#pragma directive ➤ 14.5

C99 provides a pragma named FP_CONTRACT that gives the programmer control over contraction. Here's how the pragma is used:

```
#pragma STDC FP_CONTRACT on-off-switch
```

The value of *on-off-switch* is either ON, OFF, or DEFAULT. If ON is selected, the compiler is allowed to contract expressions; if OFF is selected, the compiler is prohibited from contracting expressions. DEFAULT is useful for restoring the default setting (which may be either ON or OFF). If the pragma is used at the outer level of a program (outside any function definitions), it remains in effect until a subsequent FP_CONTRACT pragma appears in the same file, or until the file ends. If the pragma is used inside a compound statement (including the body of a function), it must appear first, before any declarations or statements; it remains in effect until the end of the statement, unless overridden by another pragma. A program may still call fma to perform an explicit contraction even when FP_CONTRACT has been used to prohibit automatic contraction of expressions.

Comparison Macros

```
int isgreater(real-floating x, real-floating y);
int isgreaterequal(real-floating x, real-floating y);
int isless(real-floating x, real-floating y);
int islessequal(real-floating x, real-floating y);
int islessgreater(real-floating x, real-floating y);
int isunordered(real-floating x, real-floating y);
```

Our final category consists of function-like macros that compare two numbers. These macros are designed to accept arguments of any real floating type.

The comparison macros exist because of a problem that can arise when floating-point numbers are compared using the ordinary relational operators such as < and >. If either operand (or both) is a NaN, such a comparison may cause the *invalid* floating-point exception to be raised, because NaN values—unlike other floating-point values—are considered to be unordered. The comparison macros can be used to avoid this exception. These macros are said to be "quiet" versions of the relational operators because they do their job without raising an exception.

isgreater
isgreaterequal
isless
islessequal

The isgreater, isgreaterequal, isless, and islessequal macros perform the same operation as the >, >=, <, and <= operators, respectively, except that they don't cause the *invalid* floating-point exception to be raised when the arguments are unordered.

islessgreater

The call islessgreater(x, y) is equivalent to (x) < (y) || (x) > (y), except that it guarantees not to evaluate x and y twice, and—like the previous macros—doesn't cause the *invalid* floating-point exception to be raised when x and y are unordered.

isunordered

The isunordered macro returns 1 if its arguments are unordered (at least one of them is a NaN) and 0 otherwise.