

**1. Intro.** Serhiy Grabarchuk noticed that there are ten ways to glue a small domino over a large domino, where the sides of the large domino are  $\sqrt{2}$  times as long as the sides of the small one, and the small domino is at a  $45^\circ$  angle with respect to the large one. (This means that the two grids line up nicely.) This program generates DLX data to pack those ten pieces into an  $m \times n$  box—meaning a rectangular box that could hold  $mn/2$  large dominoes, if there were no small ones. (Such a box could also hold  $mn - m - n$  small dominoes, tilted, if there were no large ones.)

Let's use Cartesian coordinates instead of matrix-like coordinates. Imagine that the grid for large dominoes is bounded by the lines  $x = 0$ ,  $x = 2n$ ,  $y = 0$ , and  $y = 2m$ . Represent each large square by its midpoint. Thus the  $mn$  large squares are  $(x, y)$  for  $0 < x < 2n$  and  $0 < y < 2m$ , where  $x$  and  $y$  are odd.

The center of every large square is a corner point of four small squares. And the center point of every small square is  $(x, y)$  where  $x + y$  is odd. That makes  $n(m - 1)$  cases with  $x$  odd, and  $(n - 1)m$  cases with  $x$  even.

The pieces are numbered 0 to 9, somewhat arbitrarily.

This program simply packs the pieces without overlapping. With change files I can add other constraints.

```
#include <stdio.h>
#include <stdlib.h>
int m, n; /* command-line parameters */
int piece[10][4] = {{0, 3, 1, 4}, {1, 2, 2, 3}, {0, 1, 1, 2}, {1, 0, 2, 1}, {0, -1, 1, 0}, {-1, 0, 0, -1}, {-2, 1, -1, 0},
{-1, 2, 0, 1}, {-2, 3, -1, 2}, {-1, 4, 0, 3}};
int work[8];
⟨Subroutine 3⟩;
main(int argc, char *argv[])
{
    register int d, k, p, x, y, xmin, xmax, ymin, ymax;
    ⟨Process the command line 2⟩;
    ⟨Print the item-name line 4⟩;
    for (p = 0; p < 10; p++)
        for (d = 0; d < 4; d++) ⟨Print the options for piece p rotated d times 5⟩;
}

2. ⟨Process the command line 2⟩ ≡
if (argc ≠ 3 ∨ sscanf(argv[1], "%d", &m) ≠ 1 ∨ sscanf(argv[2], "%d", &n) ≠ 1) {
    fprintf(stderr, "Usage: %s m n\n", argv[0]);
    exit(-1);
}
if (m ≥ 31 ∨ n ≥ 31) {
    fprintf(stderr, "Sorry, m and n must be less than 31!\n");
    exit(-2);
}
printf("%s %d %d\n", argv[0], m, n);
```

This code is used in section 1.

3.  $\langle \text{Subroutine 3} \rangle \equiv$   
**char** *encode*(**int** *x*)  
{  
    **if** (*x* < 0) **return** '?';  
    **if** (*x* < 10) **return** '0' + *x*;  
    **if** (*x* < 36) **return** 'a' + (*x* − 10);  
    **if** (*x* < 62) **return** 'A' + (*x* − 36);  
    **return** '?';  
}

This code is used in section 1.

4.  $\langle \text{Print the item-name line 4} \rangle \equiv$   
**for** (*p* = 0; *p* < 10; *p*++) *printf*("d□", *p*);  
*printf*("|");  
**for** (*x* = 1; *x* < *n* + *n*; *x*++)  
    **for** (*y* = 1; *y* < *m* + *m*; *y*++)  
        **if** ((*x* & 1) ∨ (*y* & 1)) *printf*("□%c%c", *encode*(*x*), *encode*(*y*));  
*printf*("□\n");

This code is used in section 1.

```

5.  ⟨ Print the options for piece  $p$  rotated  $d$  times 5 ⟩ ≡
{
  switch ( $d$ ) {
  case 0:  $work[2] = 0, work[3] = 2;$ 
     $work[4] = piece[p][0], work[5] = piece[p][1];$ 
     $work[6] = piece[p][2], work[7] = piece[p][3];$ 
    break;
  case 1:  $work[2] = 2, work[3] = 0;$ 
     $work[4] = piece[p][1], work[5] = -piece[p][0];$ 
     $work[6] = piece[p][3], work[7] = -piece[p][2];$ 
    break;
  case 2:  $work[2] = 0, work[3] = -2;$ 
     $work[4] = -piece[p][0], work[5] = -piece[p][1];$ 
     $work[6] = -piece[p][2], work[7] = -piece[p][3];$ 
    break;
  case 3:  $work[2] = -2, work[3] = 0;$ 
     $work[4] = -piece[p][1], work[5] = piece[p][0];$ 
     $work[6] = -piece[p][3], work[7] = piece[p][2];$ 
    break;
  }
   $xmin = xmax = ymin = ymax = 0;$ 
  for ( $k = 2; k < 8; k += 2$ ) {
    if ( $work[k] < xmin$ )  $xmin = work[k];$ 
    if ( $work[k] > xmax$ )  $xmax = work[k];$ 
    if ( $work[k+1] < ymin$ )  $ymin = work[k+1];$ 
    if ( $work[k+1] > ymax$ )  $ymax = work[k+1];$ 
  }
  for ( $x = (1 - xmin) | 1; x + xmax < n + n; x += 2$ )
    for ( $y = (1 - ymin) | 1; y + ymax < m + m; y += 2$ ) {
      printf ("%d_%c%c%c%c%c%c",  $p$ ,  $encode(x + work[0])$ ,  $encode(y + work[1])$ ,
         $encode(x + work[2])$ ,  $encode(y + work[3])$ ,  $encode(x + work[4])$ ,  $encode(y + work[5])$ ,
         $encode(x + work[6])$ ,  $encode(y + work[7])$ );
      printf ("\n");
    }
}

```

This code is used in section 1.

**6. Index.**

*argc*: [1](#), [2](#).  
*argv*: [1](#), [2](#).  
*d*: [1](#).  
*encode*: [3](#), [4](#), [5](#).  
*exit*: [2](#).  
*fprintf*: [2](#).  
*k*: [1](#).  
*m*: [1](#).  
*main*: [1](#).  
*n*: [1](#).  
*p*: [1](#).  
*piece*: [1](#), [5](#).  
*printf*: [2](#), [4](#), [5](#).  
*sscanf*: [2](#).  
*stderr*: [2](#).  
*work*: [1](#), [5](#).  
*x*: [1](#), [3](#).  
*xmax*: [1](#), [5](#).  
*xmin*: [1](#), [5](#).  
*y*: [1](#).  
*ymax*: [1](#), [5](#).  
*ymin*: [1](#), [5](#).

- ⟨ Print the item-name line 4 ⟩    Used in section 1.
- ⟨ Print the options for piece  $p$  rotated  $d$  times 5 ⟩    Used in section 1.
- ⟨ Process the command line 2 ⟩    Used in section 1.
- ⟨ Subroutine 3 ⟩    Used in section 1.

WINDMILL-DOMINOES-DLX

	Section	Page
Intro .....	<a href="#">1</a>	1
Index .....	<a href="#">6</a>	4