Solution to Exercise 108.

There are several possible representations for three-valued logic. A convenient one is based on the following observation: if we interpret *false* as 0, *unknown* as 1 and *true* as 2, then AND corresponds to the minimum of the two numbers, while OR corresponds to the maximum. The following program defines the type `Tribool` and the two operators, and it uses them to reproduce the truth tables.

```
1  // Prog: Tribool.C
2  // imp,ements three-valued logic
3
4  #include<iostream>
5  #include<cassert>
6
7  struct Tribool {
8    // INV : value in {0, 1, 2}
9    // 0 = false, 1 = unknown, 2 = true
10   unsigned int value;
11 };
12
13 // PRE: val in {0, 1, 2}
14 // POST: return value is a Tribool with the corresponding value
15 Tribool tribool (unsigned int val)
16 {
17   assert (val <= 2);
18   Tribool result;
19   result.value = val;
20   return result;
21 }
22
23 // POST: returns x AND y
24 Tribool operator&& (Tribool x, Tribool y)
25 {
26   Tribool result;
27   if (x.value < y.value)
28     result.value = x.value;
29   else
30     result.value = y.value;
31   return result;
32 }
33
34 // POST: returns x OR y
35 Tribool operator|| (Tribool x, Tribool y)
36 {
37   Tribool result;
38   if (x.value > y.value)
39     result.value = x.value;
40   else
41     result.value = y.value;
42   return result;
43 }
44
45 // POST: Tribool value is written to std::cout
46 void print (Tribool x)
47 {
48   if      (x.value == 0)  std::cout << "false   ";
49   else if (x.value == 1)  std::cout << "unknown ";
50   else                    std::cout << "true    ";
51 }
52
53 int main()
54 {
```

```
55    // print 3 x 3 truth table for AND
56    for (int x_val = 0; x_val < 3; ++x_val) {
57      Tribool x = tribool (x_val);
58      for (int y_val = 0; y_val < 3; ++y_val) {
59        Tribool y = tribool (y_val);
60        print (x && y);
61      }
62      std::cout << "\n";
63    }
64    std::cout << "\n";
65
66    // print 3 x 3 truth table for OR
67    for (int x_val = 0; x_val < 3; ++x_val) {
68      Tribool x = tribool (x_val);
69      for (int y_val = 0; y_val < 3; ++y_val) {
70        Tribool y = tribool (y_val);
71        print (x || y);
72      }
73      std::cout << "\n";
74    }
75    std::cout << "\n";
76
77
78    return 0;
79  }
```

Solution to Exercise 109.

The most natural representation is by an `unsigned int` value in the range $\{0, \ldots, 6\}$. Addition then simply adds the values and takes the result modulo 7. Subtraction could in principle be realized by subtratcing the values and taking the result modulo 7, but the problem is that the intermediate subtraction result could be negative (and therefore not representable as `unsigned int` value). To overcome this problem, we simply add 7 to the first value before we subtract the second one; this is guaranteed to yield a positive value, and modulo 7, it makes no difference. The following program defines the type Z_7 and the two operators, and it uses them to reproduce the addition and subtraction table.

```
1   #include<iostream>
2
3   struct Z_7 {
4     // INV : value in {0, 1, 2, 3, 4, 5, 6}
5     unsigned int value;
6   };
7
8   // POST: return value is the sum of a and b
9   Z_7 operator+ (Z_7 a, Z_7 b)
10  {
11    Z_7 result;
12    result.value = (a.value + b.value) % 7;
13    return result;
14  }
15
16  // POST: return value is the difference of a and b
17  Z_7 operator- (Z_7 a, Z_7 b)
18  {
19    Z_7 result;
20    result.value = (7 + a.value - b.value) % 7;
21    return result;
```

```
22  }
23
24  int main ()
25  {
26    Z_7 a;
27    Z_7 b;
28
29    // print table for addition
30    for (unsigned int a_val = 0; a_val < 7; ++a_val) {
31      for (unsigned int b_val = 0; b_val < 7; ++b_val) {
32        a.value = a_val;
33        b.value = b_val;
34        std::cout << (a + b).value << " ";
35      }
36      std::cout << "\n";
37    }
38    std::cout << "\n";
39
40    // print table for subtraction
41    for (unsigned int a_val = 0; a_val < 7; ++a_val) {
42      for (unsigned int b_val = 0; b_val < 7; ++b_val) {
43        a.value = a_val;
44        b.value = b_val;
45        std::cout << (a - b).value << " ";
46      }
47      std::cout << "\n";
48    }
49    std::cout << "\n";
50
51    return 0;
52  }
```

Solution to Exercise 110. The following program is at the same time the solution to the next Exercise 111.

```
1   // Program: rational.C
2   // Define and use operations on rational numbers.
3
4   #include <iostream>
5
6   // the new type Rational
7   struct Rational {
8     int n;
9     int d; // INV: d != 0
10  };
11
12  // POST: return value is the sum of a and b
13  Rational operator+ (Rational a, Rational b)
14  {
15    Rational result;
16    result.n = a.n * b.d + a.d * b.n;
17    result.d = a.d * b.d;
18    return result;
19  }
20
21  // POST: return value is the difference of a and b
22  Rational operator- (Rational a, Rational b)
23  {
24    Rational result;
25    result.n = a.n * b.d - a.d * b.n;
26    result.d = a.d * b.d;
27    return result;
```

```
28  }
29
30  // POST: return value is the product of a and b
31  Rational operator* (Rational a, Rational b)
32  {
33    Rational result;
34    result.n = a.n * b.n;
35    result.d = a.d * b.d;
36    return result;
37  }
38
39  // POST: return value is the quotient of a and b
40  // PRE:  b != 0
41  Rational operator/ (Rational a, Rational b)
42  {
43    Rational result;
44    result.n = a.n * b.d;
45    result.d = a.d * b.n;
46    return result;
47  }
48
49  // POST: return value is true if and only if a == b
50  bool operator== (Rational a, Rational b)
51  {
52    return a.n * b.d == a.d * b.n;
53  }
54
55  // POST: return value is true if and only if a != b
56  bool operator!= (Rational a, Rational b)
57  {
58    return !(a == b);
59  }
60
61  // POST: return value is true if and only if a < b
62  bool operator< (Rational a, Rational b)
63  {
64    // here we have to watch out for signs
65    if (a.d > 0 && b.d > 0 || a.d < 0 && b.d < 0)
66      // no sign reversal in multplying by a.d and b.d
67      return a.n * b.d < a.d * b.n;
68    else
69      // sign reversal
70      return a.n * b.d > a.d * b.n;
71  }
72
73  // POST: return value is true if and only if a <= b
74  bool operator<= (Rational a, Rational b)
75  {
76    return a < b || a == b;
77  }
78
79  // POST: return value is true if and only if a > b
80  bool operator> (Rational a, Rational b)
81  {
82    return b < a;
83  }
84
85  // POST: return value is true if and only if a >= b
86  bool operator>= (Rational a, Rational b)
87  {
88    return a > b || a == b;
89  }
90
91
92  // POST: a has been written to o
```

```
93  std::ostream& operator<< (std::ostream& o, Rational a)
94  {
95    return o << a.n << "/" << a.d;
96  }
97
98  // POST: a has been read from i
99  // PRE:  i starts with a rational number of the form "n/d"
100 std::istream& operator>> (std::istream& i, Rational& a)
101 {
102   char c; // separating character, e.g. '/'
103   return i >> a.n >> c >> a.d;
104 }
105
106 int main ()
107 {
108   // input
109   std::cout << "Rational number r:\n";
110   Rational r;
111   std::cin >> r;
112
113   std::cout << "Rational number s:\n";
114   Rational s;
115   std::cin >> s;
116
117   // test the operations
118   std::cout << "Sum is        " << r + s    << ".\n";
119   std::cout << "Difference is " << r - s    << ".\n";
120   std::cout << "Product is    " << r * s    << ".\n";
121   std::cout << "Quotient is   " << r / s    << ".\n";
122   std::cout << "r == s?       " << (r == s) << ".\n";
123   std::cout << "r != s?       " << (r != s) << ".\n";
124   std::cout << "r < s?        " << (r < s)  << ".\n";
125   std::cout << "r <= s?       " << (r <= s) << ".\n";
126   std::cout << "r > s?        " << (r > s)  << ".\n";
127   std::cout << "r >= s?       " << (r >= s) << ".\n";
128
129   return 0;
130 }
```

Solution to Exercise 111. See solution to Exercise 110.

Solution to Exercise 112. It turns out that only `operator+` requires real work; the following program evaluates it according to the case distinction in the small table that is given.

```
1   #include<iostream>
2
3   struct extended_int {
4     unsigned int u;  // the absolute value
5     bool         n;  // the sign (true means negative)
6   };
7
8   // POST: return value is the sum of a and b
9   //              ||  a > b     ||  a <= b
10  // ====================================
11  // (a,+)+(b,+) ||          (a+b,+)
12  // (a,+)+(b,-) || (a-b,+)     ||  (b-a,-)
13  // (a,-)+(b,+) || (a-b,-)     ||  (b-a,+)
14  // (a,-)+(b,-) ||          (a+b,-)
15  extended_int operator+ (extended_int a, extended_int b)
16  {
17    extended_int result;
```

```
18    if (a.n == b.n) {
19      result.u = a.u + b.u;
20      result.n = a.n;
21    }
22    else
23      if (a.u > b.u) {
24        result.u = a.u - b.u;
25        result.n = a.n;
26      }
27      else {
28        result.u = b.u - a.u;
29        result.n = b.n;
30      }
31    return result;
32  }
33
34  // POST: return value is -a
35  extended_int operator- (extended_int a)
36  {
37    a.n = !a.n;
38    return a;
39  }
40
41  // POST: return value is the difference of a and b
42  extended_int operator- (extended_int a, extended_int b)
43  {
44    return a + (-b);
45  }
46
47  // POST: return value is the product of a and b
48  extended_int operator* (extended_int a, extended_int b)
49  {
50    extended_int result;
51    result.u = a.u * b.u;
52    result.n = (a.n || b.n) && !(a.n && b.n); // XOR
53    return result;
54  }
55
56  // POST: a has been written to o
57  std::ostream& operator<< (std::ostream& o, extended_int a)
58  {
59    if (a.n) o << "-";
60    return o << a.u;
61  }
62
63  // POST: a has been set to i
64  void set (extended_int& a, int i)
65  {
66    if (i < 0) {
67      a.u = -i; a.n = true;
68    } else {
69      a.u = i; a.n = false;
70    }
71  }
72
73  // now test it
74  int main() {
75    extended_int x;
76    extended_int y;
77    for (int i = -1; i < 2; ++i)
78      for (int j = -1; j < 2; ++j) {
79        // set x to i
80        set (x, i);
81        set (y, j);
82        std::cout << "x = " << x << ", y = " << y << "\n";
```

```
83        std::cout << " x + y = " << x + y << "\n";
84        std::cout << " x - y = " << x - y << "\n";
85        std::cout << " x * y = " << x * y << "\n";
86      }
87    return 0;
88  }
```

Solution to Exercise 113. Let us use the following shortcuts:

E exact match
P promotion
S standard conversion

Here is the table of match qualities for the two parameters.

	A	B	C
a)	(S,S)	**(S,E)**	(S,S)
b)	(S,P)	(E,S)	(S,S)
c)	(E,S)	(S,E)	(S,S)
d)	(S,S)	(S,S)	**(S,E)**
e)	**(S,P)**	(S,S)	(S,S)
f)	(P,E)	(S,S)	(E,S)

A best match is indicated in bold. This implies that a) resolves to B; b),c) are ambiguous; d) resolves to C; e) resolves to A; f) is ambiguous.

Solution to Exercise 114.

```
1   // Prog: geometry.C
2   // defines basic 3d geometry operations on points (rotations,
3   // projections), and uses them to rotate a cube
4
5   #include <IFM/window>
6   #include <iostream>
7   #include <cmath>
8
9   // 2d points
10  // ---------
11  struct point2 {
12    double x;
13    double y;
14  };
15
16  // 3d points
17  // ---------
18  struct point3 {
19    double x;
20    double y;
21    double z;
22  };
23
24  // rotations
25  // ---------
26  // POST: return value is p rotated around the x-axis by alpha (radians)
27  point3 rotate_x (point3 p, double alpha)
28  {
29    point3 r;
30    double cosa = std::cos(alpha);
31    double sina = std::sin(alpha);
```

```
32    r.x = p.x;
33    r.y = cosa * p.y - sina * p.z;
34    r.z = sina * p.y + cosa * p.z;
35    return r;
36  }
37
38  // POST: return value is p rotated around the y-axis by alpha (radians)
39  point3 rotate_y (point3 p, double alpha)
40  {
41    point3 r;
42    double cosa = std::cos(alpha);
43    double sina = std::sin(alpha);
44    r.x = sina * p.z + cosa * p.x;
45    r.y = p.y;
46    r.z = cosa * p.z - sina * p.x;
47    return r;
48  }
49
50  // POST: return value is p rotated around the z-axis by alpha (radians)
51  point3 rotate_z (point3 p, double alpha)
52  {
53    point3 r;
54    double cosa = std::cos(alpha);
55    double sina = std::sin(alpha);
56    r.x = cosa * p.x - sina * p.y;
57    r.y = sina * p.x + cosa * p.y;
58    r.z = p.z;
59    return r;
60  }
61
62  // projection
63  // ----------
64  // PRE: v.z != p.z
65  // POST: return value is projection of p onto the plane z = 0,
66  //       with respect to view point v
67  point2 project (point3 p, point3 v)
68  {
69    double t = p.z / (v.z - p.z);
70    point2 r;
71    r.x = p.x - t * (v.x - p.x);
72    r.y = p.y - t * (v.y - p.y);
73    return r;
74  }
75
76  // line drawing
77  // ------------
78  // POST: draws a line from the projection of p to the projection
79  //       of q with respect to viewpoint v, and with offset o
80  void draw_line (ifm::Wstream w, point3 p, point3 q, point3 v, point2 o)
81  {
82    point2 p2 = project (p, v);
83    point2 q2 = project (q, v);
84    w << ifm::Line
85      (int(p2.x+o.x), int(p2.y+o.y), int(q2.x+o.x), int(q2.y+o.y));
86  }
87
88  int main()
89  {
90    // construct a cube
91    point3 cube[8];
92    int size = 200;
93    int i = 0;
94    for (double x=-size/2; x <= size/2; x += size)
95      for (double y=-size/2; y <= size/2; y += size)
96        for (double z=-size/2; z <= size/2; z += size) {
```

```
 97          cube[i].x = x;
 98          cube[i].y = y;
 99          cube[i].z = z;
100          ++i;
101        }
102
103    // open window
104    ifm::Wstream w ("Rotate cube: press x, y, z or q to quit");
105
106    // drawing loop
107    double alpha = 0.1; // rotation angle
108    point3 v;           // viewpoint
109    v.x = 200;
110    v.y = 200;
111    v.z = 1000;
112    point2 o;           // offset
113    o.x = 250;
114    o.y = 250;
115    for (;;) {
116      // draw all the 12 dube edges
117      w.clear();
118      draw_line (w, cube[0], cube[1], v, o);
119      draw_line (w, cube[0], cube[2], v, o);
120      draw_line (w, cube[0], cube[4], v, o);
121      draw_line (w, cube[1], cube[3], v, o);
122      draw_line (w, cube[1], cube[5], v, o);
123      draw_line (w, cube[2], cube[3], v, o);
124      draw_line (w, cube[2], cube[6], v, o);
125      draw_line (w, cube[3], cube[7], v, o);
126      draw_line (w, cube[4], cube[5], v, o);
127      draw_line (w, cube[4], cube[6], v, o);
128      draw_line (w, cube[5], cube[7], v, o);
129      draw_line (w, cube[6], cube[7], v, o);
130      w << ifm::flush;
131
132      // wait for keystroke
133      int key = w.get_key();
134      // we assume ASCII encoding
135      switch (key) {
136      case 'x':
137        for (int i=0; i<8; ++i)
138          cube[i] = rotate_x (cube[i], alpha);
139        break;
140      case 'y':
141        for (int i=0; i<8; ++i)
142          cube[i] = rotate_y (cube[i], alpha);
143        break;
144      case 'z':
145        for (int i=0; i<8; ++i)
146          cube[i] = rotate_z (cube[i], alpha);
147        break;
148      case 'q':
149        return 0;
150        break;
151      default:
152        break; // do nothing
153      }
154    }
155
156    return 0;
157
158  }
```