HFORTH: A High Level Business Language in FORTH

Pierre Moreton
99Ø Arnold Way
San Jose, CA 95128

ABSTRACT

A presentation of some features of HFORTH, a high level business language written in FORTH, some recent applications, and the extension to THE ROBOT, a menu driven program generator.

INTRODUCTION

FORTH has demonstrated the interest of its concepts essentially in the scientific domain, in process control and in graphics. Its main arguments are speed, compactness, and interactivity. However, I have spent over seven years using FORTH exclusively for writing business applications, a domain for which FORTH was not originally designed, and where none of these characteristics are critical.

I selected the language for other original features it has:

1) the capability to easily define a very high level, English-like language on top of it, to which anyone can be trained in a couple of hours.

2) the ease of building a library of standard business routines, so that a new application can be written from existing modules.

3) the impressive **flexibility** to change any feature of an application.

4) the portability of the resulting packages.

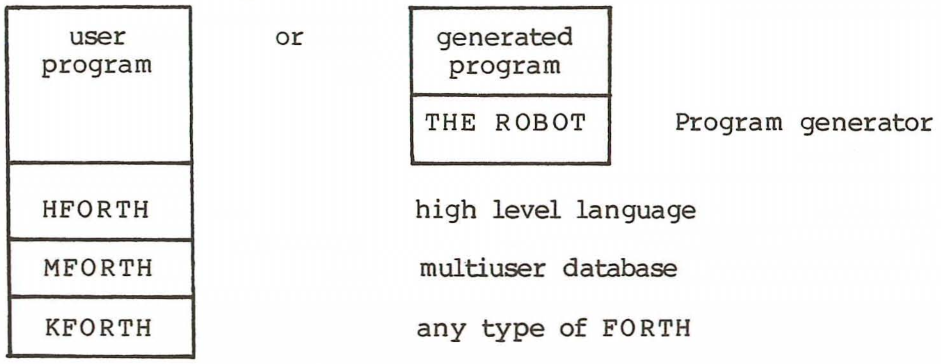
The main characteristic of the business software, in vertical markets like banking, insurance, real estate, medical analysis, law offices, and food industries, in which I have been involved, is that the specifications are continually changing. This is because the user does not really know what he wants until he actually uses the product, because each user has his own way of managing his business, and because his needs are in constant evolution. Looking back to all these applications, for which the size of the compiled code varies from 3ØØk to 2ØØØk, it seems to me that the major desirable points of a business language are:

1) speed of development, provided by a very high level set of business oriented instructions.

2) low cost for entirely modifying an application, even in a final stage, provided by a straightforward readable syntax.

This paper is about the steps accomplished in this direction.

The structure of the system is the following:



HFORTH

This language is intended to be used by programmers, totally ignoring what FORTH is about. It is made of about 12Ø basic words with which any business application (so far) can be written. For very specific functions, or in the very rare cases where speed may become critical, the underlying FORTH can be used and mixed in the HFORTH code. I do not intend to give here a tutorial on HFORTH, but examples in two areas, data declarations and files, show some significant caracteristics and explain why it is so cost effective to program in HFORTH. In what follows the HFORTH words are underlined.

1. Data Declarations

The basic syntax is

```
    V CLIENT#          : N :  5 Ø ..
```

where V is a defining word for an HFORTH variable.
N means that the preceding variable is numerical, in this case with 5 integers and Ø decimals. This will be used to control the entry format and to display accordingly, although all the numbers are internally handled in double precision.
.. marks the end of the definition.

There are four basic types of HFORTH variables:

```
Numerical, as above
Strings (text)
   V NAME            : T : 21  ..
Dates
   V DATE.OF.BIRTH  : D :  6 Ø : TEST-DATE : DIS-DATE ..
Boolean (true or false) with another defining word
   LOGIC MARRIED
```

Automatic Controls
To the variable can be attached two optional procedures, as above in the definition DATE.OF.BIRTH. The first one will be executed at entry time, in addition to the simple format checking, to control that the entered value conforms to some special characteristic. TEST-DATE is an HFORTH

word which controls that the entered number is consistent with a date and stores it as an absolute day on which calculations can be performed. The second one redisplays the entered value as desired. DIS-DATE is an HFORTH word which redisplays the date with slashes and the name of the day of the week.

The control procedures can be either existing HFORTH words or further defined by the programmer:

```
   V FAMILY.SITUATION : T :  2 : CTL.SIT   : DIS.SIT  ..
with
   :CONTROL CTL.SIT
                  IF( ENT-VALUE =/= " MA " AND
                       ENT-VALUE =/= " SI " )TRUE
                       " Enter MA or SI please " -> TEXT-ERROR
                  THEN
;
```

An entry not equal to MA or SI will issue the above message and will prompt for another entry.

```
   :DISPLAY DIS.SIT
                  ACCORDING-TO DIS-VALUE
                     VALUE " MA " ==> PRINT " Married "
                     VALUE " SI " ==> PRINT " Single "
                  END-VALUES
;
```

In this example the content of the variable never appears to the user because it is displayed differently.

These HFORTH procedures have a strong flavor of FORTH, because the : and ; are used the same way, but they show the use of some HFORTH words, which have the "natural" notation instead of RPN. ENT-VALUE and DIS-VALUE are general names that replace any HFORTH variable and take all its characteristics (dynamic vectoring). It allows one to build libraries of procedures of general use.

In the above example, if we wanted the program to accept more than these two programmed values, but control that the entered value matches with a series of values stored in a file filled in by the user, this can be executed automatically if the definition mentions it:

```
   V PROFESSION.CODE : T : 3 ..  TABLE PROFESSIONS
```

In this case the system will forbid any entry of a value which does not exist in the named file (PROFESSIONS) and if this value exists, the system will display the field of its related file which is associated with THE-KEY, called its REFERENCE (PROFESSION.NAME, for instance).

At entry time some essential commands are built in:

* If the user wants to enter a value which has not yet been entered in the related file, typing a * will open a window to add a new entry in this file, and the entry of this value will then be permitted without

interrupting the normal transaction. The word NO* can disable this remote creation (in file systems for instance).

? The entry ? will open a window to display all the keys of the related file and the entry of ?XXX will display only the keys of this file starting with XXX.

A last optional feature can be added to the data definition, which is the attribute HM meaning Help Message. This will display the data for which a help message, of variable length, can be entered by the word HMESSAGES and stored in a file independently from the code. The appropriate message will be displayed when typing ? instead of an entry for this data. It supercedes the listing of a related file in case both attributes are given to a datum.

All these attributes that can be given to a variable at definition time are stored in the parameter field, of variable length, and executed automatically, without any additional code.

2. Files

All the business applications use a great number of data files, and the need for high level words to manipulate them is obvious.

2.1 Declarations

The typical syntax is

```
   1Ø1 STR CLIENTS
       ORD CLIENT#
       RFR NAME
           DATE.OF.BIRTH
           FAMILY.SITUATION
           PROFESSION
       END-STR
```

This descriptor is a logical file called structure. By its label (here 1Ø1) it is assigned, through a table (unknown by the programmer) of assignments, to a physical file on disk. The assignment of the physical file is done by

```
   1Ø1 LABELFILE C:PCLIENTS
```

assuming that the file PCLIENTS has been created on disk. This assignment can be changed dynamically by a command like 1Ø2 ASSIGN CLIENTS.

STR is a defining word for the name of the structure (CLIENTS).
ORD means that we want a file ordered (balanced AVL tree) by the key whose name follows (CLIENT#).
RFR means that we want to associate the field whose name follows to the key; that is to say, each time the program will mention PRINT CLIENT# it will display not only the content of CLIENT# but the NAME too. I call this feature automatic decodification.
END-STR ends the definition and stores in the parameter field of

CLIENTS, in addition to the PFAs of all the fields, the type of file and the record length that it computes from each field.

Each field can be given an attribute like

```
  AH always hidden
  AD always displayed
  OH optionally hidden
  OD optionally displayed
```

This feature is of considerable interest in customizing a running application for a specific customer without either changing the code or recompiling, but as an installation procedure.

2.2 File Manipulation

A set of very general words allow us to enlarge our library of modules:

```
   THE-KEY REFERENCE

   ENTER-RECORD
```

prompts for an entry for each field of the record, and eventual modifications of the entries. Here are some examples of these words.

```
  : ADD    ENTER-NEW-KEY  \ prompts for the key field
                            \ and controls that it does not exist
           JUMP            \ = CR but scroll inside a window
           ENTER-NKEY     \ prompts for all other fields
           <SECUR SAVE SECUR> \ stores the record under
                                \ security protection
  ;
  : DISP IN CLIENTS                \ makes this file current
           BEGIN ENTER-KEY  \ accepts only an existing key
           WHILE( THE-KEY =/= BLANK )TRUE
                    ACCESS  \ reads disk and copy to files buffer
                    DISPLAY-RECORD
           REPEAT
        OUT                   \ closes the file
  ;
```

As these routines can be totally written in general terms, a super high level word MANAGE does all the usual actions on a file: ADD, UPDATE, DELETE, DISPLAY, for a given record, and LIST a series of records with the option to print or not.

Example:

```
  V FILE.TO.MANAGE  : T : 20 ..

  : RUN
     ENTER FILE.TO.MANAGE       \ prompts for a file name
     IN [ FILE.TO.MANAGE ]     \ makes this file current
          \ by executing whatever is in the string between brackets
             MANAGE    \ prompts for the action and executes it
     OUT
  ;
```

2.3 Relations Between Files

They are automatic when the same field name belongs to two different files and is the key of one of them. They can be forced between fields with different names by the TABLE function discussed previously.

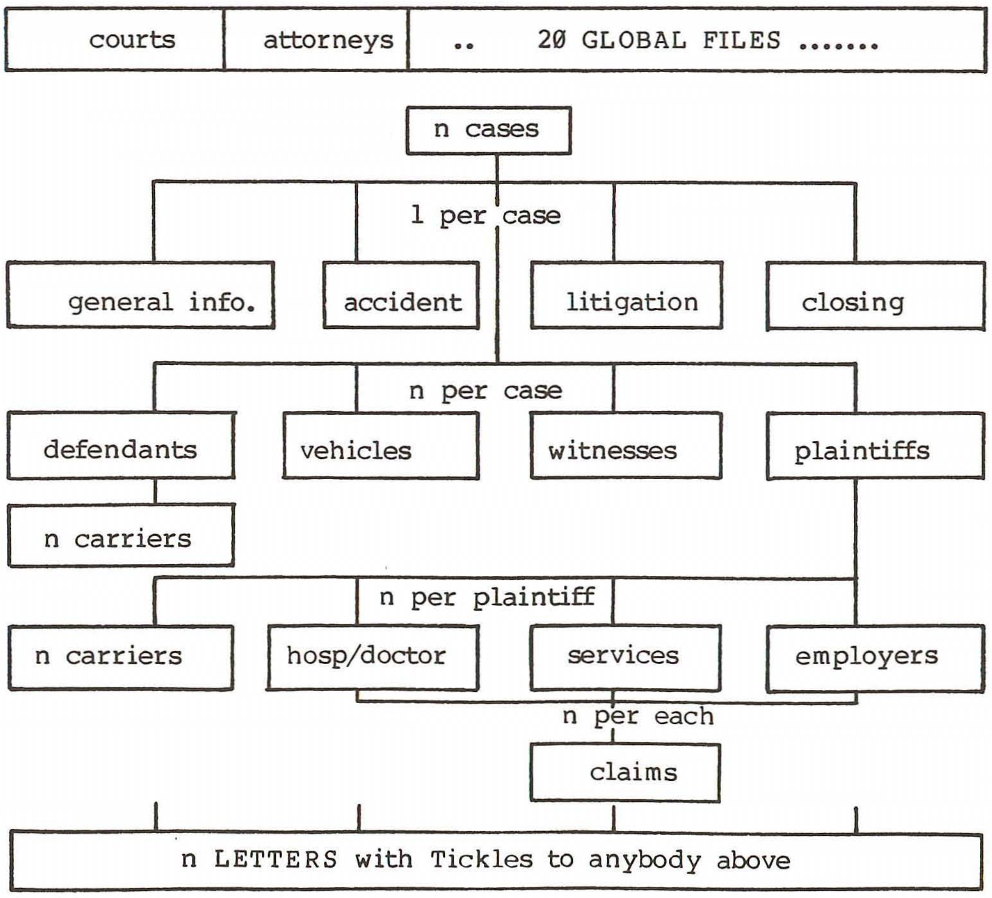
All these instructions and functions considerably reduce the number of lines of code to write, and subsequently the time spent in coding and testing, which has alway been one of the important goals of FORTH.

APPLICATIONS

Here are two examples of applications written in HFORTH.

1. PERSONAL INJURY System for Law Offices

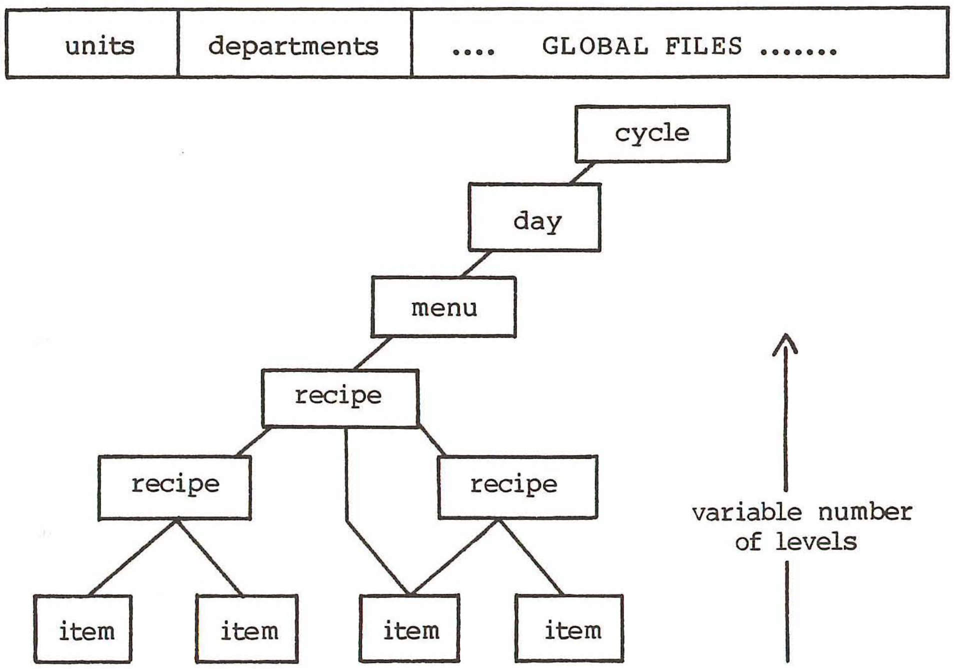
This application manages the database of all the information involved in an action to sue for personal damages that occurred in an accident. The structure of the database is the following:



2. FOOD Management System

This application is used to build recipes made either of basic items or of other previously defined recipes.

A menu is made of several recipes, a day is made of several menus, and finally a cycle is made of several days. An order can be entered at any level, and the system scales the various paths in units and quantities to give the order to the vendors, to the kitchen, and to the inventory. It also performs a Cost/Sales analysis. The basic database design is the following:



THE ROBOT

THE ROBOT is a program generator which can generate business applications that do not require a complex database structure. It can, however, handle a 2 level tree structure, and any relation between an almost unlimited number of files. It includes a Report Generator and a Word Processor capable of retrieving any information from the database. Its features are very similar to DBASE II, but with a greater ease of use by the fact that it is entirely menu driven and does not require any programming at all.

Here is the automatic documentation that the user gets from his entries in the generator.

```
------------------------------------------------------------------------
  SYSTEMS INC.        THE ROBOT : BILLING            DATE : Ø6/12/85
                FILES Define the files
                DIP   Display a record on printer Nbre : 7

------------------------------------------------------------------------
  FILE NAME ..........LINES
  FILE CODE ..........LIN
  [ MASTER FILE NAME ].INVOICES
  RECORD LENGTH .........58
  GENERATED? ..........Yes


L# FIELD NAME      INDEX TYPE   INT DEC CONTROL        O/D FILES/CALCULAT
------------------------------------------------------------------------

KY CHRONO.LINES      Yes Number   5  Ø                 INC


Ø3 INVOICES          Yes Number   3  Ø                 ENT

Ø4 PRODUCT#              Number   3  Ø                 ENT

Ø5 PRODUCT.NAME          Text    25                    R/M PRODUCTS

Ø6 CATEGORY          Yes Text     3                    R/O PRODUCTS

Ø7 UNIT.PRICE            Number   3  2                 R/M PRODUCTS

Ø8 QUANTITY              Number   3  Ø                 ENT
                                                       TO+ PRODUCTS
Ø9 TOTAL.PRICE           Number   5  2                 CAL Ø7+Ø8
                                                       TO+ PRODUCTS
1Ø AMOUNT                Number   5  2                 CAL Ø9
                                                       TO+ INVOICES

------------------------------------------------------------------------
REPORT DEFINITION :
------------------------------------------------------------------------

   REPORT CODE ..........LIN1
   REPORT NAME ..........Ordered list of
   NAME (continued)......LINES
   FILE NAME ............LINES
   REPORT INDEX..........INVOICE#
   REPORT WIDTH............72
   GENERATED? ...........Yes


   NAME OF FIELDS      INDEX SEQ         TOTAL      WIDTH HEADING
------------------------------------------------------------------------

01 CHRONO.LINES         Yes   2                        7  CHRONO.
Ø2 INVOICE#             Yes   6                        4  INV#
Ø3 CATAGORY             Yes   7                        3  CAT
Ø4 PRODUCT#             Yes   8                        4  PROD
Ø5 PRODUCT.NAME               9                       25  PRODUCT.NAME
Ø6 UNIT.PRICE                1Ø                        7  UNIT.PR
Ø7 QUANTITY                  12                        4  QUAN
Ø8 TOTAL.PRICE               14          Yes          1Ø  TOT.PRICE
```

The generator creates an HFORTH source code from the specific entries of the user, which is then automatically compiled and unloaded on disk in various file overlays. The user menu is generated, too. For larger applications this code can be used as a basis, leaving to the professional programmer only the most complex part of the application.

CONCLUSION

These two products demonstrate, assuming this is still to be done, the high capability of FORTH to compete with the most popular products in a domain for which it was obviously never designed. I personally made the decision to develop all my tools and packages in FORTH in 1978, and today I am glad I did.