OBJECT ORIENTED LOCAL VARIABLES / DATA STRUCTURES FOR F83

ABSTRACT

An integrated package of compiler extensions providing symbolic object oriented programming for local variables and data structures is described. The user syntax is simple, consistent, and greatly enhances programming ease and readability of the resultant code. The data structures compiler uses extended memory (32 bit addressing) operations to permit full use of available memory. The approach emphasizes runtime efficiency at the expense of a small increase in compiler complexity.

This paper summarizes the features of the extension package. A paper discussing the details of the approach and implementation has been submitted to the Journal of FORTH Application and Research.

EXTENDED MEMORY OPERATIONS

A set of extended memory operators based on a 32 bit address or pointer is added to the F83 virtual machine emulation. These operators (X@, X!, X2@, X2!, etc.) are discussed in a companion paper entitled "Extended Memory Operations for F83".

The memory immediately after the 64 kilobyte dictionary segment is defined as a HEAP memory resource and is allocated as needed by the word HEAPALLOT.

OBJECT ORIENTED DATA STRUCTURES

The Data Structures compiler is an example of what might be termed a 'phrase compiler'. The approach makes use of a set of defining words to generate a tree of 'datatype' elements which define the phrase compiler action. Each element of the tree resolves the next level in terms of a limited 'local' vocabulary instead of the currently active search order. Control is returned to the normal Interpreter/Compiler only after the phrase has been completely processed.

The Data Structures defining syntax is similar to that described by Pountain [POUN86] in the Aug. 86 BYTE. A new datatype is defined by the sequence:

```
TYPE> <newdatatypename>
<subelement declarations> or <size declaration>
<newdatatype local operations>
ENDTYPE> <newdatatypename>
```

A defining word <newdatatypename> is created in the CURRENT vocabulary which may be used to create specific instantiations of

definition. The symbols i1 to ik map onto the input stack configuration and must agree in number, size (16 or 32bit) and order with the expected inputs. The compiler symbol < separates those inputs i1 to kh that are to remain from those inputs ih+1 to ik that are to be discarded. The symbol / separates the inputs from any temporary variables t1 to tm. Finally the symbol > indicates the beginning of the output variables o1 to oj which determine the number, size and order of the outputs left on the stack after execution of the word <name>. Any local symbol may represent a 32 bit quantity by following it (after a space) with the symbol #. The closed parenthesis ends the declaration phase and a stack frame size literal followed by a stack frame setup word F: are automatically compiled to begin the body of the definition. Normal compilation of <body of definition> follows. The redefinition of {;} compiles a stack cleanup sequence and flushes the local symbols before executing the normal {;}.

Input and output 'lists' (unspecified number of elements) on the stack and pointers to data structures may be declared and processed using additional features described in the full paper. The local pointer variables allow any area of memory to be overlayed by any datatype structure and processed symbolically.

The remainder of the definition is compiled in the same manner as any other FORTH word. Reference to a local symbol results in a stack frame offset being compiled as either a constant or a literal. The programmer then uses a stack frame fetch or store (S@,S!,S2@,S2!) similar to ordinary variables. In addition, the mechanisms of the Data Structures compiler are used to provide generic @ and ! operations for both 16 and 32 bit local variables.

The symbols which appear in a local variable declaration list exist only during the compilation of that definition. Those symbols are forgotten and the memory reclaimed when compilation is complete. Since the local symbols are at the top of the search order during the compilation they will conveniently mask other words in the vocabulary with identical names.

An example of using the local variable compilation mode to code a second order integer polynomial y=a+bx+cx2 = a+x(b+cx) would be:

```
: IPOLY   F( A B C X > Y )
    C @  X @  *  B @  +  X @  *  A @  +  Y !  ;
```

Source code is available on East Coast Forth Board (703-442-8695) in file ROHDAF83.BLK.

REFERENCE

[POUN86] Pountain, Dick; "Object-Oriented FORTH", BYTE Magazine, August 1986.