Constants and Literals

On this page:

Allowable Types for Constants

Slice allows you to define constants for the following types:

- An integral type (`bool`, `byte`, `short`, `int`, `long`)
- A floating point type (`float` or `double`)
- `string`
- `enum`

Here are a few examples:

Slice

```
const bool      AppendByDefault = true;
const byte      LowerNibble = 0x0f;
const string    Advice = "Don't Panic!";
const short     TheAnswer = 42;
const double    PI = 3.1416;

enum Fruit { Apple, Pear, Orange };
const Fruit     FavoriteFruit = Pear;
```

The syntax for literals is the same as for C++ and Java (with a few minor exceptions).

Boolean constants

Boolean constants can only be initialized with the keywords `false` and `true`. (You cannot use `0` and `1` to represent `false` and `true`.)

Integer literals

Integer literals can be specified in decimal, octal, or hexadecimal notation.

For example:

Slice

```
const byte TheAnswer = 42;
const byte TheAnswerInOctal = 052;
const byte TheAnswerInHex = 0x2A;       // or 0x2a
```

Be aware that, if you interpret `byte` as a number instead of a bit pattern, you may get different results in different languages. For example, for C++, `byte` maps to `unsigned char` whereas, for Java, `byte` maps to `byte`, which is a signed type.

Note that suffixes to indicate long and unsigned constants (`l`, `L`, `u`, `U`, used by C++) are illegal:

Slice

```
const long Wrong = 0u;          // Syntax error
const long WrongToo = 1000000L; // Syntax error
```

The value of an integer literal must be within the range of its constant type, as shown in the Built-In Basic Types table; otherwise the compiler will issue a diagnostic.

Floating-point literals

Floating-point literals use C++ syntax, except that you cannot use an `l` or `L` suffix to indicate an extended floating-point constant; however, `f` and `F` are legal (but are ignored).

Here are a few examples:

Slice

```
const float P1 = -3.14f;    // Integer & fraction, with suffix
const float P2 = +3.1e-3;   // Integer, fraction, and exponent
const float P3 = .1;        // Fraction part only
const float P4 = 1.;        // Integer part only
const float P5 = .9E5;      // Fraction part and exponent
const float P6 = 5e2;       // Integer part and exponent
```

Floating-point literals must be within the range of the constant type (`float` or `double`); otherwise, the compiler will issue a diagnostic.

String literals

String literals support the same escape sequences as C++.

Here are some examples:

Slice

```
const string AnOrdinaryString = "Hello World!";

const string DoubleQuote =      "\"";
const string TwoSingleQuotes =  "'\'";     // ' and \' are OK
const string Newline =          "\n";
const string CarriageReturn =   "\r";
const string HorizontalTab =    "\t";
const string VerticalTab =      "\v";
const string FormFeed =         "\f";
const string Alert =            "\a";
const string Backspace =        "\b";
const string QuestionMark =     "\?";
const string Backslash =        "\\";

const string OctalEscape =      "\007";    // Same as \a
const string HexEscape =        "\x07";    // Ditto
```

Note that Slice has no concept of a null string:

Slice

```
const string nullString = 0;    // Illegal!
```

Null strings simply do not exist in Slice and, therefore, do not exist as a legal value for a string anywhere in the Ice platform. The reason for this decision is that null strings do not exist in many programming languages.

Many languages other than C and C++ use a byte array as the internal string representation. Null strings do not exist (and would be very difficult to map) in such languages.

A constant definition may also refer to another constant. It is not necessary for both constants to have the same Slice type, but the value of the existing constant must be compatible with the type of the constant being defined.

Consider the examples below:

Slice

```
const int SIZE = 500;

const int DEFAULT_SIZE = SIZE; // OK
const short SHORT_SIZE = SIZE; // OK
const byte BYTE_SIZE = SIZE;   // ERROR
```

The `DEFAULT_SIZE` constant is legal because it has the same type as `SIZE`, and `SHORT_SIZE` is legal because the value of `SIZE` (`500`) is within the range of the Slice `short` type. However, `BYTE_SIZE` is illegal because the value of `SIZE` is outside the range of the `byte` type.

See Also

- Enumerations
- Structures
- Sequences
- Dictionaries