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

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)

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

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

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

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

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

Use **minimum.**

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

Use **maximum.**

Solution b)

AND:

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

OR:

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

Short Summary

Tribool Exercise

- Short Summary:

struct:

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

&& :

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

|| :

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

Tribool Exercise

- Short Summary:

struct:

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

&& :

```
// POST: return
Tribool ope
  Trib
                                        );
                                     value;
```

Is this correct?

```
           x OR y
     rator|| (const Tribool x, const Tribool y) {
     ool result;
  if (x.value > y.value)   result.value = x.value;
  else                     result.value = y.value;
  return result;
}
```

Tribool Exercise

Is this correct?

- Want some correctness verification!
- Check: **output truth tables** with our operators.

Tribool Exercise

- Want to use something like:

```
// Truth Table for &&
for (int x_val = 0; x_val < 3; ++x_val) {
  for (int y_val = 0; y_val < 3; ++y_val)
    std::cout << (to_Trib(x_val) && to_Trib(y_val)) << "  ";

  std::cout << "\n";
}
```

Tribool Exercise

- Want to use something like:

```
// Truth Table for &&
for (int x_val = 0; x_val < 3; ++x_val) {
  for (int y_val = 0; y_val < 3; ++y_val)
    std::cout << (to_Trib(x_val) && to_Trib(y_val)) << "  ";

  std::cout << "\n";
}
```

Problem 1:

We need a «converter»

Tribool Exercise

- Want to use something like:

```
// Truth Table for &&
for (int x_val = 0; x_val < 3; ++x_val) {
  for (int y_val = 0; y_val < 3; ++y_val)
    std::cout << (to_Trib(x_val) && to_Trib(y_val)) << "  ";

  std::cout << "\n";
}
```

Problem 2:

We need <<

Problem 1:

We need a «converter»

Back to Exercise

Exercise c)

Write the «converter» function:

```
// PRE: val in {0, 1, 2}
// POST: return value is a Tribool with the
//         corresponding value
Tribool to_Trib (const unsigned int val);
```

Solution c)

Solution:

```
// PRE: val in {0, 1, 2}
// POST: return value is a Tribool with the
//       corresponding value
Tribool to_Trib (const unsigned int val) {
  assert (val <= 2);

  Tribool result;
  result.value = val;
  return result;
}
```

Exercise d)

Write the output operator `<<` for your `Tribool` type.

Solution d)

Solution (a very compact form):

```
// POST: Tribool value is written to std::cout
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   ";
}
```

And Our Test Output…

Tribool Exercise

- Want to use something like:

```
// Truth Table for &&
for (int x_val = 0; x_val < 3; ++x_val) {
  for (int y_val = 0; y_val < 3; ++y_val)
    std::cout << (to_Trib(x_val) && to_Trib(y_val)) << "  ";

  std::cout << "\n";
}
```

Tribool Exercise

- Want to use something like:

```
// Truth Table for &&
for (int x_val = 0; x_val < 3; ++x_val) {
  for (int y_val = 0; y_val < 3; ++y_val)
    std::cout << (to_Trib(x_val) && to_Trib(y_val)) << "  ";

  std::cout << "\n";
}
```

```
Output is:

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