Chapter 4

Data Types and Literal Constants

A *data type* is a named set of values, together with a literal representation of such values (constants) and a collection of operations that interpret and manipulate the values. There are eight types of data in CM Fortran:

- integer — 32-bit integer and 64-bit integer
- real — single-precision real and double-precision real
- complex — single-precision complex and double-precision complex
- logical
- character

Each type is different and has a different internal representation. The integer, real, and complex types are *numeric types*; the logical and character types are *nonnumeric types*. The character data type has a length attribute that constrains the set of character values to those with length equal to the value of the attribute.

A *data entity* is a constant, a variable, an expression value, or a function result, and is either a scalar or an array. If the entity is an expression value, the type is determined by the rules of expression evaluation (Chapter 7).

The name of a data type may be used as a *type specifier* to specify the type and possibly other attributes of a named object or function. An object or function may be declared explicitly or implicitly. Chapter 5 describes how objects are declared and how their type and other attributes are specified.

Copyright © 1989–1994 Thinking Machines Corporation

4.1 Literal Constants

Literal constants (R4-1) represent scalar values.

A *literal-constant* is one of: R4-1

- *integer-literal-constant*
- *real-literal-constant*
- *complex-literal-constant*
- *logical-literal-constant*
- *char-literal-constant*
- *boz-literal-constant*

The syntax of each form of constant is given in the following sections.

4.2 Numeric Types and Constants

The *numeric* types are provided for numerical computation. The usual operations of arithmetic — addition (+), subtraction (−), multiplication (*), division (/), exponentiation (**), negation (unary −), and identity (unary +) — are defined intrinsically.

Each numeric type includes a zero value, which is considered to be neither negative nor positive. The value of a signed zero is the same as the value of an unsigned zero.

4.2.1 Integer Types and Constants

The *integer* types include the 32-bit integer type and the 64-bit integer type.* The type specifier for the 32-bit integer type is the keyword `INTEGER*4`, and the type specifier for the 64-bit integer type is the keyword `INTEGER*8`. The set of values for the *integer* types is a subset of the mathematical integers. The subset for the `INTEGER*4` type includes all the integer values from `-2**31` to `2**31-1`; the subset for the `INTEGER*8` type includes all the integer values from `-2**63` to `2**63-1`.

NOTE: Integer values default to `INTEGER*4`.

An *integer-literal-constant* is one of: R4-2

- *digit [digit]...*
- *digit [digit]... _scalar-int-initialization-expr*

* The 64-bit integer type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

- The value of a *scalar-int-initialization-expr* must be nonnegative and must specify a representation method that exists on the processor. In other words, it must evaluate to 32 or 64.

An *integer-literal-constant* (R4-2) is interpreted as a decimal value.

Examples of `INTEGER*4` literal constants:

```
4374
186
284
5
```

Examples of `INTEGER*8` literal constants:

```
956_64
4374_64
```

An unsigned binary, octal, or hexadecimal literal constant must correspond to an integer scalar entity.

A *boz-literal-constant* is one of: R4-3

- *binary-constant*
- *octal-constant*
- *hex-constant*

A *binary-constant* is one of: R4-4

- `B'`*digit [digit]...*`'` *[scalar-int-initialization-expr]*
- `B"`*digit [digit]...*`"` *[scalar-int-initialization-expr]*

- The value of *digit* must be 0 or 1.

An *octal-constant* is one of: R4-5

- `O'`*digit [digit]...*`'` *[scalar-int-initialization-expr]*
- `O"`*digit [digit]...*`"` *[scalar-int-initialization-expr]*

- The value of *digit* must be 0 through 7.

A *hex-constant* is one of: R4-6

- `Z'`*hex-digit [hex-digit]...*`'` *[scalar-int-initialization-expr]*
- `Z"`*hex-digit [hex-digit]...*`"` *[scalar-int-initialization-expr]*

Copyright © 1989–1994 Thinking Machines Corporation

A *hex-digit* is one of: R4-7

- *digit*
- A
- B
- C
- D
- E
- F

In these constants, the binary, octal, and hexadecimal digits are interpreted according to their respective number systems.

4.2.1.1 Restrictions on INTEGER*8 Type

INTEGER*8 values cannot be used in READ, WRITE, and PRINT statements as anything other than the I/O list items (the data to be transferred). This type cannot be used for unit numbers, record numbers, status variables, and so on.

If an INTEGER*8 variable is used as the index variable in a DO loop, the total number of iterations cannot exceed (2**31)–1.

4.2.2 Real Types and Constants

The *real* types have values that approximate the mathematical real numbers, and include the *single-precision real* type and the *double-precision real* type. The type specifier for the single-precision real type is the keyword REAL and the type specifier for the double-precision real type is the keyword DOUBLE PRECISION. Literal constants of type real can be written in several forms.

A *real-literal-constant* is one of: R4-8

- *single-precision-real-literal-constant*
- *double-precision-real-literal-constant*

A *single-precision-real-literal-constant* is one of: R4-9

- *integer-literal-constant real-exponent*
- *integer-literal-constant* . *integer-literal-constant [real-exponent]*
- . *integer-literal-constant [real-exponent]*
- *integer-literal-constant* . *[real-exponent]*

A *real-exponent* is: R4-10

E *[sign] integer-literal-constant*

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

A *double-precision-real-literal-constant* is one of: R4-11

- *integer-literal-constant double-precision-exponent*
- *integer-literal-constant* **.** *integer-literal-constant double-precision-exponent*
- **.** *integer-literal-constant double-precision-exponent*
- *integer-literal-constant* **.** *double-precision-exponent*

A *double-precision-exponent* is: R4-12

`D` *[sign] integer-literal-constant*

A *sign* is one of: R4-13

- +
- –

A *single-precision real literal constant* (R4-9) is written with an optional preceding sign and without an exponent letter, or with exponent letter `E`. A *double-precision real literal constant* (R4-11) is written with an exponent letter `D`. The value of a real constant that contains an exponent is the product of the constant that precedes the exponent letter and the power of ten indicated by the integer following the exponent letter. A real literal constant may be written with more digits before the exponent than the compiler will use to approximate the value of the constant.

A single-precision real value has about 7 decimal digits of precision, and a double-precision real value has about 16 decimal digits of precision. The absolute magnitude of a real value (in front-end memory) ranges from approximately `0.29E-38` to `1.7E+38`. The greatest positive double-precision real is `1.7976931348623158D+308`. The least positive normalized double-precision real is `2.2250738585072015D-308`; the least positive denormalized double-precision real is `5.0D-324`.

Examples of single-precision real literal constants:

```
-12.78
+1.6e3      ! 16000
2.1
```

Examples of double-precision real literal constants:

```
0.45D-4     ! 0.00045
10.93d02    ! 1093.
.123D-3     ! 0.000123
3D4         ! 30 000.
```


Copyright © 1989–1994 Thinking Machines Corporation

4.2.3 Complex Types and Constants

The *complex* type and the *double-precision complex* type have values that approximate the mathematical complex numbers. The values of a complex type are ordered pairs of real values, and literals of type complex are specified as a parenthesized pair of numeric literal constants (R4-14). The first real value is called the *real part*, and the second is called the *imaginary part*. The type specifier for the single-precision complex type is the keyword `COMPLEX`, and the type specifier for the double-precision complex type is the keyword `DOUBLE COMPLEX`.

A *complex-literal-constant* is: R4-14
- *single-precision-complex-literal-constant*
- *double-precision-complex-literal-constant*

A *single-precision-complex-literal-constant* is: R4-15
(*real-part* , *imaginary-part*)

A *double-precision-complex-literal-constant* is: R4-16
(*real-part* , *imaginary-part*)

A *real-part* is one of: R4-17
- *[sign] integer-literal-constant*
- *[sign] single-precision-real-literal-constant*
- *[sign] double-precision-real-literal-constant*

An *imaginary-part* is one of: R4-18
- *[sign] integer-literal-constant*
- *[sign] single-precision-real-literal-constant*
- *[sign] double-precision-real-literal-constant*

- Neither the *real-part* nor the *imaginary-part* of a *single-precision-complex-literal-constant* may be of type double-precision real.
- Either the *real-part* or the *imaginary-part* of a *double-precision-complex-literal-constant* must be of type double-precision real.

If either the real or the imaginary part of a complex literal constant is of type integer, the integer constant is converted to type real.

Examples of single-precision complex literal constants:

```
( 1, -1 )
( 1.0, -1.0 )
( 3, 3.1e6 )
( PI, -PI )    ! if PI is a named constant equal to 3.14159
```


Copyright © 1989–1994 Thinking Machines Corporation

Examples of double-precision complex literal constants:

```
( 1., -1.d0 )
( 3, 3.1d6 )
( PI, -PI )    ! if PI is a named constant equal to 3.14159d0
```

4.3 Character Type and Constants

The character type has a set of values composed of character strings. A *character string* is a sequence of characters, numbered from left to right starting from 1, up to the number of characters in the string. The number of characters in the string is called the *length* of the string, and is greater than zero. Any character representable in the ASCII character set may occur in a character string. Strings of different lengths are all of type `CHARACTER` (but have a different length attribute). The type specifier for the character type is the keyword `CHARACTER`. A *character literal constant* (R4-19) is written as a sequence of one or more characters, delimited by either apostrophes or quotation marks.

A *char-literal-constant* is one of: R4-19

- ' *ASCII-character [ASCII-character]...* '
- " *ASCII-character [ASCII-character]...* "

The delimiting apostrophes or quotation marks are not part of the value of the character literal constant.

An apostrophe character within a character constant delimited by apostrophes is represented as two consecutive apostrophes (without intervening blanks); in this case, the two apostrophes are counted as one character. Similarly, a quotation mark character within a character constant delimited by quotation marks is represented as two consecutive quotation marks and the two quotation marks are counted as one character.

The *concatenation* operator (//) is the only operation defined on character entities. The concatenation operator is discussed in Section 7.7 on character expressions. There are a number of intrinsic functions that accept character arguments, described in Section 17.3.

Character values are stored as a contiguous sequence of bytes in front-end memory, one byte per character.

Copyright © 1989–1994 Thinking Machines Corporation

Examples of character literal constants:

```
"Don't"
'Don"t'
"The ASCII quote ("") character's decimal value is 34."
```

The first two constants have the character value *Don't*.

4.4 Logical Type and Constants

The *logical type* has one of two values that represent the logical values true or false. A logical literal constant (R4-20) has just two forms.

A *logical-literal-constant* is one of: R4-20

- `.TRUE.`
- `.FALSE.`

The literal constant `.TRUE.` represents the truth value *true* and the literal constant `.FALSE.` represents the truth value *false*. The type specifier for the logical type is the keyword `LOGICAL`.

The operators defined for data entities of logical type are: negation (`.NOT.`), conjunction (`.AND.`), inclusive disjunction (`.OR.`), exclusive disjunction (`.XOR.`), logical equivalence (`.EQV.`), and logical nonequivalence (`.NEQV.`), as described in Section 7.9. There are also relational operators that compare the values of data entities of other types and yield a logical value; these operators are described in Section 7.8.

Logical values occupy a single numeric program unit of front-end memory.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation