Tribool Exercise

Tribool Exercise

- **Tribool:** three-valued logic
  `{false, unknown, true}`

Tribool Exercise

- **Tribool:** three-valued logic
  `{false, unknown, true}`

- Operators `AND`, `OR` exist:

AND	false	unknown	true
false	false	false	false
unknown	false	unknown	unknown
true	false	unknown	true

OR	false	unknown	true
false	false	unknown	true
unknown	unknown	unknown	true
true	true	true	true

(From: Script Exercise 142)

Exercise a)

Exercise a)

Implement a type `Tribool` which will be used to represent variables for three-valued logic.

(Remember: `{false, unknown, true}`)

Solution a)

Other solutions are of course also possible.

```
struct Tribool {
  // 0 = false, 1 = unknown, 2 = true
  unsigned int value; // INV: value in {0, 1, 2}
};
```

(This solution has handy properties for later subtasks.)

Exercise b)

Exercise b)

Implement the boolean operators `&&` and `||` for your `Tribool` type.

&&	false	unknown	true
false	false	false	false
unknown	false	unknown	unknown
true	false	unknown	true

||	false	unknown	true
false	false	unknown	true
unknown	unknown	unknown	true
true	true	true	true

(From: Script Exercise 142)

Solution b)

Other solutions also possible.

But we can benefit from representation `{0,1,2}`.

&&	false	unknown	true
false	false	false	false
unknown	false	unknown	unknown
true	false	unknown	true

||	false	unknown	true
false	false	unknown	true
unknown	unknown	unknown	true
true	true	true	true

(From: Script Exercise 142)

Solution b)

Other solutions also possible.

But we can benefit from representation `{0,1,2}`.

&&	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

||	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

(From: Script Exercise 142)

Solution b)

Other solutions also possible.

But we can benefit from representation {0,1,2

Use **minimum.**

&&	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

||	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

(From: Script Exercise 142)

Solution b)

Other solutions also possible.

But we can benefit from representation {0,1,2

Use **minimum.**

&&	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

Use **maximum.**

||	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

(From: Script Exercise 142)

Solution b)

AND:

```
// POST: returns x AND y
Tribool operator&& (const Tribool x, const Tribool y) {
  Tribool result;
  result.value = std::min(x.value, y.value);
  return result;
}
```

OR:

```
// POST: returns x OR y
Tribool operator|| (const Tribool x, const Tribool y) {
  Tribool result;
  result.value = std::max(x.value, y.value);
  return result;
}
```

(From: Script Exercise 142)

Exercise c)

Exercise c)

Determine which overloads are called. And their values.

(1)
```
Tribool operator&& (const Tribool x, const Tribool y); // as before
```

(2)
```
Tribool operator&& (const Tribool x, const bool y) {
  Tribool y_as_tribool;
  y_as_tribool.value = 2*y; // trick:  2*false == 0  and  2*true == 2
  return x && y_as_tribool;
}
```

(3)
```
Tribool operator&& (const bool x, const Tribool y) {
  return y && x;
}
```

```
Tribool t; t.value = 1; // unknown
t && true;
t && t;
false && t;
false && true;
```

Exercise c)

Determine which overloads are called. And their values.

(1)
(2)
(3)

```
Tribool operator&& (const Tribool x, const Tribool y); // as before

Tribool operator&& (const Tribool x, const bool y) {
  Tribool y_as_tribool;
  y_as_tribool.value = 2*y; // trick:  2*false == 0  and  2*true == 2
  return x && y_as_tribool;
}

Tribool operator&& (const bool x, const Tribool y) {
  return y && x;
}
```

```
Tribool t; t.value = 1; // unknown
t && true;              // (2) (1)          value: unknown
t && t;
false && t;
false && true;
```

Exercise c)

Determine which overloads are called. And their values.

(1)
(2)
(3)

```
Tribool operator&& (const Tribool x, const Tribool y); // as before

Tribool operator&& (const Tribool x, const bool y) {
  Tribool y_as_tribool;
  y_as_tribool.value = 2*y; // trick:  2*false == 0  and  2*true == 2
  return x && y_as_tribool;
}

Tribool operator&& (const bool x, const Tribool y) {
  return y && x;
}
```

```
Tribool t; t.value = 1; // unknown
t && true;              // (2) (1)          value: unknown
t && t;                 // (1)              value: unknown
false && t;
false && true;
```

Exercise c)

Determine which overloads are called. And their values.

(1)
(2)
(3)

```
Tribool operator&& (const Tribool x, const Tribool y); // as before

Tribool operator&& (const Tribool x, const bool y) {
  Tribool y_as_tribool;
  y_as_tribool.value = 2*y; // trick:  2*false == 0  and  2*true == 2
  return x && y_as_tribool;
}

Tribool operator&& (const bool x, const Tribool y) {
  return y && x;
}
```

```
Tribool t; t.value = 1; // unknown
t && true;              // (2) (1)          value: unknown
t && t;                 // (1)              value: unknown
false && t;             // (3) (2) (1)      value: false
false && true;
```

Exercise c)

Determine which overloads are called. And their values.

(1)
(2)
(3)

```
Tribool operator&& (const Tribool x, const Tribool y); // as before

Tribool operator&& (const Tribool x, const bool y) {
  Tribool y_as_tribool;
  y_as_tribool.value = 2*y; // trick:  2*false == 0  and  2*true == 2
  return x && y_as_tribool;
}

Tribool operator&& (const bool x, const Tribool y) {
  return y && x;
}
```

```
Tribool t; t.value = 1; // unknown
t && true;              // (2) (1)          value: unknown
t && t;                 // (1)              value: unknown
false && t;             // (3) (2) (1)      value: false
false && true;          // This is mean! It's the one for bool. ;-)
                        //                  value: false
```

Exercise d)

Exercise d)

Overload the output operator `<<` for your `Tribool` type:

```
// POST: Tribool value is written to o
std::ostream& operator<< (std::ostream& o, const Tribool x);
```

(Hint: You can think of `o` as `std::cout`.
In fact, `std::cout` is of type `std::ostream`.

This means that your overload allows you to write
`std::cout << my_tribool;`)

Solution d)

Solution (a very compact form):

```
// POST: Tribool value is written to o
std::ostream& operator<< (std::ostream& o, const Tribool x) {
  if      (x.value == 0)  return o << "false  ";
  else if (x.value == 1)  return o << "unknown";
  else                    return o << "true   ";
}
```

(From: Script Exercise 142)

Solution d)

Solution (a very compact form):

```
// POST: Tribool value is written to o
std::ostream& operator<< (std::ostream& o, const Tribool x) {
  if      (x.value == 0)  return o << "false  ";
  else if (x.value == 1)  return o << "unknown";
  else                    return o << "true   ";
}
```

Remark:

This is `operator<<` for strings. (*)

(*) It is the one you're using whenever you output something: `std::cout << "Hello";`

Verify Implementation

Verify Code

- Print truth table to test implementation:

```
// Truth Table for &&
Tribool f; f.value = 0; // false
Tribool u; u.value = 1; // unknown
Tribool t; t.value = 2; // true

std::cout << (f && f) << (f && u) << (f && t) << "\n"
          << (u && f) << (u && u) << (u && t) << "\n"
          << (t && f) << (t && u) << (t && t) << "\n";
```

(From: Script Exercise 142)

Verify Code

- Print truth table to test implementation:

```
// Truth Table for &&
Tribool f; f.value = 0; // false
Tribool u; u.value = 1; // unknown
Tribool t; t.value = 2; // true

std::cout << (f && f) << (f && u) << (f && t) << "\n"
          << (u && f) << (u && u) << (u && t) << "\n"
          << (t && f) << (t && u) << (t && t) << "\n";
```

```
Output is:

        false    false    false
        false    unknown  unknown
        false    unknown  true
```

(From: Script Exercise 142)