.ORPHANS

An Unreferenced Code Finder

Jamison H. Abbott
ibidinc
Suite 607
179 Allyn St.
Hartford, CT 06103

ABSTRACT

For a variety of reasons a program may contain compiled code that is not used by it. This tool, .ORPHANS, allows the Programmer to find that unused or 'orphan' code so that it may be removed. The choice of a very 'Forth-like' implementation of this tool, relying heavily on the data stack and the dictionary, instead of a more classical approach, using a symbol table, results in trade-offs in speed versus generality. This paper also briefly discusses how .ORPHANS can be used for pruning a Forth nucleus; this can be particularly useful in target-compiling.

INTRODUCTION

This tool was developed as a result of both the all too common need to fit more code into a 64K dictionary and my suspicion that some of the compiled code that was already in the dictionary was not being used. One reason that some of the code in the dictionary might have become unused is due to human error on the Programmer's part--usually as a side effect of revising the source code. Moreover, if one is utilizing some unfamiliar library routines, it is quite possible that some of the routines in the library are not needed by the final application. The detection of those unreferenced routines is not an easy task for the Programmer if the library is large; with this tool, it becomes semi-automated. In addition, much of the nucleus code may also be unreferenced by an application; this tool can find that code as well.

DESIGN & IMPLEMENTATION

The design is based on knowledge of the implementation of the dictionary structure. We traverse the dictionary from top (latest compiled word) to bottom. For each word encountered in the dictionary we look at the data stack to see if that word's CFA is in it. If it is, we remove the CFA and continue down to the next word in the dictionary. If it's not there, we print out a message flagging that word as an orphan and continue down to the next word in the dictionary.

However, before continuing down to the next word in the dictionary we check to see if the current word is a colon

definition. If it is, we take all the CFAs that make up the body of the word (the 'constituent' CFAs) and put them on the stack. To prevent the stack from growing too large we check the stack for CFAs that match the constituent ones so that we never put any duplicate CFAs on the stack.

This process continues until either the FENCE is reached or until the bottom of the dictionary is reached (see listing). For those whose implementation does not include FENCE, it may be defined as:

```
FENCE   --- addr        A variable that contains an
        address below which FORGETting is trapped.  [LMI84]
```

APPLICATION

To use .ORPHANS follow the instructions (see listing) on screens four and five respectively for finding unreferenced code and for pruning the nucleus. Because .ORPHANS doesn't always handle certain cases correctly (see below) any words that references with the search function of your editor.

CONCLUSION

.ORPHANS has been used sucessfully in the development of several programs. However, since .ORPHANS makes such intimate use of the dictionary, the source code for it may have to be modified to conform to a particular user's implementation of Forth. Although an implementation of .ORPHANS which used a symbol table instead of relying on the dictionary would probably more accurately handle such cases as: Vectored execution, QUANs and other multiple-CFA words, Children of defining words, Words that are only referenced inside Code words, Recursive definitions and Vocabularies (these cases are not usually handled correctly by .ORPHANS); a symbol table implementation would, most likely, also be much slower [FEI85].

REFERENCES

[LMI84] Laboratory Microsystems Incorporated, PC/FORTH *Language Reference Manual*

[FEI85] Gary Feierbach and Paul Thomas, *Forth tools and applications*, Reston Publishing Company, Inc., 1985 (CONCORDANCE program: pp. 30-34)

APPENDIX -- Non Forth-83 Words -- [LMI84]

```
.NAME     addr ---      Given the address of the name  field
          of  a dictionary header,  displays the name on the
          current output device.
ALIGN     addr1 --- addr2      If addr1 is odd,  round it  up
          to the next higher address
>NAME, >LINK, >BODY, NAME>     (field address conversion words)
```

Screen # 4

```
( .ORPHANS: UN-REFERENCED CODE   FINDER        14:13 11/12/85 )
```

To use .ORPHANS to find unreferenced code:

First, clean out the dictionary by performing a COLD. Then, load the .ORPHANS screens. Next, load your program's source code on top of .ORPHANS. Last, invoke the utility by typing the word .ORPHANS and answering the prompts. When the word .ORPHANS is interpreted the NFA of your program's last (top-most in the dictionary) word should be on top of the stack: (e.g. LATEST .ORPHANS). You will then be asked if you want the trace on. Answer 'Y' if you want to see the full execution of the utility including the CFA's of all the words that it encounters in the dictionary. If you answered 'Y', then you will also be asked if want to single-step thru the execution of the utility.

Screen # 5

```
( .ORPHANS: UN-REFERENCED CODE   FINDER        14:13 11/12/85 )
```

To use .ORPHANS to prune the Forth nucleus:

The procedure in this case is similar to that when finding unreferenced code; however, in this case you should load your program's source code FIRST and then load .ORPHANS on top of it! BUT, be careful to invoke .ORPHANS with the NFA of the topmost word of your application (use WORDS or VLIST to see) and NOT the topmost word of .ORPHANS! (Note: this procedure will prevent .ORPHANS from 'seeing' itself so it won't be counted in the referenced code.)

Also, since .ORPHANS stops when it encounters the 'fence' marker you should deactivate the fence by setting to zero (the bottom of the dictionary).

Screen # 6

```
( .ORPHANS: UN-REFERENCED CODE   FINDER    jha 17:01 04/12/85 )
( Laboratory Microsystems -- PC/FORTH 3.0 -- FORTH '83 )
( Modified jha 8/9/85: added skip-over on literal strings )
' unnest  CONSTANT  ;-CFA
VARIABLE CURR-CFA
VARIABLE SAVE-PFA
0 CONSTANT DUMMY
     ( BELOW: Get CFA's of non code-type definitions )
' FORTH @    CONSTANT vocabulary-CFA ( 0327h )
' S0 @       CONSTANT user-CFA       ( 0318h )
' CURR-CFA @ CONSTANT variable-CFA   ( 0308h )
' DUMMY @    CONSTANT constant-CFA   ( 0308h )
' WORDS @    CONSTANT colon-CFA
' SOURCE @   CONSTANT source-CFA

' litq CONSTANT (.")-CFA
```

Screen # 7

```
( .ORPHANS: UN-REFERENCED CODE   FINDER    jha 17:01 04/12/85 )
( Laboratory Microsystems -- PC/FORTH 3.0 -- FORTH '83 )
: .TYPE   ( cfa -- / display type of definition )
   SPACE DUP DUP @ 2- =
   IF   DROP ." ( code )"
   ELSE @
      CASE
         colon-CFA       OF ." ( colon )"      ENDOF
         variable-CFA    OF ." ( variable )"   ENDOF
         constant-CFA    OF ." ( constant )"   ENDOF
         user-CFA        OF ." ( user )"       ENDOF
         vocabulary-CFA  OF ." ( vocabulary )" ENDOF
         source-CFA      OF ." ( source )"     ENDOF
         ." ( unknown ) "  7 EMIT
      ENDCASE
   THEN ;
```

Screen # 8

```
( .ORPHANS: UN-REFERENCED CODE   FINDER    jha 17:01 04/12/85 )
( Laboratory Microsystems -- PC/FORTH 3.0 -- FORTH '83 )
VARIABLE SINGLE-STEP
VARIABLE TRACE
: N.  ( n -- ) DUP U. HEX 0 (# 41 HOLD 104 HOLD # # # # #)
   TYPE DECIMAL ;
: NAME&CFA   ( CFA -- / displays the name and CFA )
   DUP >NAME .NAME ." (CFA= " N. ;
: .ORPHAN  ( -- ) CURR-CFA @ CR DUP DUP >NAME .NAME
   ."  --IS AN ORPHAN " ."  (CFA=" N.  .TYPE ;
: MESSAGE1 CR ." Now Below the Fence -- Quitting..." CR ;

: MESSAGE2   ( -- ) CR CR ." Starting the Run... " SINGLE-STEP @
   IF CR ." ( Press any key to continue the trace ) " THEN ;
: MESSAGE3   ( -- ) CR CR CR ." Do you want trace on? (Y/N) " ;
: MESSAGE4 CR CR ." Do you want single-step on ? (Y/N)  " ;
```

Screen # 9

```
( .ORPHANS: UN-REFERENCED CODE   FINDER    jha 17:01 04/12/85 )
( Laboratory Microsystems -- PC/FORTH 3.0 -- FORTH '83 )

: EMPTY   ( a b c ... --- / empties the stack of all items )
   DEPTH 0 ?DO  DROP  LOOP ;

: .AT   ( --- / DISPLAY THE DEF. THAT WE ARE CURRENTLY AT )
   TRACE @
   IF CURR-CFA @ DUP
      CR 12 SPACES ." AT: " NAME&CFA .TYPE
   THEN ;
```

Laboratory Microsystems PC/FORTH 3.00 09:13 05/12/86 f:orphans.scr

Screen # 10

```
( .ORPHANS: UN-REFERENCED CODE   FINDER    jha 17:01 04/12/85 )
( Laboratory Microsystems -- PC/FORTH 3.0 -- FORTH '83 )
VARIABLE NOT-ON-STACK-FLAG
VARIABLE S-MATCH
: ?NOT-ON-STACK  ( n -- f / Search the data stack for a match,
   drop it from the stack if found.  Leave a result flag. )
   S-MATCH !  1 NOT-ON-STACK-FLAG !
   DEPTH 0
   ?DO
      I PICK  S-MATCH @ =
      IF    I ROLL DROP   0 NOT-ON-STACK-FLAG !   0
      ELSE  1
      THEN
   +LOOP
   NOT-ON-STACK-FLAG @
;
```

Screen # 11

```
( .ORPHANS: UN-REFERENCED CODE   FINDER    jha 17:01 04/12/85 )
( Laboratory Microsystems -- PC/FORTH 3.0 -- FORTH '83 )

VARIABLE ALREADY-ON-STACK-FLAG
VARIABLE CFA-MATCH
: ALREADY-ON-STACK ( n -- n f / Search the data stack for a
   match.  Leave then match value and a result flag. )
   CFA-MATCH !  0 ALREADY-ON-STACK-FLAG !
   DEPTH 0
   ?DO
      I PICK  CFA-MATCH @ =
      IF    1 ALREADY-ON-STACK-FLAG !
      THEN
   LOOP
   CFA-MATCH @ ALREADY-ON-STACK-FLAG @
;
```

Screen # 12

```
( .ORPHANS: UN-REFERENCED CODE   FINDER    jha 17:06 04/12/85 )
( Laboratory Microsystems -- PC/FORTH 3.0 -- FORTH '83 )
: PUSH-EM ( -- a b ... \leaves the CFA's on the stack )
   CURR-CFA @ >BODY DUP   BEGIN  @   ( curr-PFA call-CFA )
   DUP (.")-CFA = ( check for strings\curr-PFA call-CFA flag)
    IF DROP 2+ DUP C@ + 1- ALIGN DUP @ THEN ( curr-PFA call-CFA)
       DUP ;-CFA <>           ( -- curr-PFA call-CFA flag )
   WHILE                      ( -- curr-PFA call-CFA )
      DUP CURR-CFA @ U< NOT      ( check for fwd branch)
      OVER >NAME FENCE @ U< OR    ( below fence ? )
      IF   DROP
      ELSE SWAP   SAVE-PFA ! ALREADY-ON-STACK IF DROP THEN
           SAVE-PFA @
      THEN 2+ DUP
   REPEAT   DROP DROP ;
```

Screen # 13

```
( .ORPHANS: UN-REFERENCED CODE   FINDER    jha 17:01 04/12/85 )
( Laboratory Microsystems -- PC/FORTH 3.0 -- FORTH '83 )
0 CONSTANT BOTTOM   ( Bottom of Dictionary ? )
: NEXT-DEF.   ( --- / put cfa of next definition in CURR-CFA )
   CURR-CFA @ >LINK @
   DUP BOTTOM U> NOT ABORT" Bottom of Dictionary"
   NAME>           ( -- nextCFA )
   CURR-CFA ! ;
: PUSH-CFA'S  ( --- a b c ... / pushes CFA's on the stack unless
                the current definition is not a colon definition.)
   CURR-CFA @
   DUP DUP @ 2- =
   IF DROP
   ELSE @ colon-CFA = IF PUSH-EM THEN
   THEN
;
```

Screen # 14

```
( .ORPHANS: UN-REFERENCED CODE   FINDER    jha 17:01 04/12/85 )
( Laboratory Microsystems -- PC/FORTH 3.0 -- FORTH '83 )
: .ORPHANS ( nfa -- / prints the names of uncalled definitions )
   DUP FENCE @ U< NOT
   IF   MESSAGE3 KEY ASCII Y =
        IF   MESSAGE4 KEY ASCII Y = IF 1 ELSE 0 THEN 1 ELSE 0 0
        THEN TRACE ! SINGLE-STEP !
        MESSAGE2 NAME> CURR-CFA ! .AT .ORPHAN
        BEGIN
           PUSH-CFA'S NEXT-DEF.  CURR-CFA @ >NAME FENCE @ U< NOT
        WHILE
           SINGLE-STEP @ IF PCKEY ?DUP 2DROP THEN .AT
           CURR-CFA @ ?NOT-ON-STACK IF .ORPHAN THEN
        REPEAT
   ELSE DROP
   THEN EMPTY   MESSAGE1 ( .S )  ;
```

Screen # 15

```
( .ORPHANS: UN-REFERENCED CODE   FINDER  jha   09:57 04/17/85 )
      ( Laboratory Microsystems -- PC/FORTH 1.25 )
    ( Modified jha 8/9/85: added skip-over on literal strings )
HEX     0B61  CONSTANT  ;-CFA   DECIMAL
0 VARIABLE SAVE-PFA
0 VARIABLE CURR-CFA
0 CONSTANT DUMMY
' FORTH CFA @    CONSTANT vocabulary-CFA
' S0 CFA    @    CONSTANT user-CFA
' CURR-CFA CFA @ CONSTANT variable-CFA
' DUMMY CFA @    CONSTANT constant-CFA
' VLIST CFA @    CONSTANT colon-CFA

' (.") CFA CONSTANT (.")-CFA      ( 183E )

: >PFA   ( cfa -- pfa )   2+ ;
```

Laboratory Microsystems PC/FORTH 3.00 09:14 05/12/86 f:orphans.scr