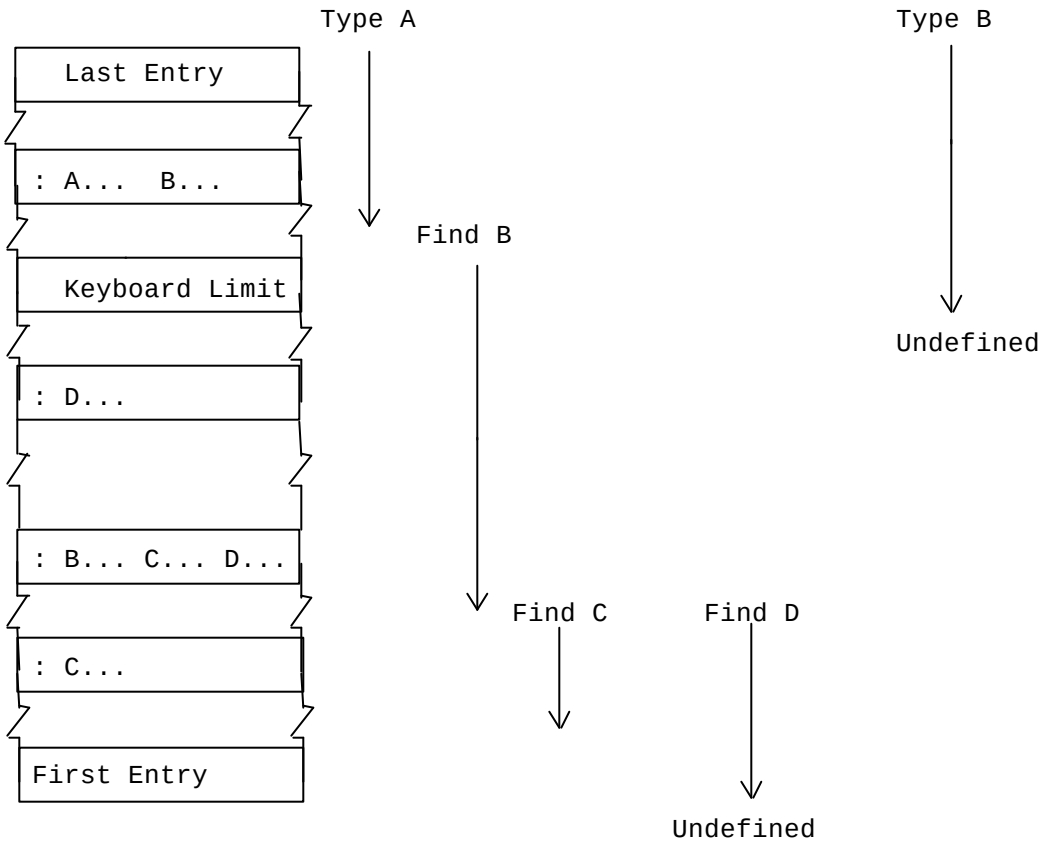

Fig. 5 Dictionary search depends upon the order in which words are declared, and the context in which they are sought:

Words read from the keyboard are sought from the last entry - subject to a lower limit.

Words read from definition are sought before the definition.

The arrows show the portion of the dictionary searched for the definition indicated in the dictionary.

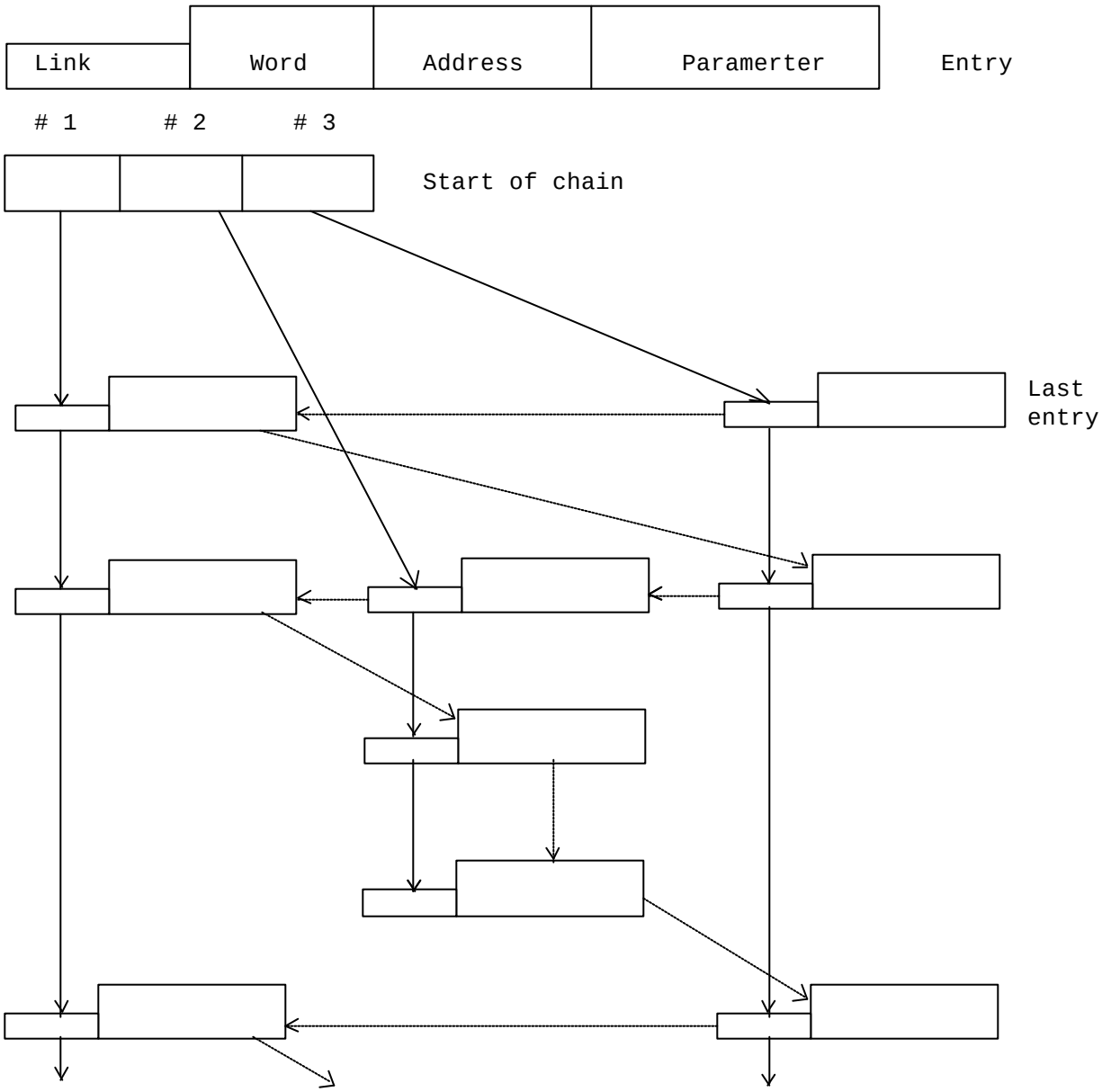

Fig 6 Dictionary showing 3 search chains

Dashes show physical order; lines show search order.

Search Proceeds:

1- Scramble word to select chain #1, #2, or #3.

2- Find last entry in chain from array of chain heads.

3- Follow links comparing word with entry.

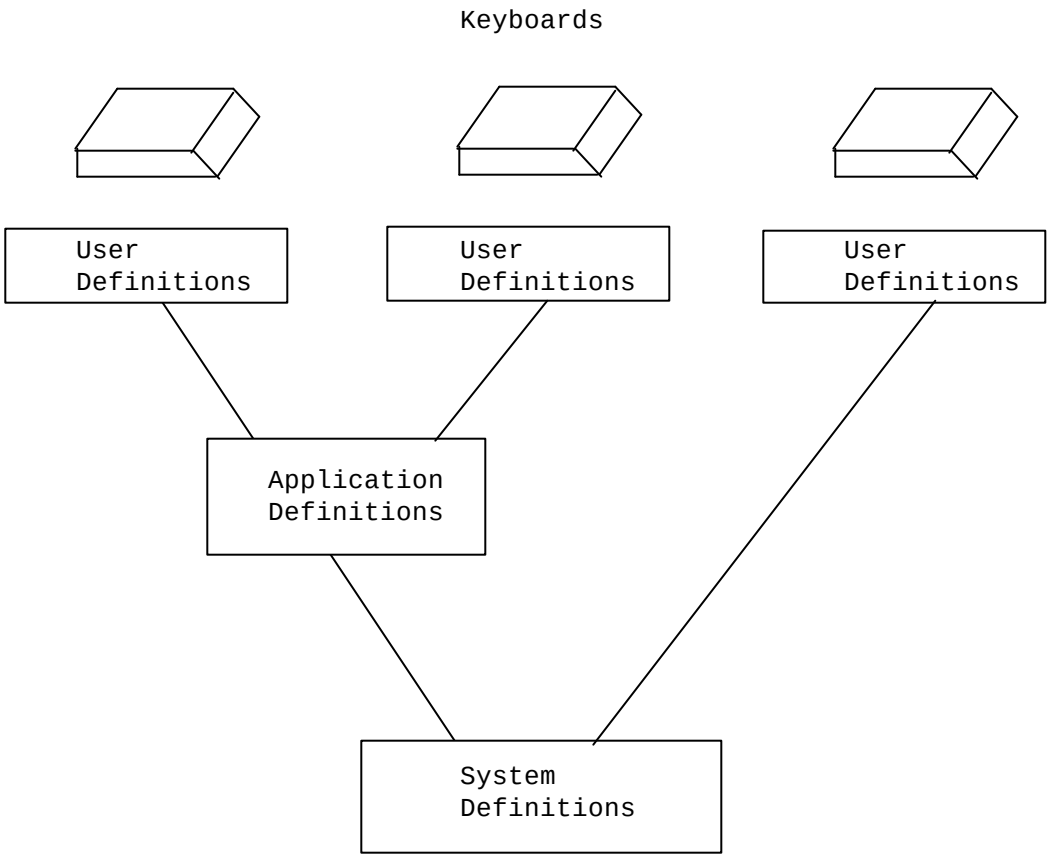

Figure 7

User dictionaries may be linked together and with the system dictionary.

```
. F   *DUP   *DUP   x1   RETURN

: SHIFT     -1  @T   0   -3  =T  ;

: SUMSQ     SHIFT   DO
      F   -3  @T   +   -3   =T
      1+   CONTINUE   BD;
```

Fig. 8 FORTH source language to evaluate and type a sum of squares with limits on the stack.