: (<name> --) "colon "

tart the definition of a new Forth word. Enters COMPILE mode by setting STATE to TRUE. The code that follows will be compiled into the word. Sets CONTEXT and CURRENT vocabularies to the same. Terminate with ; which sets SMUDGE bit .

```
\ Define a new Forth function then test it.
: DOUBLE ( n -- n*2 )
    DUP + ;
7 DOUBLE .  ( prints 14 )
```

Related Words: ; CREATE

:CREATE (<name> --) "colon create"

:CREATE acts to create a dictionary header, using the next word available in the input stream. It is used by: CREATE VARIABLE CONSTANT and other defining words. :CREATE is a DEFERred word. See (CREATE).

Related Words: (CREATE)

:LIBRARY (<name> --) "colon library"

Define a new Amiga Library to be called using CALL . Any Library that has an FD file can be called from JForth. The standard libraries have already been defined for JForth.

```
:LIBRARY STUFF  ( define new library )
STUFF?  ( open library )
```

See the chapter on "Calling Amiga Library Routines".

:STRUCT (<name> --)

Defines a new structure. See the special section on interfacing with the Amiga.

File: JU:C_STRUCT.

Related Words: ;STRUCT .. ..@ ..!

; (--) "semicolon"

Terminate a colon definition. Compiles EXIT into the definition being created. EXIT is the execution time procedure of ; Clears the smudge bit. Set STATE back to zero. Set size field to reflect size of word. Places the system back into interpret mode. ; is immediate.

```
: HI ." Hello!" CR ;
```

Related Words: EXIT : BOTH INLINE :CREATE

;STRUCT (--) "semicolon struct"

Terminate a 'C' like structure definition. Save a pointer to the last member defined for DST to use. See chapter on "Accessing Amiga 'C' Structures".

File: JU:C_STRUCT

< (n1 n2 -- flag) "less than"

Compare n2 with n1. Leaves TRUE if n1<n2 , FALSE if not.

```
7 23 <  ( leaves TRUE )
```

Related Words: > = U< <=

<# (d --) "less sharp"

<# starts numeric output string conversion. Initializes the pointer in HLD to point to the beginning of PAD. Terminate with a #> . See # .

```
S->D <# # # # # #S #>  ( gives at least 5 digits)
```

Related Words: #S # #> SIGN HOLD HLD

<= (n1 n2 -- flag) "less than or equal"

Compare n2 with n1. Leaves TRUE if n1 <= n2 , FALSE if not.

```
7 23 <=  ( leaves TRUE )
23 23 <= ( leaves TRUE )
```

Related Words: > = U< < >=

<FASTEMIT> (char --) "bracket fast emit"

Transmits one character to the output device using buffered I/O. This is the default contents of the DEFERred word EMIT.

<FASTEMIT> is installed into EMIT when FAST I/O mode is entered. To speed up printing, the character will be held in a buffer until:

```
the FAST I/O buffer is filled,
a CR is executed,
or KEY , EXPECT or FLUSHEMIT is executed.
```

This is faster than outputting characters one at a time. <FASTEMIT> calls +OUT to adjust OUT to reflect the current cursor position. This is done on each character.

Related Words: EMIT FAST SLOW FLUSHEMIT

<FLUSHEMIT> (--) "bracket flush emit"

This is the default contents of the DEFERred word, FLUSHEMIT.

When executed in FAST mode, <FLUSHEMIT> sends any characters held in the FAST buffer to be sent to the standard EMIT device.

= (n1 n2 -- flag) "equals"

Compares top two values n1 and n2 on stack and replaces them with flag = true if they are equal and flag = false if they are not equal.

Related Words: < > 0= -

> (n1 n2 -- flag) "greater than"

Compare top two values on stack. Leave TRUE if n1 greater than n2, otherwise leave FALSE.

```
23 7 > ( leave TRUE )
```

Related Words: < = U> IF >=

>= (n1 n2 -- flag) "greater than or equal"

Compare top two values on stack. Leave TRUE if n1 >= n2, otherwise leave FALSE.

```
23 7 >= ( leave TRUE )
23 23 >= ( leaves TRUE )
```

Related Words: < = U> IF >

>ABS (rel-addr -- abs-addr) "to absolute"

Convert JForth relative address to 68000 absolute address.

JForth "addresses" are actually an offset from the base of the dictionary. This allows us to use addresses that stay the same every time we run JForth, even though JForth may load into different places in memory each time. This makes JForth code more relocatable. The Amiga operating system, however, uses normal 68000 addresses. You must convert JForth relative addresses to 68000 absolute addresses before passing them to the Amiga.

```
DATE-VARIABLES @ >ABS ( Amiga addr )
CALL DOS_LIB DATESTAMP
```

Related Words: >REL S@

>BODY (cfa -- body) "to body"

Convert the CFA or tick address (start of the executable code) of a CREATE or CREATE/DOES> child to its Body address (location of the related data). See BODY>.

>BODY will only return a meaningful result if the passed-in address points to the CFA of a word created via CREATE, either directly or via a CREATE/DOES> defining word. This does NOT include the CFA for other data-structures, such as VARIABLEs, CONSTANTs, or VALUEs.

[Note: In JForth Version 1.2 and earlier, >**BODY** was a noop, implying the **body** and the **cfa** were the same!]

Related Words: ' BODY> >NAME NAME> DO-DOES-SIZE

>DOS (addr count --) "to dos"

Convert the string residing at ADDR that is COUNT bytes long to a NUL-TERMINATED string in the DOS0 buffer. This can then be passed to a DOS Library routine.

```
( moves a filename to the DOS0 buffer )
" df0:c/dir" COUNT >DOS
DOS0 0COUNT TYPE
```

Related Words: DOS0 +DOS $>0

>IN (-- addr) "to in"

This is the user-variable that stores the index that INTERPRET uses to track its current place in the TIB.

>IN is set to 0 by QUERY, and incremented as INTERPRET processes the input stream.

```
: TYPE&EXEC  ( <name> -- , type name then execute. )
    >IN @  ( save location )
    CR 32 WORD COUNT TYPE CR
    >IN !  ( restore location for interpreter to execute )
;
: FOO ." Hello" CR ;
TYPE&EXEC FOO
```

Related Words: TIB EXPECT BLK PULLTIB PUSHTIB

>INLINE to inline

```
( relative-altered-return-address -- )
( --R-- absolute-return-address )
```

>INLINE takes a relative address and converts it to an absolute address before putting it back on the return stack.

>INLINE is part of a set of 4 words that allow for transportable inline data words. >INLINE puts an altered inline data address back to the top of the return stack, in the proper return stack format. In JForth for the Amiga, the return stack contains absolute addresses, but @ and other data movement words are relative. See chapter on transportability.

Related Words: INLINE> INLINE@ INLINE+

>LINK (cfa -- link) "to link"

>LINK converts a CFA to a LINK address.

Related Words: >NAME N>LINK

>NAME (cfa -- nfa) "to name'

>NAME converts the code-field-address of a dictionary header (that returned by ') to the name-field-address.

```
' SWAP  >NAME ID.
```

Related Words: ' NAME>

>NEWLINE (--) "to new line'

Perform a CR unless the cursor is already at the start of a new line.

Related Words: LINELIMIT OUT CR

>PARENT (child-CFA -- parent-CFA) "to parent"

>PARENT is a JForth unique word to fill the need for a transportable way to convert a child CFA to its parent CFA . See related words.

Related Words: CREATE DOES>

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

>R pops a value from top of stack and pushes the value on top of return stack. You can use the return stack as a place to store temporary variables. You must clean up before the end of a definition, however, or your program will crash.

```
: EXAMPLE ( n m -- n+1 m )
    >R 1+ R> ;    ( faster than SWAP 1+ SWAP )
```

Related Words: R> X>R XR> XRDROP R@ RDROP RPICK

>REL (absolute-addr -- relative-addr) "to rel"

Inverse of >ABS . Converts an absolute 68000 address to a relative JForth address. Addresses returned by Amiga Library routines must be converted to relative before accessing them with JForth.

Related Words: >ABS

>US (n --) (---user-stack--- n) "to user"

Push N onto the User Stack. The User Stack is another stack that can be used as a convenient place to push data. It is used by during compilation by IF DO and other conditional words.

Related Words: R> >R US> US@

? (addr --) "question"

Print the value stored at ADDR . Defined as : ? @ . ;

```
VARIABLE VAR1   789 VAR1 !
```

```
VAR1 ?
```

?ABORT" (flag --) "question abort quote"

?ABORT is a JForth synonym for ABORT" .

?CLOSEBOX (-- flag) "question close box"

Return TRUE if the Close Box gadget has been hit in an open graphics window. The window must have been made current using GR.SET.CURWINDOW. See chapter on Graphics and Event handling, also JD:DEMO_BOXES.

File: JU:AMIGA_EVENTS

?CONTROL-D (-- flag) "question control d"

Return TRUE if control D has been hit on the keyboard. Used in the debugger to interrupt execution and enter debug mode.

File: JU:DEBUGGER

?COMP (--) "question comp"

?COMP issues an error message and QUITs if JForth is not in compile mode, ie. STATE is TRUE. Use to ensure that the system is in compile mode.

```
Related Words: STATE ?EXEC
```

?CSP (--) "question c s p"

?CSP issues an error and QUITs if the current stack pointer position does not equal the value stored in CSP. Uses WARNING and MESSAGE. Used for compiler security because an unbalanced stack means compilation error. JForth uses !CSP to save the stack pointer position upon start of compilation and ?CSP to check it when finished compiling.

```
Related Words: DEPTH SP@
```

?DUP (n -- n n | 0) "question dup"

Duplicates top of stack if non-zero. Same as -DUP .

```
Related Words: DUP 0=
```

?ERROR (flag error-number --) "question error"

?ERROR issues error message specified by N and QUITs if flag = true.

```
Related Words: ERROR MESSAGE WARNING QUIT ABORT (ABORT)
```

?EXEC (--) "question execute"

?EXEC prints error message and QUITs if system is not in execute mode. STATE = zero if system is in execute mode. : uses ?EXEC

```
Related Words: ?COMP STATE
```

?EXIT (flag --) "question exit"

?EXIT will exit a word if flag on stack is true, else it will continue. RETURN is safer, since it will compile ?EXIT if not within a loop, and it will compile <?RETURN> if within any level of nested do loops.

```
Related Words: EXIT RETURN ?RETURN
```

?GOTO.ERROR (flag --)

Jump to ERROR: label if flag is true.

File: JU:GOTO_ERROR

Related Words: GOTO.ERROR

?LEAVE (flag --)

LEAVE a DO LOOP if the flag is true. See LEAVE .

Related Words: LEAVE DO LOOP RETURN

?LETTER (char -- flag)

If the ASCII character on the stack is alphabetic, return a TRUE; otherwise return FALSE.

Related Words: NUMBER? ASCII

?OF (value flag -- value |)

Executes the following code up to an ENDOF, and drops the value if the flag is true. See CASE.

?PAIRS (n1 n2 --) "question pairs"

Tests n2 and n1 for equality. If equal, they are discarded and execution goes on. If not equal, an error message is given and QUIT is executed. It is used for checking whether conditional constructs are paired correctly. Usual message is CONDITIONALS NOT PAIRED.

Related Words: ?DO_FLAG ?IF_FLAG AGAIN THEN ELSE LOOP

?PAUSE (--) "question pause"

Pause if a key has been hit. ?PAUSE calls ?terminal . If a key has been activated, it will stop and wait for a line of text followed by a carriage return. It will execute the statements in that line then continue after where the ?PAUSE was executed. It is used by VLIST DUMP TYPEFILE and some other printing words.

Related Words: ?TERMINAL KEY

?RETURN (flag --) "question return"

?RETURN takes a flag and returns if the flag is non-zero. See RETURN.

Related Words: RETURN EXIT ?EXIT EXIT

?STACK (--) "question stack"

?STACK executes ?ERROR if stack underflow or overflow occurs.

An UNDERFLOW occurs if more items are removed from the stack than are actually on the stack. An OVERFLOW occurs if so many items are added to the stack that it grows down into the PAD memory area.

?STAY (stay-flag --) "question stay"

?STAY is the opposite logic version of ?LEAVE; ?STAY leaves if the flag is false.

?TERMINAL (-- key-hit?) "question terminal"

Check the keyboard to see if a key has been hit. Return TRUE if so. KEY can then be used to read the character. ?TERMINAL is deferred.

```
: YAK BEGIN ." Yak Yak Yak" CR ?TERMINAL UNTIL ;
```

Related Words: ?PAUSE ?CLOSEBOX ?CONTROL-D

?VISIBLE (char -- flag) "question visible"

If the character on the stack is a visible, printable character (would cause the cursor position to move to the right one column) return TRUE; otherwise return FALSE.

```
: SAFE.EMIT ( char -- , emit if visible, '.' if not )
    DUP ?VISIBLE NOT
    IF DROP ASCII . THEN EMIT ;
```

@ (addr -- n) "fetch"

Fetch the 32 bit value N from the address on the stack. This is the primary operator for reading memory. The address is a relative JForth address. The address must be even.

```
VARIABLE VAR1   567 VAR1 !
VAR1 @ .   ( print 567 )
```

Related Words: C@ W@ ! >REL ODD@

@&CLOSEFILES (var-addr --) "fetch and close files"

This word accepts the address of a VARIABLE or USER-variable that contains the address of a memory block containing a list of file pointers.

An application can easily maintain a list of its open files by calling ALLOCBLOCK and using the area as a stack, pushing each file-pointer as it receives them. All of the files may be closed at once by passing the VARIABLE or USER-variable pointing to this area to @&CLOSEFILES.

If the address passed to @&CLOSEFILES contains zero, there will be no effect.

Once @&CLOSEFILES has closed the files, it will free the memory block used to store the list, and store a zero in the pointer addr.

Related Words: @ ALLOCBLOCK FCLOSE

@&FREEBLOCKS (var-addr --) "fetch and free blocks"

This word accepts the address of a VARIABLE or USER-variable that contains the address of a memory block containing a list of allocated memory blocks.

An application can easily maintain a list of its allocated memory blocks by calling ALLOCBLOCK and using the area as a stack, pushing each block pointer as it receives them. All of the blocks may later be freed at once by passing the VARIABLE or USER-variable pointing to this area to @&FREEBLOCKS.

If the address passed to @&FREEBLOCKS contains zero, there will be no effect.

Once @&FREEBLOCKS has freed the areas in the list, it will free the memory block used to store the list, and store a zero in the pointer addr.

Related Words: ALLOCBLOCK FREEBLOCK

@EXECUTE (addr --) "fetch execute"

Fetch a CFA from the given address and execute that word. See EXECUTE.