Boost.Tribool

Douglas Gregor <dgregor -at- cs.indiana.edu>

Table of Contents

Introduction

The 3-state boolean library contains a single class, `boost::logic::tribool`, along with support functions and operator overloads that implement 3-state boolean logic.

Tutorial

Basic usage

The `tribool` class acts like the built-in `bool` type, but for 3-state boolean logic. The three states are `true`, `false`, and `indeterminate`, where the first two states are equivalent to those of the C++ `bool` type and the last state represents an unknown boolean value (that may be `true` or `false`, we don't know).

The `tribool` class supports conversion from `bool` values and literals along with its own `indeterminate` keyword:

```
tribool b(true);
b = false;
b = indeterminate;
tribool b2(b);
```

`tribool` supports conversions to `bool` for use in conditional statements. The conversion to `bool` will be `true` when the value of the `tribool` is always true, and `false` otherwise. Consequently, the following idiom may be used to determine which of the three states a `tribool` currently holds:

```
tribool b = some_operation();
if (b) {
  // b is true
}
else if (!b) {
  // b is false
}
else {
  // b is indeterminate
}
```

`tribool` supports the 3-state logic operators `!` (negation), `&&` (AND), and `||` (OR), with `bool` and `tribool` values. For instance:

```
tribool x = some_op();
tribool y = some_other_op();
if (x && y) {
  // both x and y are true
}
else if (!(x && y)) {
  // either x or y is false
}
else {
  // neither x nor y is false, but we don't know that both are true

  if (x || y) {
    // either x or y is true
  }
}
```

Similarly, `tribool` supports 3-state equality comparisons via the operators `==` and `!=`. These operators differ from "normal" equality operators in C++ because they return a `tribool`, because potentially we might not know the result of a comparison (try to compare `true` and `indeterminate`). For instance:

```
tribool x(true);
tribool y(indeterminate);

assert(x == x); // okay, x == x returns true
assert(x == true); // okay, can compare tribools and bools
```

The `indeterminate` keyword (representing the `indeterminate` `tribool` value) doubles as a function to check if the value of a `tribool` is indeterminate, e.g.,

```
tribool x = try_to_do_something_tricky();
if (indeterminate(x)) {
  // value of x is indeterminate
}
else {
  // report success or failure of x
}
```

Renaming the indeterminate state

Users may introduce additional keywords for the indeterminate value in addition to the implementation-supplied `indeterminate` using the `BOOST_TRIBOOL_THIRD_STATE` macro. For instance, the following macro instantiation (at the global scope) will introduce the keyword `maybe` as a synonym for `indeterminate` (also residing in the `boost` namespace):

```
BOOST_TRIBOOL_THIRD_STATE(maybe)
tribool x = maybe;
if (maybe(x)) { /* ... */ }
```

`tribool` input/output

`tribool` objects may be read from and written to streams by including the `boost/logic/tribool_io.hpp` header in a manner very similar to `bool` values. When the `boolalpha` flag is not set on the input/output stream, the integral values 0, 1, and 2 correspond to `tribool` values `false`, `true`, and `indeterminate`, respectively. When `boolalpha` is set on the stream, arbitrary strings can be used to represent the three values, the default being "false", "true", and "indeterminate". For instance:

```
tribool x;
cin >> x; // Type "0", "1", or "2" to get false, true, or indeterminate
cout << boolalpha << x; // Produces "false", "true", or "indeterminate"
```

`tribool` input and output is sensitive to the stream's current locale. The strings associated with false and true values are contained in the standard `std::numpunct` facet, and the string naming the indeterminate type is contained in the `indeterminate_name` facet. To replace the name of the indeterminate state, you need to imbue your stream with a local containing a `indeterminate_name` facet, e.g.:

```
BOOST_TRIBOOL_THIRD_STATE(maybe)
locale global;
locale test_locale(global, new indeterminate_name<char>("maybe"));
cout.imbue(test_locale);
tribool x(maybe);
cout << boolalpha << x << endl; // Prints "maybe"
```

If you C++ standard library implementation does not support locales, `tribool` input/output will still work, but you will be unable to customize the strings printed/parsed when `boolalpha` is set.
<xi:include></xi:include>

Testsuite

Acceptance tests

Test	Type	Description	If failing...
tribool_test.cpp	run	Test all features of the `boost::logic::tribool` class.	
tribool_rename_test.cpp	run	Test the use of the `BOOST_TRIBOOL_THIRD_STATE` macro.	
tribool_io_test.cpp	run	Test tribool input/output.	