

NAME

fc – Fortran compiler

SYNOPSIS

fc [**-c**] *sfile1.f* ... *ofile1* ...

DESCRIPTION

Fc is the UNIX Fortran compiler. It accepts three types of arguments:

Arguments whose names end with *‘.f’* are assumed to be Fortran source program files; they are compiled, and the object program is left on the file *‘sfile1.o’* (i.e., the file whose name is that of the source with *‘.o’* substituted for *‘.f’*).

Other arguments (except for **-c**) are assumed to be either loader flags, or object programs, typically produced by an earlier *fc* run, or perhaps libraries of Fortran-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out**.

The **-c** argument suppresses the loading phase, as does any syntax error in any of the routines being compiled.

The following is a list of differences between *fc* and ANSI standard Fortran (also see the BUGS section):

1. Arbitrary combination of types is allowed in expressions. Not all combinations are expected to be supported at runtime. All of the normal conversions involving integer, real, double precision and complex are allowed.
2. Two forms of “implicit” statements are recognized: **implicit integer /i-n/** or **implicit integer (i-n)**.
3. The types **doublecomplex**, **logical*1**, **integer*1**, **integer*2**, **integer*4** (same as **integer**), **real*4** (**real**), and **real*8** (**double precision**) are supported.
4. **&** as the first character of a line signals a continuation card.
5. **c** as the first character of a line signals a comment.
6. All keywords are recognized in lower case.
7. The notion of ‘column 7’ is not implemented.
8. G-format input is free form; leading blanks are ignored, the first blank after the start of the number terminates the field.
9. A comma in any numeric or logical input field terminates the field.
10. There is no carriage control on output.
11. A sequence of *n* characters in double quotes “” is equivalent to *n* **h** followed by those characters.
12. In **data** statements, a hollerith string may initialize an array or a sequence of array elements.
13. The number of storage units requested by a binary **read** must be identical to the number contained in the record being read.
14. If the first character in an input file is “#”, a preprocessor which implements “#define” and “#include” preprocessor statements is called. These preprocessor statements are similar to the corresponding C preprocessor statements; see the C reference manual for details. The preprocessor does not recognize Hollerith strings written with *n* **h**.

In I/O statements, only unit numbers 0-19 are supported. Unit number *n* refers to file *fortn*; (e.g. unit 9 is file *‘fort09’*). For input, the file must exist; for output, it will be created. Unit 5 is permanently associated with the standard input file; unit 6 with the standard output file. Also see *setfil*(III) for a way to associate unit numbers with named files.

FILES

a.out	loaded output
f.tmp[123]	temporary (deleted)
/usr/fort/errors	compile-time error messages
/usr/fort/fc1	compiler proper
/lib/fr0.o	runtime startoff
/lib/filib.a	interpreter library
/lib/libf.a	builtin functions, etc.
/lib/liba.a	system library

SEE ALSO

rc(I), which announces a pleasant Fortran dialect; the ANSI standard; *ld*(I) for loader flags. For some subroutines, try *ierror*, *getarg*, *setfil*(III).

DIAGNOSTICS

Compile-time diagnostics are given in English, accompanied if possible with the offending line number and source line with an underscore where the error occurred. Runtime diagnostics are given by number as follows:

- 1 invalid log argument
- 2 bad arg count to amod
- 3 bad arg count to atan2
- 4 excessive argument to cabs
- 5 exp too large in cexp
- 6 bad arg count to cmplx
- 7 bad arg count to dim
- 8 excessive argument to exp
- 9 bad arg count to idim
- 10 bad arg count to isign
- 11 bad arg count to mod
- 12 bad arg count to sign
- 13 illegal argument to sqrt
- 14 assigned/computed goto out of range
- 15 subscript out of range
- 16 real**real overflow
- 17 (negative real)**real
- 100 illegal I/O unit number
- 101 inconsistent use of I/O unit
- 102 cannot create output file
- 103 cannot open input file
- 104 EOF on input file
- 105 illegal character in format
- 106 format does not begin with (
- 107 no conversion in format but non-empty list
- 108 excessive parenthesis depth in format
- 109 illegal format specification
- 110 illegal character in input field
- 111 end of format in hollerith specification
- 112 bad argument to setfil
- 120 bad argument to ierror
- 999 unimplemented input conversion

BUGS

The following is a list of those features not yet implemented:

arithmetic statement functions

scale factors on input

Backspace statement.