ANSI C with elements of C^{++}

Task List 8 (Christmas and New Year Special Exercise)

Due on 06.01.2011

Because Christmas time is magic time, the ANSI C Christmas Exercise is about magic squares. A magic square of size N is a square that is filled with the numbers from 1 to N^2. Each number occurs exactly once in the square. The numbers in each row, in each column, and in each diagonal have the same sum. This is a magic square of size 3 :

8	1	6
3	5	7
4	9	2

We will write a program that searches for magic squares of size 4. This is done by guessing numbers in array `square[4][4]`. The value 0 is used for positions that are not yet filled in. It can be checked that the row/column/diagonal sum in a magic square of size 4 has to be equal to 34.

1. Write a function `clear( int p[4][4] )` that fills a square with zeroes.
2. Write a function `printsquare( int p[4][4] )` that prints a square.
3. Write a function `int rowsumcorrect( int p[4][4], i )` that returns 1 if the sum of the i-th row equals 34. (Rows are horizontal. For example, `p[0][0] ... p[0][3]` is a row.)
4. Write a function `int columnsumcorrect( int p[4][4], i )` that returns 1 if the sum in the i-th column equals 34. (Columns are vertical. For example, `p[0][0] ... p[3][0]` is a row.)
5. Write a function `int diagtlbrsumcorrect( int p[4][4] )` that returns 1 if the sum in the diagonal from top left to bottom right equals 34.
6. Write a function `int diagbltrsumcorrect( int p[4][4] )` that returns 1 if the sum in the diagonal from bottom left to top right equals 34.
7. Write a function `int occurs( int p[4][4], int i )` that returns 1 if the number i occurs in the square p.

8. We are going to search for a magic square by trying. The order in which we will be searching is defined by the following square. The first position to be guessed has number 1. The second position has number 2. From the square below, one can see that we first guess position $(0, 0)$, then $(0, 1)$, $(0, 2)$, $(0, 3)$, $(1, 2)$, etc. The last position is $(3, 3)$.

1	9	8	7
2	10	6	12
3	5	13	15
4	11	14	16

The main program has the following form:

```
main( ... )
{
   int square[4][4];

   clear(p);

   searchsquares( 1, square );
   printf( "there are no more magic squares\n" );
}


// We guess a number on the n-th position, and then continue
// searching. If n = 17, the square is complete and we
// (proudly) print it.

void searchsquares( int level, int p [4][4] )
{
   // If level == 17, we found a solution.

   if( level == 17 )
   {
      printsquare(p);
      return;
   }

   // Our first task is to decide which position we try to fill in:

   int x,y;
   switch( level )
   {
   case 1:
```

```
   x = 0; y = 0; break; // The numbers come from the 4X4 square above.
case 2:
   x = 0; y = 1; break;
....
case 10:
   x = 1; y = 1; break;
case 11:
   x = 1; y = 3; break;


case 16:
   x = 3; y = 3; break;
}

// Next step is to try the possibilities on point (x,y)
// and to check their correctness:

for( int guess = 1; guess < 17; ++ guess )
{
   if( ! occurs( p, guess ))
   {
      p[x][y] = guess;
         // Fill in position.

      // You have to decide which checks must be done at level
      // n.

      int ok = 0;
      switch(i)
      {
      case 4:
         if( columnsumcorrect(p,0))
            ok = 1;
         break;
      case 7:
         if( diagbltrsumcorrect(p))
            ok = 1;
         break;
      case 9:
         if( rowcorrect(p,0))
            ok = 1;
         break;
      .....  (many more cases.)
      case 16:
         if( rowcorrect(p,3) &&
             columncorrect(p,3) &&
                   diagtlbrcorrect(p))
                  ok = 1;
               break;
            }

            // If ok, then the number that we filled in, passed the checks.
            // We continue to the next level.

            if( ok )
               searchsquares( level + 1, p );

         }
      }
      p[x][y] = 0;

   }
```

If you want, you can add a counter to `findsquares`, so that it counts the solutions.