Algorithms

A Group Construct for Field Words

Contributed by Charles B. Duff, Software Manager,
Kriya Systems, Inc., 505 N. Lakeshore #5510, Chicago, IL 60611

Glen Haydon's very useful field definition words [HAY81], [WAT82] are made more useful with the addition of a GROUP construct that allows the aggregation of several fields into a repeating group. This accommodates situations in which a record definition contains an array or more complex indexed structure that should be accessed symbolically. Haydon's field definition words are based upon the dual compile/run time behavior of defining words. Referring to the definition of TFIELD, (see screen #9), its compile time behavior is to store two values in its parameter field: the start offset of the field and its length. A running offset is then accumulated and left for the next field word to use. The runtime behavior of FIELD is to add its offset to the base address passed on the stack, and then push its length, leaving (addr1—addr2 length), which is suitable for words like TYPE or CMOVE.

I have added 3 words to Haydon's scheme as a way of implementing repeating groups: GROUP, TIMES, and ENDG on screens 12-14. GROUP allocates storage for a count (number of repeating elements) and a length at compile time. The count is filled in by TIMES when used in a phrase such as GROUP NAMES 4 TIMES. GROUP leaves the compile-time offset on the stack, and pushes a 0 to serve as a new base offset for fields within the group. The fields are defined, accumulating a running offset, which is actually the length of the group. ENDG then uses this to fill in the GROUP's length and compute a new offset that is the previous offset plus the number of occurrences times the group length. ENDG consumes a stack element, dropping out to the previous level. The Forth stack-based architecture is very convenient in situations like this, because it allows indefinite nesting of GROUPs with each base address preserved on the stack.

Screen 14 demonstrates the use of the GROUP words. The word FILLER on screen 15 is useful when you want to allocate room in a record without having to name it. It simply increments the running compile-time offset without creating a dictionary entry.

Haydon's field words are very useful for mapping an arbitrary buffer or data structure with symbolic names, and can go a long way to clarify source. Although they were originally presented as part of a larger database scheme, they stand alone quite well.

References

[HAY81] G. Haydon, "Elements of a Forth Data Base Design", *Dr. Dobb's Journal*, Volume 6, No. 9 September, 1981.

[WAT82] R. Watkins, "The INDEXER: Enhancements to a Data Base Model", *Forth Dimensions*, Vol. 4, No. 5.

Screen # 7
```
( Field definition words                                    CBD 11/02/83 )
```

-->
These words can be used to describe a template for an arbitrary area of memory. The GROUP construct enables nested repeating groups of fields.

Sequence 2: The stack is used to maintain a running offset into the current group or record. A GROUP will add another value to the stack, and field statements update the top stack value to obtain the next offset.

Sequence 3: A field reference, when passed an address, will return a new address reflecting its stored offset value, and possibly the length of the field.

Screen # 8
```
( Field words - field                                       cbd 07/20/83 )

( strt len --- nuoffs)  ( Define an n-byte field )
( Returns address of field at sequence 3 )
: FIELD     <BUILDS                      ( Stores strt,len in pf)
               OVER , +                  ( Leaves cum cnt )
            DOES>    ( Runtime: offset --- offset+strt )
               @ + ;
-->
Examples:     <Seq 2>
  0 20 FIELD F1     ( 0  20 --- 20 )
     10 FIELD F2    ( 20 10 --- 30 )
              <Seq 3>
  100 F1 .S     100 OK
  100 F2 .S     120 OK
```

Screen # 9
```
( Field words - tfield                                      cbd 07/20/83 )

( strt len --- nuoffs)  ( Define a text field )
( Returns address and length of field at sequence 3 )
: TFIELD    <BUILDS                      ( Stores strt,len in pf)
               OVER , DUP , +            ( Leaves cum cnt )
            DOES>    ( Runtime: offset --- offset+strt len )
               2@ ROT +
               SWAP ;
-->
Examples:     <Seq 2>
  0 20 TFIELD F1     ( 0  20 --- 20 )
     10 TFIELD F2    ( 20 10 --- 30 )
              <Seq 3>
  100 F1 .S     100 20 OK
  100 F2 .S     120 10 OK
```

Screen # 10

```
( Field words - wfield                                         cbd 07/20/83 )

( offs --- offs+2 )  ( Define a word field )
( Returns address of field at sequence 3 )
: WFIELD     <BUILDS                    ( Stores strt in pf)
                 DUP , 2+               ( Leaves cumulative count )
             DOES>     ( Runtime: offset --- offset+strt )
                 @ + ;
-->
Examples:      <Seq 2>
  0 WFIELD F1      ( 0 --- 2 )
    WFIELD F2      ( 2 --- 4 )
               <Seq 3>
  100 F1 .S      100 OK
  100 F2 .S      102 OK
```

Screen # 11

```
( Field words - bfield                                         cbd 07/20/83 )

( offs --- offs+1 )  ( Define a byte field )
( Returns address and length of field at sequence 3 )
: BFIELD     <BUILDS                    ( Stores strt in pf)
                 DUP , 1+               ( Leaves cumulative count )
             DOES>     ( Runtime: offset --- offset+strt )
                 @ + ;
-->
Examples:      <Seq 2>
  0 BFIELD F1      ( 0 --- 1 )
    BFIELD F2      ( 1 --- 2 )
               <Seq 3>
  100 F1 .S      100 OK
  100 F2 .S      101 OK
```

Screen # 12

```
( Field words - group                                    cbd 07/20/83 )

( offs --- offs 0 )  ( Begin a group of fields )
( Stores group length and occurrences for later use. )
( Occurrences must be set with TIMES, length with ENDG )
: GROUP     <BUILDS DUP , 0
                          HERE 4 ERASE 4 ALLOT ( room for occ, len)
( addr occ# --- nuaddr )  ( Converts address and occurrence to )
( beginning address of that occurrence )
                DOES>  >R  R 2+ @            ( get occ on stack )
                      OVER  < IF R>
( This can be rem- )   CR ." Group index out of bounds " CR
( oved after tested)  ABORT ELSE
                      1- R 4 + @ * R> @ + +
                      THEN ;  ( offs + ind-1 * len + strt )
-->
```

Screen # 13

```
( Field words — times endg                               cbd 07/20/83 )

( occ --- )  ( Store occurrences in last group defined )
( Must be used after any use of GROUP )
: TIMES     LATEST  PFA 4 + ! ;

( offs len --- nuoffs )
( Stores the group length in named group's pfa )
: ENDG          DUP -FIND IF DROP >R R 6 + ! ( Group len in pf )
                          R> 4 + @ * + ( add occ*len to stack addr)
                       ELSE ABORT  ( group not found )
                       THEN ;

-->
```

Screen # 14

```
( Field words — Examples of use                                   cbd 07/20/83 )
-->
;S     <Seq 2>
0  WFIELD  F1                    ( 0      --- 2 )
   WFIELD  F2                    ( 2      --- 4 )
   GROUP  G1  3 TIMES            ( 4      --- 4 0 )
      10 FIELD  F3               ( 4 0    --- 4 10 )
      20 FIELD  F4               ( 4 10   --- 4 30 )
   ENDG  G1                      ( 4 30   --- 94    )
DROP

      <Seq 3>
100     F2      S.  102 OK
100     F4      S.  110  20 OK
100  2 G1       S.  134 OK       ( 2nd occurrence of G1 )
100  2 G1  F4 S.  144  20 OK     ( F4 within "          " )
```

Screen # 15

```
( Field words — filler                                            cbd 11/08/83 )

( offs --- offs+n )  ( Adjust compile-time offset to skip unneeded data )

: FILLER          + ;  IMMEDIATE    ( execute at compile time )


;S
```