```
1  // Implementation of the eight queens problem
2  // YOUR NAME HERE
3
4  #include <iostream>
5
6  using namespace std;
7
8  const int nrows = 8;
9  const int ncols = 8;
10
11 // For the board, zero means unoccupied and not attacked
12 // -1 means occupied
13 // Non-zero is the number of previously placed pieces attacking this space
14 int board[nrows][ncols];
15 int nSolutions = 0;
16
17 void PrintSolution()
18 {
19   for (int r = 0; r < nrows; ++r)
20     {
21       for (int c = 0; c < ncols; ++c)
22         {
23           if (board[r][c] < 0)      cout << " Q ";
24           else                      cout << " - ";
25         }
26       cout << endl;
27     }
28 }
29
30
31 void Place(int r, int c)
32 {
33   board[r][c] = -1; // Occupied
34   for (int c1 = 0; c1 < ncols; ++c1)
35     {
36       if (c != c1) board[r][c1]++; // attacked
37     }
38   for (int r1 = 0; r1 < nrows; ++r1)
39     {
40       if (r != r1) board[r1][c]++; // attacked
41     }
42   int r1 = r - 1;
43   int c1 = c - 1;
44   // upper left diagonal
45   while (r1 >= 0 && c1 >= 0)
46     {
47       board[r1--][c1--]++;
48     }
49   r1 = r - 1;
50   c1 = c + 1;
51   // upper right diagonal
52   while (r1 >= 0 && c1 < ncols)
53     {
54       board[r1--][c1++]++;
55     }
56   r1 = r + 1;
```

Program queens.cc

```
  c1 = c - 1;
  // lower left diagonal
  while (r1 < nrows && c1 >= 0)
    {
      board[r1++][c1--]++;
    }
  r1 = r + 1;
  c1 = c + 1;
  // lower right diagonal
  while (r1 < nrows && c1 < ncols)
    {
      board[r1++][c1++]++;
    }
}

void Remove(int r, int c)
{
  board[r][c] = 0; // Not occupied
  for (int c1 = 0; c1 < ncols; ++c1)
    {
      if (c1 != c) board[r][c1]--; // no longer attacked
    }
  for (int r1 = 0; r1 < nrows; ++r1)
    {
      if (r1 != r) board[r1][c]--; // no longer attacked
    }
  int r1 = r - 1;
  int c1 = c - 1;
  // upper left diagonal
  while (r1 >= 0 && c1 >= 0)
    {
      board[r1--][c1--]--;
    }
  r1 = r - 1;
  c1 = c + 1;
  // upper right diagonal
  while (r1 >= 0 && c1 < ncols)
    {
      board[r1--][c1++]--;
    }
  r1 = r + 1;
  c1 = c - 1;
  // lower left diagonal
  while (r1 < nrows && c1 >= 0)
    {
      board[r1++][c1--]--;
    }
  r1 = r + 1;
  c1 = c + 1;
  // lower right diagonal
  while (r1 < nrows && c1 < ncols)
    {
      board[r1++][c1++]--;
    }
}

```

Program queens.cc (continued)

```
113
114
115  bool verbose = true;
116
117  void Try(int c)
118  {
119    if (c == ncols)
120      { // Solution found
121        nSolutions++;
122        if (verbose)
123          {
124            cout << "Found solution number " << nSolutions
125                 << endl;
126            PrintSolution();
127          }
128        return;
129      }
130
131    for (int r = 0; r < nrows; ++r)
132      {
133        // if (!verbose && c <= 1) cout << "Trying col " << c
134        //                              << " row " << r << endl;
135        if (board[r][c] == 0)
136          { // We can place on this square
137            Place(r,c);
138            Try(c + 1);
139            Remove(r,c);
140          }
141      }
142  }
143
144  int main(int argc, char** argv)
145  {
146    Try(0);
147    cout << "Found " << nSolutions << " solutions" << endl;
148  }
```

Program queens.cc (continued)