Identifiers

Siem Korteweg
J. v. Stolberglaan 16
3931 KA Woudenberg
The Netherlands

Hans Nieuwenhuijzen
Sterrewacht "Sonnenborgh"
Zonnenburg 2
3512 NL Utrecht
The Netherlands

Abstract

This paper formalizes the use of identifiers for the identification of dictionary entries.

Identifiers Identical to Strings

Throughout this paper, strings and identifiers will denote a row of characters and a count. Character rows of strings consist of exactly 'count' characters, whereas character rows of identifiers might contain fewer characters (cf. older FORTH versions that used only the first three characters of a word for the identifier in the headers).

Whenever a word of a program is read, the FORTH compiler converts it to a string. Then FORTH searches among the identifiers of the headers in the dictionary for a 'hit'. The criterion for a string to be a hit for an identifier is:

1. The character rows of the string and the identifier must be identical.
2. The counts of the string and the identifier must be equal.

When condition 1 is met, condition 2 is also met, but other kinds of identifiers can exist that require an analogous condition that is nontrivial. We will define different kinds of identifiers and we will divide the headers into two groups according to the differences between their identifiers. The following is a generalization of the 'whatever' technique discussed in [JOOS82].

When we want to use printable characters we have to know the 'value' of their internal representation (e.g. ASCII). We also want to be able to manipulate characters directly from FORTH. Let us define the definition '&' to identify the ASCII value of the character following the '&' character, i.e., '&A' pushes 65 (decimal) on the stack. We could construct an entry in the dictionary for every ASCII character, but then we would be very generous with memory. It would be handy to have only one entry in the dictionary (viz. for '&'). The problem is how to find the identifier of definition '&' when we read the string '&A' or '&B'. To do this we use a new type of identifier. We construct a dictionary entry for the definition '&' that will have an identifier consisting of character row '&' and count 2. We will denote this identifier as '&^' (the '^' character stands for any character that can be used within an identifier). The criterion for a string to be a hit for this identifier is:

1. The character row of the string must start with '&'.
2. The count of the string must be 2.

Conditions 1 and 2 mean that '&' must be followed by exactly one character. Note that the count (=2) is larger than the number of characters in the character row (=1). This implies that this kind of identifier is not compatible with strings whose character rows contain exactly N number of characters (where N = count).

We define a file system (cf. [JOOS81]) to have commands of type:

<FILE-COMMAND> <SYSTEM-ATTRIBUTE>:<FILENAME>

These two commands should cause the execution of the definition <FILE-COMMAND> on the file <FILENAME> , which is to be found on device <SYSTEM-ATTRIBUTE> . As in the case of '&' , we do not know in advance what string will follow after <SYSTEM-ATTRIBUTE>: . As above, we define one entry in the dictionary for the definition <SYSTEM-ATTRIBUTE>: . But, what value should the count of the identifier of this definition have? We do not know the length of the identifier <FILENAME> in advance; we only know its minimum length (one character) and its maximum length (implementation dependent). To solve this problem we use the following criterion for a string to be a hit for this identifier:

1. The character row of the string must start with the character row of the identifier (i.e. with <SYSTEM-ATTRIBUTE>:).
2. (the count of string) ≥ (length of <SYSTEM-ATTRIBUTE>: + 1).

Conditions 1 and 2 mean that <SYSTEM-ATTRIBUTE>: must be followed by at least one character; the generalization of the conditions is obvious. Condition 2 means that the count does not fully matter. We see that this kind of identifier is also incompatible with strings.

Thus there are four possible kinds of headers:

count = length of character row	count > length of character row	remarks
4 test 4 TEST	6 test 6 TEST**	count does matter
4 test N+4 TEST*. . . .* NCHARS	6 test N+6 TEST***. . . .* NCHARS	count does not fully matter i.e. N > = 0

Where:

— * denotes any printable character.
— lowercase words denote the identifiers in the headers.
— uppercase words denote the strings we are searching with.

The case that count < length of the character row is not meaningful as it means that we have more characters in the character row than we check for.

The headers whose identifiers are compatible with strings, (i.e., the identifiers with: 'count does matter' and count = #chars) will be called Normal headers. All the others will be called Special headers. The reason to call them Normal and Special, is that the majority of the headers will be of the first kind.

The implementation of Special headers does not need to afflict the dictionary structure. It will only require a generalization of the routine comparing identifiers in headers with the strings with which we are searching. This routine must react upon the above mentioned properties of the headers, which means that headers have to contain information to mark the different properties. Two bits per header will suffice for this. Note that this is an extension of the fact that the property precedence is stored in the headers (usual Forth implementations).

For most headers count does matter, so we will implement the assignment of this property analogously to the precedence property. That is, for all headers count does matter, except when the routine COUNT.NOCARE is used analogously to the routine IMMEDIATE . The count=#chars property can be assigned by means of assignments to the user-accessible value WIDTH , denoting the maximum number of characters to be stored in the character rows of identifiers.

Special and Normal headers can be mixed freely in the dictionary. Note however, that an identifier having character row 'A' whose count does not fully matter, redefines all the previously defined routines whose identifiers start with a 'A'.

Conclusions

Special headers give a lot of memory saving as each of them represents a whose class of identifiers.

Implementation of special headers causes some time overhead for the searching routine because of the generalized comparison routine that is to be used.

Allocation of a status in the header offers the possiblity to select headers on a number of criteria. This is useful to expand the system's possibilities.

Acknowledgements

We wish to thank prof. dr. J. v. Leeuwen of the department of Computer science of the State University of Utrecht for giving one of us the opportunity to work a year at the Observatory for the final phase of his study. We also wish to thank Rieks Joosten and Frans Cornelis for the time they spent discussing and analyzing this concept and its use, and Hans v. Koppen for the use of his room and his Apple II computer.

To obtain a copy of the FYS FORTH manual of FYS FORTH 0.3 please contact: Hans Nieuwenhuijzen, Sterrewacht "Sonneborgh", Zonneburg 2, 3512 NL Utrecht, the Netherlands.

References

[JOOS81] R. Joosten, *FYS FORTH 0.2/0.3 Preliminary User Manual*, State University Utrecht, Utrecht the Netherlands, 1981

[JOOS82] R. Joosten, The 'Whatever' Technique *Proceedings of The 1982 Rochester FORTH Conference On Data Bases And Process Control,* Rochester, New York, pp 270–271.

Manuscript received June 1984.