R   (-- n)   (n --R-- n)

Copies the top of the return stack onto the top of the data stack. As this name has been superseded by R@, we do not recommend R.

Standards: FIG , R@ in '79 and '83.

`Related Words: >R  R>  R@`

R#   (-- addr)   "r sharp"

R# is a USER-variable that is only applicable in the BLOCK environment.

After the source file SCRED has been loaded, it is used to store the current cursor position (relative to the start of the BLOCK) in the supplied, FIG-like line editor.

```
     R# @ C/L /  ( leaves line# cursor is on )
```

`Related Words: LOAD BLOCK Used by many words in line editor and SCRED.`

R>   (-- n)   (n --R--)   "r from"

Takes a value N off of the return stack and pushes it onto the data stack. WARNING Only remove thing from the return stack that you put there with >R . Otherwise the subroutine return mechanism will fail. The return stack is a handy place to temporarily store values.

```
     : UNDER1+  ( a b -- a+1 b )
         >r  ( save B ) 1+ r> ;
```

`Related Words: >R R@ RDROP`

R0   (-- addr)   "r zero"

USER variable containing the initial value for the return stack pointer. This is used internally by JForth System Manager

`Related Words: RP!`

R@   (-- n)   (n --R-- n)   "r fetch"

Copy top of return stack onto top of stack .

Standards: '83 and '79 . FIG uses R

```
     : EXAMPLE ( -- )  5 >R R@ ( will be 5 ) RDROP ;
```

`Related Words: >R  R>  R`

RANDOM (-- rnd)

Generate a 16 bit pseudo-random number using the linear congruential method. This is in JU:RANDOM. The sequence is based on the variable RAND-SEED.

`Related Words: CHOOSE`

RANGEOF (value low high -- value |)

Executes the following code up to an ENDOF, and drops the value, if the value falls within the given range. The range includes LOW and HIGH. See CASE.

RAWEXPECTECHO   (-- var-addr)

See the chapter on CLONE.

RDROP   (--) (n --R--)   "r drop"

Drop top item N off the return stack. Only drop things you put there or the subroutine mechanism will fail.

```
     : EXAMPLE ( -- )  5 >R RDROP ;  ( won't crash )
```

`Related Words: DROP XRDROP >R R> R@`

READLINE   "read line"

(file-pointer var-addr addr maxlen -- addr #read | addr -1 if EOF)

```
     file-pointer  specifies the file to read.
     var-addr  holds the address of the virtual-buffer.
       addr = the place in memory to place the data
     maxlen = maximum # characters to read
```

READLINE is used to read the next line of a file into memory, stopping at the EOL character, or if the memory limit is reached. The memory is read through a 1K sequential virtual-buffer that has been allocated via OPENFV . See OPENFV .

The memory address is passed through. In addition, if not at end-of-file, the number of characters that were actually read is returned. Otherwise a -1 is returned. Note that an empty line will return 0, a valid line-length.

Virtual memory areas allocated for reading should be closed via CLOSEFVREAD .

See chapter on File I/O for more information.

```
     : NEXTLINE   ( -- flag , read and display next line )
         MY-FILE @  MY-BUFFER  PAD  1000  READ-LINE
         ( -- addr n1 )   dup 0>
         IF TYPE TRUE
         ELSE 2DROP FALSE
         THEN ;
```

`Related Words: OPENFV  LINESFILLV  CLOSEFVREAD FREAD`

RECURSE (--)

Calls word currently being defined. Recursion is powerful tool but must be used with caution. The rules for recursion are:

1) Do NOT use global variables because they will be changed by recursive calls. Use Local Variables instead.

2) Provide a way to stop recursion. Excessively deep recursion will overflow the return stack.

Here is an example of using recursion to calculate N factorial. N Factorial is defined as:

```
     N*(N-1)*(N-2)....*1
```

We can define it recursively as:

```
     FACT(N) = N * FACT(N-1)
     FACT(1) = 1
```

Here is the Forth code. If this is the first time you have used recursion, it will seem a little mind bending. LISP programmers do this stuff all the time.

```
     : FACT  ( N -- N-factorial )
        dup 1 >
        IF
           dup 1- recurse *
        THEN
     ;
     4 FACT . \ should print 24
```

RECURSIVE (--)

This is not in the dictionary because it is so easy to define. When a word is being defined, it is SMUDGED so that if the compilation fails it can't be called. By calling UNSMUDGE, we can allow a word to call itself. Compare this example to the example under RECURSE.

```
: RECURSIVE  ( -- ) unsmudge ; immediate

: FACT  ( N -- N-factorial )
   RECURSIVE  \ make this word recursive
   dup 1 >
   IF
      dup 1- FACT *  \ calls itself!
   THEN
;
```

REDEF?   (-- addr)

This user-variable is scanned by (CREATE) . If a Forth word is redefined a message will be printed if this variable if TRUE.

```
REDEF? OFF
: FOO ;
: FOO ;   ( NO message!)
```

REPEAT   (--)

Used to terminate a BEGIN ... WHILE ... REPEAT conditional construct. Unconditionally branches to code following BEGIN. See WHILE for example.

For advanced users: REPEAT is actually an IMMEDIATE word that has the following stack diagram.

(addr-begin begin-flag waddr while-flag --)

```
addr-begin = address following BEGIN to branch to
begin-flag = shows BEGIN started the loop
     waddr = address of 0BRANCH offset created by WHILE
while-flag = pairs checking signature for WHILE
```

`Related Words: BEGIN WHILE UNTIL IF ELSE THEN DO LOOP`

RETURN   (--)

RETURN causes an immediate exit from the the current colon definition, even from within nested DO LOOP's . Like the other control structures, IF ELSE LOOP etc... , It has an immediate part and a run time part. RETURN figures out the LOOP nest level at compile time, then compiles the appropriate number of return stack items to drop. If used outside any LOOP structures, RETURN compiles EXIT . The run time action is to exit the function when executed.

```
Run time:
  Stack: ( -- )
  Return stack: ( return-address loop-parameters... --R--)
Compile time
  Stack: ( -- )
  Return stack: ( --R-- )
```

```
: EXAMPLE   ( -- )  10 0
    DO  100 0
        DO  I . ." about to return, bye" RETURN
        ." won't get here"
        LOOP ." won't ever print this"
    LOOP  ." Also won't print this" ;
    ( different from LEAVE)
```

`Related Words: ?RETURN EXIT ?EXIT LEAVE`

REVERTVOC   (-- addr)

REVERTVOC is a USER-variable that is examined by : "colon". If found true, it will, at the very beginning of the definition, change the vocabulary that will be searched to the CURRENT vocabulary. The default state of REVERTVOC is FALSE. REVERTVOC has been provided for compatibility with older vocabulary systems.

`Related Words: : VOCABULARY DEFINITIONS ORDER`

ROOT   (--)

ROOT is the root vocabulary of JForth. It will always be searched, even if the word ONLY was used. See the section on Vocabulary for more information.

Standard: '83 experimental proposal.

`Related Words: VOCABULARY ORDER VOCS`

ROT   (a b c -- b c a)

Rotates the third item to the top of the stack.

```
    11 22 33 ROT . ( prints 11 )
```

`Related Words: SWAP DUP OVER PICK >R R>`

RP! r p store

RP! has two different meanings, based on the standard you are using.

Standards: FIG and 79 (JForth default) ...

(--)

Initializes the return stack pointer to the address contained in R0 . Used in COLD and QUIT .

'83 (available from JU:MULTISTANDARD file)

(addr --)

Sets the return stack pointer to the addr on the data stack.

`Related Words: R0 COLD QUIT`

RP@   (-- addr)   "r p fetch"

RP@ pushes the return stack pointer address on top of the stack .

RPICK   (n -- rval-n)

This is the return stack equivalent of PICK. RPICK uses the value N on the stack to copy the contents of cell number N out from the return stack and push the value on top of the stack . This is the only way to access items deep down on the return stack . This is zero based so a 0 RPICK is equivalent to an R@ . RPICK is very fast and efficient, compared to using multiple calls to R> .

```
: EXAMPLE ( -- )
    5 >R 6 >R 7 >R 2 RPICK . ( 5 ) 3 XRDROP ;
```

S->D   (n -- d)   "s to d" "single to double"

S->D sign extends a 32 bit value N into a 64 bit double number D . This means the MSB of N is duplicated through the upper half of D .

```
-23 S->D D.  ( prints -23)
```

`Related Words: W->S  B->S`

S0   (-- addr)   "s zero"

S0 is a user variable containing the initial value of the stack pointer.

`Related Words: SP@ 0SP DEPTH R0`

SAVE-FORTH   (<filename> --)

Save the current JForth dictionary in a file that can be executed later.

This yields two practical uses in the JForth development cycle:

Useful utilities and in general, any program may be saved in compiled form, and therefore almost instantly brought up . This will prove valuable, as you can save a compiled image with just the routines you've tailored for developing and debugging your application.

SAVE-FORTH is a vital part of the mechanism for expanding dictionary size. As your program grows in size, you may receive the "insufficient dictionary available" message. The following procedure lists the steps for increasing your JForth image size:

1) Set the #K variable...it tells SAVE-FORTH how large an image to save. In our example, we simply increment #K by 20 with +! ...

```
20 #K +!
```

Our new dictionary will be   20 X 1K (1K=1024) or 20,480 bytes larger. Note: you can also place the absolute value for size in #K ... if it contained 100 , we could type:

```
120 #K !
```

This would have the same result.

2) Save the image to the disk, using your own filename:

```
SAVE-FORTH MYFORTH
```

NOTE: if #K holds 120, you will need slightly more than 120K of disk area (about 5% more).

3) Exit JForth, returning to the CLI with:

```
BYE
```

4) From the CLI, run the image you just created with:

```
RUN MYFORTH
```

JForth will once again appear, (with the same words.) The only difference will be the larger dictionary area. Of course, you are using that much more Amiga memory now. Note: if AmigaDOS says that you don't have enough memory to run your new image, try rebooting the system...this recovers any fragmented memory.

Trivia: By setting #K to less than the current HERE (0 is a convenient number), SAVE-FORTH will create a minimum-sized image, occupying as little disk space as possible. This image however, will have an available dictionary size less than 2K when later booted. This could be useful for saving complete applications that you plan to use without adding much to their dictionary.

Standards: JForth unique

```
Related Words: #K #U MAP
```

SAVE-IMAGE (<wordname> <filename> [options] --)

Save a cloned program to a file. See the chapter on CLONE.

SCAN (addr count char -- addr' count')

```
addr = an address
count = length to search
char = an ASCII character
addr' = address where char was found OR end+1
count' = string length after matching char, 0 if not found
```

SCAN the string at ADDR (that is COUNT bytes long) for CHAR.

If found, return the address that matched and the REMAINING LENGTH of the string including the CHAR. If not found, return the end-of-string+1 for ADDR' and a COUNT' of 0 .

Standards: F83

```
" A text string" COUNT ASCII r  SCAN  TYPE
( The above would print "ring" )
```

```
Related Words: SKIP PARSE PARSE-WORD COMPARE QUERY
```

SCAN-ALL-VOCS (--)

SCAN-ALL-VOCS is a JForth Unique generalized function for accessing all the words in JForth. Two deferred words are used. WHEN-VOC-SCANNED is executed when each vocabulary is scanned, WHEN-SCANNED is executed when each word is scanned. By setting these words you can write systems that will process all of the words in the dictionary. Please see the chapter on Vocabularies for more info.

```
Related Words: WHEN-SCANNED WHEN-VOC-SCANNED SCAN-VOC SCAN-WORDS
```

SCAN-WORDS (--)

SCAN-WORDS is a JForth Unique generalized function for accessing the words in the CONTEXT vocabulary. The deferred word WHEN-SCANNED is executed when each word is scanned.

```
Related Words: SCAN-VOC SCAN-ALL-VOCS WORDS CONTEXT
```

SCAN-VOC (--)

SCAN-VOC is a JForth Unique word used by SCAN-ALL-VOCS and SCAN-WORDS to search a vocabulary. The deferred word WHEN-SCANNED is executed when each word is scanned.

```
Related Words: WHEN-SCANNED WHEN-VOC-SCANNED SCAN-VOC SCAN-WORDS
```

SEALMODULE (-- , <modulename>)

See the chapter on MODULES.

SET-BIT (n bit# -- n')

Set a specific bit in N. Bit 0 is the LSB.

```
0 4 SET-BIT . ( 16 )
```

```
Related Words: CLR-BIT @BITS !BITS BIT-SET? BIT-CLR? OR AND XOR
```

SHIFT (n shift-count -- n')

SHIFT n the number of bits indicated by shift-count, shifting in a zero as the new bit. If shift-count is positive, the shift is to the left, otherwise it is to the right.

Useful, for example, when extracting or inserting bit fields.

Standards: 83 suggested

```
HEX 75  DUP  0F AND .  ( print 5)
-4 SHIFT 0F AND .      ( print 7)
```

`Related Words: ASHIFT 2* 2/`

SHOWME (-- , <wordname>)

See the chapter on CLONE.

SIGN (sign dbl -- dbl)

If 'sign' is negative , add a minus sign to the output string. The double number 'dbl' is not affected. Used in numeric conversion.

Standard: FIG. In '83 (n --).

```
: N>TEXT  ( N -- addr count , signed number to text )
    S->D   SWAP   OVER  DABS
    <#  #S  SIGN  #> ;
-234 N>TEXT TYPE
```

`Related Words: HOLD # #S`

SIZEMEM (memory-block -- allocated-size)

Given the address of a memory-block that had been acquired via ALLOCBLOCK, return the size of the allocated block. See chapter on Memory Management.

```
MY-MEM @ SIZEMEM  ( -- size )
```

`Related Words: ALLOCBLOCK  FREEBYTE  FREEBYTEA  FREEBLOCK`

SKIP (addr count char -- addr' count')

```
addr = an address
count = length to search
char = an ASCII character
addr' = address of 1st char that didn't match OR end+1
count' = string length after non-matching char,
       0 if entire string matches
```

Remove leading characters from a string.

Scan through the string at ADDR (that is COUNT bytes long) for CHAR, SKIPping leading occurrences until a non-matching character is found, or the end of the string. If a non-matching character is found, return its address and the REMAINING LENGTH of the string. If the entire string matches, return the end-of-string+1 for addr' and a count' of 0 .

Standards: F83

```
"     Finally a word!" COUNT BL SKIP  TYPE
( Don't type out leading spaces!)
```

`Related Words: SCAN PARSE PARSE-WORD COMPARE FIND`

SKIP-WORD

SKIP-WORD causes the next pass through WORD or FILEWORD to ignore the input stream, accepting instead the existing text at HERE.

```
: $CREATE ( $name -- , create a word whose name is on the
stack )
   HERE $MOVE
   SKIP-WORD CREATE
;
" MYDATA" $CREATE 234 ,
MYDATA @ .
```

`Related Words: WORD  LWORD  FILEWORD UNWORD $FOPEN`

SLOW   (--)

SLOW reverts the EMIT I/O to a conventional single-character scheme. JForth default I/O technique is line-buffered, see FAST. In SLOW mode, emit will output each character as soon as it it is called. This might be handy in debugging, continuously updated displays, etc.

You may want to use FLUSHEMIT to force output in FAST mode instead of using SLOW.

Standard: JForth unique .

`Related Words: (EMIT) (KEY)  FAST EMIT TYPE`

SMUDGE   (--)

Mark the LATEST definition so that FIND will be unable to locate it. SMUDGE means to hide or conceal the latest definition . SMUDGE is used in : and CODE and some other defining words to make sure an incomplete definition is not accidentally compiled or executed.

Hashing may cause unpredictable results with SMUDGE , see HASH.OFF.

Standards: FIG and '79 use SMUDGE but they TOGGLE the smudge state, JForth sets it. '83 systems use HIDE .

```
VARIABLE BAD-MAKE
: RISKY-WORD  ( -- flag , true if fails )
    bad-make @ ;
: MAKEWORD
    CREATE  SMUDGE ( Make unFINDable )
        risky-word abort" Couldn't MAKEWORD!"
        UNSMUDGE   ( Make FINDable if survived RISKY-WORD)
    DOES> .
;
MAKEWORD OKWORD
OKWORD    ( Will be found!)
TRUE BAD-MAKE !  MAKEWORD BADWORD   ( Creation fails)
BADWORD   ( Won't be found!)
```

`Related Words: UNSMUDGE REVEAL FIND CODE :`

SOURCE   (-- addr count)

This is a DEFERred word; by default it executes (SOURCE).

SOURCE is used by PARSE and PARSE-WORD to return the address and length of the available input stream. (SOURCE) normally returns the TIB start and the contents of #TIB .

Related Words: PARSE PARSE-WORD IS WHAT'S TIB #TIB QUERY WORD

SP!   "s p store"

SP! has two different meanings, based on the standard you are using.

FIG and '79 (JForth default) ...

(--)

Initializes the data stack pointer to the address contained in S0 . Used in COLD and ERROR .

'83 (available from JU:MULTISTANDARDS file)

(addr --)

Sets the data stack pointer to the addr on the data stack. Use 0SP in place of the old style SP! .

Related Words: 0SP SP@ S0 COLD

SP@   (-- addr)   "s p fetch"

In a traditional Forth this is the address of TOS , the Top Of Stack. In JForth, however, TOS is cached in register D7 inside the 68000. It has no "address". SP@ , then, actually returns the address of the SECOND item on the data stack in JForth.

Try to avoid using SP@ , use PICK and DEPTH if you can.

Related Words: RP@ DEPTH S0

SPACE   (--)

Outputs an ASCII space, 20 hex, to the current EMIT device.

Related Words: SPACES EMIT BL ASCII

SPACES   (n --)

Prints N spaces on the standard EMIT device . Ascii space = 20 hex .

Related Words: SPACE EMIT TYPE

SPAN   (-- addr)

SPAN is a user-variable that is set to the number of characters last read in by EXPECT.

```
: GET$ ( -- $string , input string )
    PAD 1+ 120 EXPECT ( place characters at PAD+1)
    SPAN @ PAD C!     ( update byte count at PAD)
    PAD ;
```

Related Words: EXPECT QUERY

SPARE   (-- addr)

Literally a spare USER variable for temporary use by any program .

```
FOPEN RAM:TEMP    SPARE !   ( handy place)
```

Related Words: USER PAD >R

SPEAK   ($string --)

Translate string to phonemes and speak them. This uses the Amiga Narrator device. Make sure you have the volume up. Check out JD:DEMO_SPEAK for examples.

```
INCLUDE? SPEAK JU:SPEAK
```

```
SPEAK.INIT  ( -- , allocate I/O request block )
" Greeting Earthlings!" SPEAK  ( etc. )
" We come in pea  AAayaueahh!" SPEAK
SPEAK.TERM  ( deallocate when through
```

SQRT   (N -- N1/2)   "square root"**

Take the integer square root of N. Truncates answer. You can scale N by 10,000 to get two "decimal places" of accuracy.

```
INCLUDE? SQRT JU:SQRT
49 SQRT .   ( prints 7 )
```

STACKSIZE   (-- var-addr)

See the chapter on CLONE.

STATE   (-- addr)

A user variable used by INTERPRET, reflecting the state of the interpreter.

The JForth interpreter may be in one of two states;

1) If STATE = zero, the interpreter is in INTERPRET Mode, executing words as they are parsed from the input stream.

2) If STATE = non-zero, the interpreter is in COMPILE Mode, compiling words as they are parsed from the input stream.

Standards: '79, '83, In FIG 0 means system is compiling .

```
: NAMEOF ( <word> -- nfa )
    BL WORD FIND   ( search dictionary )
    IF  >NAME STATE @  ( compiling mode? )
        IF [COMPILE] LITERAL  ( save in dictionary )
        THEN
    ELSE COUNT TYPE ."  not found"
    THEN
; IMMEDIATE
NAMEOF SWAP ID.
: FOO NAMEOF SWAP ID. ; ( compile SWAP's NFA as LITERAL )
FOO       ( print "SWAP" )
```

Related Words: INTERPRETING? COMPILING? [ ]

STATS   (--)

See the chapter on CLONE.

SWAP   (a b -- b a)

Exchange top two stack items.

```
23 56 .S
SWAP .S
SWAP .S
```

TASK   (--)

TASK is an empty definition, below which resides the JForth Kernal (that part of JForth which can not be VIEWed or re-generated by the programmer).

The source code for the JForth system is provided for all words above TASK, and the system can be regenerated from this point from the JF: directory on the EXTRAS disk.

TEMPBUFF (-- addr)

TEMPBUFF is a user-variable, provided as a convenience for the programmer.

The programmer may allocate a virtual-buffer via OPENFV, and place the resultant address in TEMPBUFF .

Subsequently, the programmer may access file-virtual words that directly access TEMPBUFF, providing simple stack diagrams (they don't require the buffer or a variable address parameter). The only such word provided is TEMPF, ; the programmer may easily add more.

Note: TEMPBUFF usage is non-reentrant; you must save its' contents if you call another word that will use it.

Standard: JForth unique

`Related Words: TEMPFILE  TEMPF, OPENFV`

TEMPF, (n --) "temp f comma"

This word writes the 32 bit value N to the file whose file-pointer is contained in TEMPFILE.

The write operation will be done through a virtual-buffer, opened with OPENFV, whose address is contained in TEMPBUFF.

NOTE: TEMPF, usage is non-reentrant. You must save the contents of TEMPFILE and TEMPBUFF if you call another word that will also need it.

Standard: JForth unique

`Related Words: TEMPBUFF  TEMPFILE  OPENFV  CLOSEFVWRITE`

TEMPFILE (-- addr)

TEMPFILE is a user-variable, provided as a convenience for the programmer.

The programmer may open a file via FOPEN or (FOPEN), and place the resultant file-pointer in TEMPFILE.

Subsequently, the programmer may access file-operation words that directly access TEMPFILE, providing simple stack diagrams (they don't require the file-pointer parameter). The only such word provided is TEMPF, ; the programmer may easily add more.

NOTE: TEMPFILE usage is non-reentrant. You must save its contents if you call another word that will use it.

Standard: JForth unique

`Related Words: TEMPBUFF  TEMPF,`

TEXT=? (addr1 count addr2 -- flag)

Compare two strings, each count bytes long, return a TRUE if they match, FALSE otherwise.

TEXT=? will perform a case-sensitive compare if the value of MCASE-SENSITIVE is non-zero; otherwise the compare will be performed ignoring ASCII case (JForth default condition).

```
: FROG=?  ( $word --flag )
   count 4 =
   IF " frog" count swap text=?
   ELSE drop false
   THEN
```

```
     ;
     " bird" frog=? .
     " Frog" frog=? .
```

Standard: JForth unique

Related Words: MATCH? MCASE-SENSITIVE $= COMPARE PARSE SKIP SCAN

THEN (--)

Mark the end of an IF...THEN or an IF...ELSE...THEN conditional construct. See tutorial for more examples.

```
     : EXAMPLE  ( N -- )
         5 =
         IF  ." It's a five! "
         ELSE ." Why didn't you type in five? "
         THEN
         cr ." Thank you." ( executed in any case ) ;
```

Related Words: IF ELSE BEGIN UNTIL WHILE REPEAT

TIB   (-- addr)   "terminal input buffer"

TIB returns the memory area used for receiving and storing the input stream that will be INTERPRETed.

```
     TIB 100 TYPE  ( type this line plus previous stuff)
```

Related Words: #TIB TIB0 SOURCE

TIMER?

```
     TIMER_LIB
     TIMER_NAME
```

Internal operators to manage the TIMER library. See :LIBRARY.

TIMES   (n --)

Repeat the current command line N times. TIMES should be placed at the the end of the command line. TIMES can only be used from the keyboard.

If the command-line is to repetitively operate on a number or set of numbers, they must exist on the stack prior to the beginning of the command-line.

```
     ." Hello! " CR 3 times
     ( This prints Hello! 3 times.)
```

Related Words: >IN DO LOOP

TOGGLE   (addr mask --)

Does an 8-bit EXCLUSIVE OR of byte at addr with the given mask. Result is left in location addr.

Standards: FIG

```
     " fred" COUNT OVER $ 20 TOGGLE TYPE
     ( change case of 'f' , print  Fred )
```

Related Words: XOR OR

TRACKING   (-- var-addr)

See the chapter on CLONE.

TRANSLATOR?

```
TRANSLATOR_LIB
TRANSLATOR_NAME
```

Internal operators to manage the TRANSLATOR library. See :LIBRARY.

Standard: JForth internal .

TRUE (-- true-flag)

A CONSTANT, equal to a boolean TRUE, or -1.

Standards: '83 (79 & FIG use a value of 1)

`Related Words: FALSE`

TUCK (a b -- b a b)

Duplicate and insert the top item on the stack below the second item. This is a fast coded primitive, the high-level logical equivalent is:

```
11 22 SWAP OVER .S

: NEW-DUMP  ( addr cnt -- )
    TUCK ( save count )
    DUMP  .  ." bytes dumped" ;
```

`Related Words: OVER SWAP DDUP`

TYPE (addr count --)

TYPE outputs a character string to the EMIT device. The string starts at address ADDR and is COUNT characters long . TYPE will type a maximum number of characters determined by the variable MAX-TYPE to prevent runaway output.

(If you accidentally TYPE something that messes up the screen font, enter a control 'O' to fix it.)

```
" Hello" COUNT TYPE  ( print Hello )
```

`Related Words: ID. EMIT COUNT  $TYPE ."`

TYPEFILE (<filename> --)

Output a file to the current output device. It is similar to the AmigaDOS CLI 'TYPE' command in that it can be paused and restarted from the keyboard. See ?PAUSE.

```
TYPEFILE S:STARTUP-SEQUENCE  ( see file )
```

`Related Words: ?PAUSE FOPEN DOLINES`