This chapter describes the numeric data types available in C and shows how to determine the class and sign of values represented in numeric data types. As stated in Chapter 2, "Floating-Point Data Formats," the PowerPC Numerics environment provides three numeric data formats: single (32 bits long), double (64 bits long), and double-double (two double formats combined, resulting in 128 bits). Each can represent normalized numbers, denormalized numbers, zeros, NaNs, and Infinities. See Chapter 2 for information about the numeric data formats and about how they represent values. Read this chapter to find out about the mapping of numeric formats to floating-point types in C, about the floating-point type declarations made in the PowerPC Numerics library (MathLib), and about the library utilities available that can determine the class of a floating-point value.

# C Data Types

Table 7-1 shows how the PowerPC Numerics data formats map to the C floating-point variable types. This mapping follows the recommendations in the FPCE technical report.

**Table 7-1** Names of data types

| PowerPC Numerics format | C type |
|---|---|
| IEEE single | `float` |
| IEEE double | `double` |
| Double-double | `long double` |

# Efficient Type Declarations

MathLib contains two floating-point type definitions, `float_t` and `double_t` in the header `Types.h`. If you define a variable to be `float_t` or `double_t`, it means "use the most efficient floating-point format for this architecture." Table 7-2 shows the definitions for `float_t` and `double_t` for both the PowerPC and 680x0 architecture.

**Table 7-2** `float_t` and `double_t` types

| Architecture | `float_t` type | `double_t` type |
|---|---|---|
| PowerPC | `float` | `double` |
| 680x0 | `long double` | `long double` |

For the PowerPC architecture, the most natural format for computations is double, but the architecture allows computations in single precision as well. Therefore, for the PowerPC architecture, `float_t` is defined to be `float` (single precision) and `double_t` is defined to be `double`. The 680x0 architecture is based on an 80-bit double-extended format (known as *extended*) and performs all computations in this format regardless of the type of the operands. Therefore, `float_t` and `double_t` are both `long double` (extended precision) for the 680x0 architecture.

If you declare a variable to be type `double_t` and you compile the program as a PowerPC application, the variable is of type `double`. If you recompile the same program as an 680x0 application, the variable is of type `long double`.

# Inquiries: Class and Sign

MathLib provides macros you can use to determine the class and sign of a floating-point value. All of these macros return type `long int`. They are listed in Table 7-3.

**Table 7-3** Class and sign inquiry macros

| Macro | Value returned | Condition |
|---|---|---|
| `fpclassify(x)` | `FP_SNAN` | x is a signaling NaN |
| | `FP_QNAN` | x is a quiet NaN |
| | `FP_INFINITE` | x is –∞ or +∞ |
| | `FP_ZERO` | x is +0 or –0 |
| | `FP_NORMAL` | x is a normalized number |
| | `FP_SUBNORMAL` | x is a denormalized (subnormal) number |
| `isnormal(x)` | `TRUE` | x is a normalized number |
| `isfinite(x)` | `TRUE` | x is not –∞, +∞, or NaN |
| `isnan(x)` | `TRUE` | x is a NaN (quiet or signaling) |
| `signbit(x)` | `1` | The sign bit of x is 1 (x is negative) |
| | `0` | The sign bit of x is 0 (x is positive) |

# Creating Infinities and NaNs

MathLib defines the constants `INFINITY` and `NAN`, so that you can assign these values to variables in your program, and provides the following function that returns NaNs:

```
double nan  (const char *tagp);
```

The `nan` function returns a quiet NaN with a fraction field that is equal to the argument `tagp`. The argument `tagp` is a pointer to a string that will be copied into bits 8 through 15 of the NaN's fraction field. The string should specify a decimal number between 0 and 255. For example:

```
nan("32")
```

creates a NaN with code 32. If you supply a negative string, it is the same as supplying the string "0". If you supply a string greater than 255, it is the same as supplying the string "255". For a list of predefined NaN codes, see Chapter 2, "Floating-Point Data Formats."

# Numeric Data Types Summary

This section summarizes the C constants, macros, functions, and type definitions associated with creating floating-point values or determining the class and sign of a floating-point value.

## C Summary

### Constants

```
#ifdef   powerc
#define          LONG_DOUBLE_SIZE     16
#elif    mc68881
#define          LONG_DOUBLE_SIZE     12
#else
#define          LONG_DOUBLE_SIZE     10
#endif      /* powerc */

#define          HUGE_VAL             __inf()
#define          INFINITY             __inf()
#define          NAN                  nan("255")
```

#### Class and Sign Inquiry Macros

```
#define  fpclassify  (x)   ((    sizeof (x) == LONG_DOUBLE_SIZE)     ?  \
                                 __fpclassify  (x)                  :  \
                                 (sizeof (x) == DOUBLE_SIZE)        ?  \
                                 __fpclassifyd (x)                  :  \
                                 __fpclassifyf (x))

#define  isnormal    (x)   ((    sizeof (x) == LONG_DOUBLE_SIZE)     ?  \
                                 __isnormal (x)                     :  \
                                 (sizeof (x) == DOUBLE_SIZE)        ?  \
                                 __isnormald (x)                    :  \
                                 __isnormalf (x))
```

```
#define  isfinite    (x)   ((    sizeof (x) == LONG_DOUBLE_SIZE)     ?  \
                                 __isfinite (x)                      :  \
                                 ( sizeof (x) == DOUBLE_SIZE)        ?  \
                                 __isfinited (x)                     :  \
                                 __isfinitef (x))

#define  isnan       (x)   ((    sizeof (x) == LONG_DOUBLE_SIZE)     ?  \
                                 __isnan (x)                         :  \
                                 (sizeof (x) == DOUBLE_SIZE)         ?  \
                                 __isnand (x)                        :  \
                                 __isnanf (x))

#define  signbit     (x)   ((    sizeof (x) == LONG_DOUBLE_SIZE)     ?  \
                                 __signbit (x)                       :  \
                                 (sizeof (x) == DOUBLE_SIZE)         ?  \
                                 __signbitd (x)                      :  \
                                 __signbitf (x))
```

## Data Types

```
enum NumberKind
{
   FP_SNAN = 0,                  /* signaling NaN */
   FP_QNAN,                      /* quiet NaN */
   FP_INFINITE,                  /* + or – infinity */
   FP_ZERO,                      /* + or – zero */
   FP_NORMAL,                    /* all normal numbers */
   FP_SUBNORMAL                  /* denormal numbers */
};

#ifdef   powerpc
   typedef float float_t;
   typedef double double_t;
#else
   typedef long double float_t;
   typedef long double double_t;
#endif      /* powerpc */
```

## Special Value Routines

### Creating NaNs

```
double nan                   (const char *tagp);
```