CS 1713, Assignment 7, Spring 1998
Due March 12, 1998
Chutes and Ladders

The Game. This assignment simulates the play of a popular children's game known as *Chutes and Ladders*, with a board as shown below. The board has 100 squares, numbered 1 through 100. Players start just before the first square and roll a single die (1 through 6 spots, all 6 equally likely) at each play. A player's token is moved forward by the number of spots on the die. After moving, the new square is examined for a *ladder* leading up to a higher-numbered square, and if one is present the player proceeds up the ladder. A square might also have the top of a *chute* in it, leading to a lower-numbered square, and in this case the player "slides" down the chute after arriving at the square. Arriving either at the top of a ladder or at the bottom of a chute has no significance. The game ends when a player first reaches spare 100. You are not permitted to move to square 100 if the current position plus the number of spots is greater than 100. (In that case you lose a turn.)

The Simulation. You will write a C program that pretends to be one person playing this game. *The objective of the simulation is to find a way to finish the game in the minimum number of rolls (6) that involves a chute.* (This may take several experimental computer runs. There are at least 7 essentially different ways to finish in 6 moves that involve different ladders and chutes. Note that there are usually many ways to go from the top of one ladder to the bottom of another, and these don't count as different.)

The simulation can conveniently use an array with integer values giving the new square to proceed to in case of a chute or ladder. If no chute or ladder starts at a location, the array will hold a zero value. Assuming the array is named `s`, these values are stored in locations `s[1]` through `s[100]`. (Location `s[0]` will be filled in with a `0`, but will not be used.) On runner, the file `assign7.data` in the directory `~wagner/pub/CS1713/fall97` contains the data needed to initialize the array `s` described above:

```
 0
38   0   0  14   0   0   0   0  31    0
 0   0   0   0   0   6   0   0   0    0
42   0   0   0   0   0   0  84   0    0
 0   0   0   0   0  44   0   0   0    0
 0   0   0   0   0   0  26   0  11    0
67   0   0   0   0  53   0   0   0    0
 0  19   0  60   0   0   0   0   0    0
91   0   0   0   0   0   0   0   0  100
 0   0   0   0   0   0  24   0   0    0
 0   0  73   0  75   0   0  78   0    0
```

(Notice that `0` is read for location `s[0]`, even though it is not needed.) You open this file with

```
FILE *table;
table = fopen("/home/faculty/wagner/pub/CS1713/fall97/assign7.data", "r");
if (table == NULL) { printf("Couldn't open data file\n"); exit(1); }
```

and read the ith array entry with

```
fscanf(table, "%d", &s[i]);
```

The file `assign7.initdata` in the same directory above contains an initialization for the array `s` as an alternative method.

```
int s[101] =
/*  C   O   L   U   M   N   S                */
/*  1   2   3   4   5   6   7   8   9   10   ITEM = ROW*10 + COL */
 {  0,
   38,  0,  0, 14,  0,  0,  0,  0, 31,   0,  /* ROW 0 */
    0,  0,  0,  0,  0,  6,  0,  0,  0,   0,  /* ROW 1 */
   42,  0,  0,  0,  0,  0,  0, 84,  0,   0,  /* ROW 2 */
    0,  0,  0,  0,  0, 44,  0,  0,  0,   0,  /* ROW 3 */
    0,  0,  0,  0,  0,  0, 26,  0, 11,   0,  /* ROW 4 */
   67,  0,  0,  0,  0, 53,  0,  0,  0,   0,  /* ROW 5 */
    0, 19,  0, 60,  0,  0,  0,  0,  0,   0,  /* ROW 6 */
   91,  0,  0,  0,  0,  0,  0,  0,  0, 100,  /* ROW 7 */
    0,  0,  0,  0,  0,  0, 24,  0,  0,   0,  /* ROW 8 */
    0,  0, 73,  0, 75,  0,  0, 78,  0,   0}; /* ROW 9 */
```

The Random Numbers. Random `double`s uniformly distributed between 0 and 1 can be generated using the functions `srand48()` to initialize the generator, and `drand48()` to return the next random

`double`. `srand48` is called once with any positive integer parameter value, and then `drand48()` is called any number of times with no parameter to return random `double`s. These functions (to be discussed in class) require the header `#include <stdlib.h>`. It is possible to initialize this generator with the value returned by the `time` function, so that it starts a different way each time it is called. The following code prints 5 random real numbers, a different 5 each time it is executed:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void main(void)
{
        int i;
        srand48((long)time(NULL));
        for (i = 0; i < 5; i++)
                printf("%.15f\n", drand48());
}
```

The following function returns an integer between `i` and `j` inclusive:

```
int ranint(int i, int j)
{
     return ((int) ((j - i + 1)*drand48() + i));
}
```

What to turn in. You should turn in one run showing a single simulated game, printing each of the rolls. You should check your simulated game against the real game below to see that your program is playing the game correctly. Then you should turn in whatever runs led you to discover a mimumum game (6 moves) that involves at least one chute. (You must use the computer to discover this. There is no credit for finding it by hand.)