FigForth 1.0d#

An old Forth Version for ATARI 800. Uses traditional Forth blocks for storage.

Screens Disk 1#

```
SCR # 1
  0 ( basic functions )
  1 HEX : BELL FD EMIT ;
  2 DECIMAL
  3 : FREE ( amount of memory left)
  4  MEMTOP @ PAD 256 + - ;
  5 : U. 0 D. ; ( unsigned . )
  6 : NEW-ABORT ( auto-reset )
  7  HERE 2 - ' ABORT DUP @ , ! ;
  8  IMMEDIATE
  9 512 VARIABLE *$* ( $STK size)
 10
 11
 12
 13
 14
 15 -->
```

```
SCR # 2
  0 ( printer routines ) HEX
  1 CREATE (P:) 50 C, 3A C, 9B C,
  2 SMUDGE ' (P:) CONSTANT P:
  3 : PR-ON 70 CLOSE 70 8 0 P: OPEN
  4  1 PRFLAG ! ;
  5 : PR-OFF 70 CLOSE 0 PRFLAG ! ;
  6 : PLIST ( strt end --> )
  7  1+ SWAP C EMIT ( form-feed) CR
  8  DO CR I LIST CR CR CR
  9  ?TERMINAL IF LEAVE ENDIF LOOP
 10  C EMIT CR ;
 11
 12
 13
 14
 15                              -->
```

```
SCR # 3
  0 HEX : :SELECT ( code --> ? )
  1  <BUILDS HERE 1 ALLOT 0
  2  BEGIN BL WORD
  3   HERE 1+ C@ 3B - WHILE
  4   1+ HERE NUMBER DROP C,
```

```
 5   -FIND IF DROP CFA , ELSE
 6    HERE COUNT TYPE
 7    ." UNDEFINED" QUIT ENDIF
 8  REPEAT SWAP C! DOES>
 9  DUP DUP C@ 3 * + 1+ SWAP 1+
10  DO I C@ DUP 0= IF DROP
11    I 1+ @ EXECUTE LEAVE ELSE
12    OVER = IF I 1+ @
13     EXECUTE LEAVE ENDIF
14   ENDIF 3 +LOOP ;
15                          6 LOAD
```

```
SCR # 4
 0 ( ERROR MESSAGES )
 1 EMPTY STACK
 2 DICTIONARY FULL
 3 HAS INCORRECT ADDRESS MODE
 4 ISN'T UNIQUE
 5
 6 DISK RANGE ?
 7 FULL STACK
 8 DISK ERROR !
 9
10
11
12
13
14
15 FORTH INTEREST GROUP
```

```
SCR # 5
 0 ( ERROR MESSAGES )
 1 COMPILATION ONLY, USE IN DEFN
 2 EXECUTION ONLY
 3 CONDITIONALS NOT PAIRED
 4 DEFINITION NOT FINISHED
 5 IN PROTECTED DICTIONARY
 6 USE ONLY WHEN LOADING
 7 OFF CURRENT EDITING SCREEN
 8 DECLARE VOCABULARY
 9
10
11
12
13
14
15
```

```
SCR # 6
  0 : DUMP ( from number --> )
  1  BASE @ >R HEX ( save base)
  2  0 DO  CR DUP I + DUP 0 4 D.R
  3   ." :"  8 0 DO ( 8 byte values)
  4    DUP I + C@ 2 .R SPACE LOOP
  5   1 2FE C! ( display controls)
  6   8 0 DO ( print ASCII)
  7    DUP I + C@ EMIT LOOP
  8   ?TERMINAL IF LEAVE ENDIF
  9   0 2FE C!  DROP  8 +LOOP
 10  DROP CR R> BASE ! ;
 11 : <CMOVE ( from to n -->,hi-lo)
 12  -DUP IF  1 - -1 SWAP DO
 13   OVER I + C@ OVER I + C!
 14   -1 +LOOP  ENDIF  DROP DROP ;
 15                               -->
```

```
SCR # 7
  0 ( I/O definitions )
  1 00 CONSTANT #0   30 CONSTANT #3
  2 40 CONSTANT #4   50 CONSTANT #5
  3 60 CONSTANT #6
  4 2FB CONSTANT ATACHR
  5 54 CONSTANT ROWCRS ( 1 byte)
  6 55 CONSTANT COLCRS ( 2 bytes)
  7
  8
  9
 10
 11
 12
 13
 14
 15 0 VARIABLE (COLOR)           -->
```

```
SCR # 8
  0 : ?IOERR (STAT) @ 7F > IF
  1  BASE @ HEX HERE COUNT TYPE
  2  ."  ? I/O ERROR " (STAT) @ .
  3  BASE ! SP! IN @ BLK @ QUIT
  4  ENDIF ;
  5 CREATE (S:) 53 C, 3A C, 9B C,
  6 SMUDGE ' (S:) CONSTANT S:
  7 CREATE (E:) 45 C, 3A C, 9B C,
  8 SMUDGE ' (E:) CONSTANT E:
  9 : GR. ( mode --> )
 10  ( mode+16=split,mode+32=no clr)
 11  #6 CLOSE   #0 CLOSE
 12  #0 C 0 E: OPEN
```

```
 13  #6 SWAP DUP 30 AND C OR ( aux1)
 14   SWAP  S:  OPEN  ?IOERR
 15  ;                            -->
```

```
SCR # 9
  0 : POS. ( x y --> )
  1  ROWCRS C!  COLCRS ! ;
  2 : COLOR ( col --> )
  3  (COLOR) C! ;
  4 : DR. ( x y --> )
  5  (COLOR) C@ ATACHR C!
  6  POS. #6 11 JSRCIO ;
  7 : LOC. ( x y --> val )
  8  OVER OVER POS.  #6 GET >R
  9  POS. #6 R PUT R> ;
 10 : PL. ( x y --> )
 11  POS. #6 (COLOR) C@ PUT ;
 12 : SE. ( reg hue lum --> )
 13  SWAP 10 * + SWAP 2C4 + C! ;
 14 : STICK ( port --> value)
 15  278 ( STICK0) + C@ ;        -->
```

```
SCR # 10
  0 CREATE (CVTSTK) 4 C, 2 C, 3 C,
  1  FF C, 6 C, 8 C, 7 C, FF C, 5 C,
  2  1 C, 0 C, SMUDGE
  3 : CVTSTK ( value --> 0=nothing,
  4  1=up, 3=right, 5=down, 7=left)
  5  5 - ' (CVTSTK) + C@ ;
  6 : STRIG ( port --> TF,T=not psh)
  7  284 ( STRIG0) + C@ ;
  8 : SO. ( voice pitch dist vol ->)
  9  3 D20F C! 3 232 C! 0 D208 C!
 10  SWAP 10 * + 3 PICK DUP +
 11  D201 ( AUDC1) + C!
 12  SWAP DUP + D200 ( AUDF1) + C!
 13  ;
 14
 15                               -->
```

```
SCR # 11
  0 : SPEMIT  ( char -->,special ch)
  1  1 2FE C!  EMIT  0 2FE C! ;
  2 : XIO ( cmd iocb aux1 aux2
  3         nameaddr --> )
  4  4 PICK >R ( iocb --> ret stk)
  5  R 344 + ! ( store name addr)
```

```
  6  R 34B + C! ( store aux2)
  7  R> 34A + C! ( store aux1)
  8  SWAP JSRCIO ( do cmd) ;
  9
 10
 11
 12
 13
 14
 15 DECIMAL                       -->
```

```
SCR # 12
  0 ( string definitions) HEX
  1 *$* @ ALLOT ( allocate $STK)
  2 HERE CONSTANT $0 ( $STK base)
  3 $0 VARIABLE $P ( $STK pointer)
  4 : $P! ( --> , reset $STK)
  5  $0 $P ! ;
  6 : $P@ ( --> $P value)
  7  $P @ ;
  8 : $DROP ( $ -$> )
  9  $P@ DUP C@ + 1+ $P ! ;
 10 : $LEN ( --> len of top $ )
 11  $P@ C@ ;
 12 : $FETCH ( addr len --> , -$> $)
 13  $P@ OVER 1+ - DUP $P ! ( upd$P)
 14  OVER OVER C! ( store len)
 15  1+ SWAP CMOVE ( move str) ; -->
```

```
SCR # 13
  0 : $STORE ( addr max --> actual,
  1            $ -$> )
  2  $LEN MIN >R ( actual length)
  3  $P@ 1+ SWAP R CMOVE
  4  R> $DROP ;
  5 : $VARLEN ( vaddr --> len )
  6  1 - C@ ;
  7 : $VARMAX ( vaddr --> max len )
  8  2 - C@ ;
  9 : $@ ( vaddr -->, -$> $ )
 10  DUP $VARLEN $FETCH ;
 11 : $! ( vaddr -->, $ -$> )
 12  DUP DUP $VARMAX $STORE
 13  SWAP 1 - C! ;
 14 : $. ( $ -$> , print string)
 15  $P@ 1+ $LEN TYPE $DROP ;    -->
```

```
SCR # 14
  0 : $DUP ( $ -$> $ $ )
  1  $P@ 1+ $LEN $FETCH ;
  2 : $P2@ ( 2nd$P) $P@ $LEN + 1+ ;
  3 : $LEN2 ( 2ndlen) $P2@ C@ ;
  4 : $OVER ( $1 $2 -$> $1 $2 $1 )
  5  $P2@ 1+ $LEN2 $FETCH ;
  6 : $SWAP ( $1 $2 -$> $2 $1 )
  7  $P@ >R $LEN ( $2 length)
  8  $OVER $LEN + 2+ ( $1$2$1, len)
  9 $P@ SWAP R SWAP <CMOVE ( $2$1$1)
 10  R> $P ! ; ( pop extra $1)
 11 : $+ ( $1 $2 -$> $1+$2 )
 12  $SWAP  $LEN2 $LEN +
 13  $P@ 1+ DUP 1+ $LEN <CMOVE
 14  ( remove embedded length)
 15  1 $P +! $P@ C! ; ( $P,len)  -->



SCR # 15
  0 : $FILL ( n b -->, -$> n b's )
  1  OVER 1+ $P@ SWAP - DUP $P !
  2  ( alloc space )
  3  3 PICK OVER C! ( store length)
  4  1+ ROT ROT FILL ;
  5 : $VARFILL ( vaddr b --> )
  6  OVER DUP $VARMAX SWAP $VARLEN -
  7  -DUP IF ( any left to fill?)
  8   >R OVER $@
  9   R> SWAP $FILL $+ $!
 10  ELSE DROP DROP ENDIF ;
 11 : $VARIABLE ( len -->)
 12  <BUILDS DUP C, 0 C, ALLOT
 13  DOES> 2+ ;
 14 : (") ( move inline str to $STK)
 15  R 1+ $@  R C@ R> + 1+ >R ;  -->



SCR # 16
  0 : " ( if comp, put inline str on
  1  $STK in exec, else put now)
  2  STATE @ IF COMPILE (") 22 WORD
  3   HERE C@ 1+ ALLOT
  4  ELSE  22 WORD  HERE 1+ $@
  5  ENDIF ; IMMEDIATE
  6 1 $VARIABLE $EOL ( EOL string)
  7  $EOL 9B $VARFILL
  8 : $FILE ( --> addr, $ -$> $+EOL)
  9  $EOL $@ $+  $P@ 1+ ;
 10 : $SETDR0 ( --> , $ -$> )
 11  FLUSH CLOSE-DR0 $FILE DROP
 12  DR0NAME 10 $STORE DROP ;
 13 : $SETDR1 ( --> , $ -$> )
```

```
 14  FLUSH CLOSE-DR1 $FILE DROP
 15  DR1NAME 10 $STORE DROP ;    -->
```

```
SCR # 17
  0 0 VARIABLE $TR ( temp result)
  1 : ($COMPARE) ( addr1 addr2 len
  2  --> 1,0,-1 )
  3  0 $TR !  -DUP IF ( len>0?)
  4   0 DO  OVER I + C@ OVER I + C@
  5    - -DUP IF ( no match?)
  6     0< IF -1 ELSE 1 ENDIF $TR !
  7  LEAVE ENDIF LOOP ENDIF
  8  DROP DROP $TR @ ;
  9 : ($BLCOMPARE) ( addr len -->
 10  1,0,-1  compare to blanks)
 11  0 $TR !  -DUP IF ( len>0?)
 12   0 DO  DUP I + C@ BL - -DUP IF
 13    0< IF -1 ELSE 1 ENDIF $TR !
 14  LEAVE ENDIF LOOP ENDIF
 15  DROP $TR @ ;                -->
```

```
SCR # 18
  0 : $COMPARE ( --> 1,0,-1
  1  $1 $2 -$>  , compare two str)
  2  $P2@ 1+ $P@ 1+ $LEN $LEN2 MIN
  3  ($COMPARE) DUP 0= ( match?)
  4  IF ( ck rest of str)
  5   $LEN $LEN2 < IF ( $1 longer?)
  6    DROP $P2@ 1+ $LEN +
  7    $LEN2 $LEN - ($BLCOMPARE)
  8   ELSE DROP $P@ 1+ $LEN2 +  $LEN
  9    $LEN2 - ($BLCOMPARE) MINUS
 10  ENDIF ENDIF $DROP $DROP ;
 11 : $= ( --> tf, $1 $2 -$> )
 12  $COMPARE 0= ;
 13 : $< ( --> tf, $1 $2 -$> )
 14  $COMPARE 0< ;
 15 DECIMAL                      -->
```

```
SCR # 19
  0 : "" ( push empty str on $STK)
  1  -1 $P +!  0 $P@ C! ;
  2 0 VARIABLE $TO ( offset)
  3 0 VARIABLE $TL ( length)
  4 : $EXTRACT ( varaddr off char
  5   --> t offset      -$> word
  6   --> f  [when no word left] )
```

```
  7  $TR ! ( store char)
  8  -1 $TO ! 0 $TL ! ( clr off,len)
  9  OVER $VARLEN OVER OVER <
 10  IF ( still room left in var)
 11   SWAP DO ( find 1st non-separ)
 12    DUP I + C@ $TR @ = 0=
 13    IF ( non-separator char)
 14     I $TO ! ( starting offset)
 15                              -->
```

```
SCR # 20
  0     I 1+ OVER $VARLEN OVER OVER
  1     < IF ( not at end of var)
  2      SWAP DO ( find next separ)
  3       DUP I + C@ $TR @ =
  4       IF ( found end of word)
  5        I $TO @ - $TL ! ( length)
  6       LEAVE ENDIF
  7      LOOP  $TL @ 0=
  8      IF ( assume end of line)
  9       DUP $VARLEN $TO @ -
 10       $TL ! ( length to end)
 11      ENDIF
 12     ELSE ( 1 char at end of var)
 13      1 $TL !  DROP DROP
 14     ENDIF  LEAVE
 15                              -->
```

```
SCR # 21
  0    ENDIF
  1   LOOP
  2  ELSE  DROP DROP
  3  ENDIF
  4  $TO @ 0< IF ( word not found)
  5   DROP 0 ( return FALSE)
  6  ELSE ( word was found)
  7   $TO @ + $TL @ $FETCH ( get wd)
  8   $TO @ $TL @ + ( new offset)
  9   1 ( return TRUE)
 10  ENDIF ;
 11
 12
 13
 14
 15                              -->
```

```
SCR # 22
```

```
  0 : $+! ( var --> , $ -$>, add str
  1  to variable ) ( comp. new len)
  2  >R R $VARLEN R + ( start addr)
  3  R $VARMAX R $VARLEN - ( ch lft)
  4  $STORE ( concat, ret len added)
  5  R $VARLEN + R> 1 - C!
  6  ( store new length) ;
  7 : $RESET ( auto-$STK reset)
  8  NEW-ABORT $P! DECIMAL ;
  9
 10
 11
 12
 13
 14
 15                              -->
```

```
SCR # 23
  0 : $LOAD ( filename -$> )
  1  $SETDR1 DR1 1 LOAD ;
  2 : LOAD-ED
  3  " D:EDITOR.4TH" $LOAD ;
  4 : LOAD-BED ( boot editor)
  5  " D:BOOTEDIT.4TH" $LOAD ;
  6 : LOAD-DOS
  7  " D:DOS.4TH" $LOAD ;
  8 : LOAD-TURN
  9  " D:TURNKEY.4TH" $LOAD ;
 10
 11
 12
 13
 14
 15
```

```
SCR # 24
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
```

```
OK
PR-OFF
```