```
 1: /**************************************************************************
 2:  * helpers.c
 3:  *
 4:  * Doug Lloyd
 5:  * September 28, 2011
 6:  *
 7:  * Helper functions for Problem Set 3.
 8:  **************************************************************************/
 9:
10: #include <cs50.h>
11: #include "helpers.h"
12:
13:
14: /*
15:  * Returns true if value is in array of n values, else false.
16:  */
17:
18: bool search(int value, int array[], int n) {
19:   return binarysearch(value, array, 0, n-1);
20: }
21:
22: /*
23:  * Helper function for search() - recursively does binary search
24:  */
25:
26: bool binarysearch(int value, int array[], int start, int end) {
27:
28:   /* If this happens, element not in the array */
29:   if(start > end)
30:     return false;
31:
32:   /* Calculate the midpoint */
33:   int mid = (start + end) / 2;
34:
35:   /* If target is at the midpoint, we found it */
36:   if(array[mid] == value)
37:     return true;
38:
39:   /* If target is greater than midpoint, we need to search to the right */
40:   else if(array[mid] < value)
41:     return binarysearch(value, array, mid+1, end);
42:
43:   /* If target is less than the midpoint, we need to search to the left */
44:   else
45:     return binarysearch(value, array, start, mid-1);
46: }
47:
48: /*
49:  * Sorts array of n values.
50:  */
51:
52: void sort(int values[], int n) {
53:
54:   /* Implementing selection sort */
55:   for(int i = 0; i < n; i++) {
56:
57:     /* Set the index of the smallest element to be the first element */
58:     int smallest = i;
59:     int j;
60:     for(j = i; j < n; j++) {
61:
62:       /* If we find a smaller element, set smallest to reference that one */
63:       if(values[j] < values[smallest])
64:         smallest = j;
```

```
65:     }
66:
67:     /* After searching unsorted portion, swap current element with smallest */
68:     swap(&values[i], &values[smallest]);
69:   }
70:
71:   return;
72: }
73:
74: /*
75:  * Swapping function
76:  */
77:
78: void swap(int *a, int *b) {
79:   /* Same as a regular swap, except we have pointers, so we're swapping the
80:      actual values, not copies of them */
81:   int tmp = *a;
82:   *a = *b;
83:   *b = tmp;
84:   return;
85: }
```

```
 1: /***************************************************************************
 2:  * fifteen.c
 3:  *
 4:  * Doug Lloyd
 5:  * September 28, 2011
 6:  *
 7:  * Implements The Game of Fifteen (generalized to d x d).
 8:  *
 9:  * Usage: fifteen d
10:  * whereby the board's dimensions are to be d x d,
11:  * where d must be in [DIM_MIN,DIM_MAX]
12:  ***************************************************************************/
13:
14: #define _XOPEN_SOURCE 500
15:
16: // header files
17: #include <cs50.h>
18: #include <stdio.h>
19: #include <stdlib.h>
20: #include <unistd.h>
21: #include <math.h>
22:
23: // constants
24: #define DIM_MIN 3
25: #define DIM_MAX 9
26: #define BLANK 0
27:
28: // board
29: int board[DIM_MAX][DIM_MAX];
30:
31: // dimensions
32: int d;
33:
34:
35: // prototypes
36: void clear(void);
37: void greet(void);
38: void init(void);
39: void draw(void);
40: bool move(int tile);
41: int dist(int x1, int y1, int x2, int y2);
42: bool won(void);
43: void swap(int *a, int *b);
44:
45: int main(int argc, char **argv) {
46:     // greet user with instructions
47:     greet();
48:
49:     // ensure proper usage
50:     if (argc != 2) {
51:         printf("Usage: %s d\n", argv[0]);
52:         return 1;
53:     }
54:
55:     // ensure valid dimensions
56:     d = atoi(argv[1]);
57:     if (d < DIM_MIN || d > DIM_MAX) {
58:       printf("Board must be between %d x %d and %d x %d, inclusive.\n",
59:              DIM_MIN, DIM_MIN, DIM_MAX, DIM_MAX);
60:       return 2;
61:     }
62:
63:     // initialize the board
64:     init();
```

```
 65:
 66:     // accept moves until game is won
 67:     while (true) {
 68:       // clear the screen
 69:       clear();
 70:
 71:       // draw the current state of the board
 72:       draw();
 73:
 74:       // check for win
 75:       if (won()) {
 76:         printf("ftw!\n");
 77:         break;
 78:       }
 79:
 80:       // prompt for move
 81:       printf("Tile to move: ");
 82:       int tile = GetInt();
 83:
 84:       // move if possible, else report illegality
 85:       if (!move(tile)) {
 86:         printf("\nIllegal move.\n");
 87:         usleep(50000);
 88:       }
 89:
 90:       // sleep thread for animation's sake
 91:       usleep(50000);
 92:     }
 93:
 94:     // that's all folks
 95:     return 0;
 96: }
 97:
 98: /*
 99:  * Clears screen using ANSI escape sequences.
100:  */
101:
102: void clear(void) {
103:     printf("\033[2J");
104:     printf("\033[%d;%dH", 0, 0);
105: }
106:
107: /*
108:  * Greets player.
109:  */
110:
111: void greet(void) {
112:     clear();
113:     printf("WELCOME TO THE GAME OF FIFTEEN\n");
114:     usleep(500000);
115: }
116:
117: /*
118:  * Initializes the game's board with tiles numbered 1 through d*d - 1
119:  * (i.e., fills 2D array with values but does not actually print them).
120:  */
121:
122: void init(void) {
123:
124:   // General case
125:   int sqnum = (d*d)-1;
126:   for(int i = 0; i < d; i++)
127:     for(int j = 0; j < d; j++, sqnum--)
128:       board[i][j] = sqnum;
```

```
129:
130:   // Special case for even-numbered boards
131:   if(!(d%2))
132:     swap(&board[d-1][d-2], &board[d-1][d-3]);
133:
134:   // All done
135:   return;
136: }
137:
138: /*
139:  * Prints the board in its current state.
140:  */
141:
142: void draw(void) {
143:
144:   // cycle through cells, print numbers if nonzero
145:   for(int i = 0; i < d; i++) {
146:     for(int j = 0; j < d; j++)
147:       (board[i][j] == BLANK) ? printf("  _") : printf("%3d", board[i][j]);
148:     printf("\n\n");
149:   }
150:   return;
151: }
152:
153: /*
154:  * If tile borders empty space, moves tile and returns true, else
155:  * returns false.
156:  */
157:
158: bool move(int tile) {
159:
160:   // Find tile location and blank location
161:   int tilerow, tilecol, blankrow, blankcol;
162:   for(int i = 0; i < d; i++)
163:     for(int j = 0; j < d; j++) {
164:       if(board[i][j] == tile) {
165:         tilerow = i;
166:         tilecol = j;
167:       }
168:       if(board[i][j] == BLANK) {
169:         blankrow = i;
170:         blankcol = j;
171:       }
172:     }
173:
174:   // Calculate distance between them; if dist == 1 then the squares are
175:   // adjacent and can be swapped. Otherwise, they can't.
176:
177:   if(dist(tilerow, tilecol, blankrow, blankcol) == 1) {
178:     swap(&board[tilerow][tilecol], &board[blankrow][blankcol]);
179:     return true;
180:   }
181:   return false;
182: }
183:
184: /*
185:  * Calculates the distance (taxicab geometry) between two tiles
186:  */
187:
188: int dist(int x1, int y1, int x2, int y2) {
189:   return abs((x1-x2) + (y1-y2));
190: }
191:
192: /*
```

```
193:  * Returns true if game is won (i.e., board is in winning configuration),
194:  * else false.
195:  */
196:
197: bool won(void) {
198:
199:   // Cycle through, checking to see if cells are in order
200:   int winnum = 1;
201:   for(int i = 0; i < d; i++)
202:     for(int j = 0; j < d; j++, winnum++)
203:       if(board[i][j] != (winnum % (d * d)))
204:         return false;
205:   return true;
206: }
207:
208: /*
209:  * Swaps two numbers given the pointers. Used in init() and move()
210:  */
211:
212: void swap(int *a, int *b) {
213:   int tmp = *a;
214:   *a = *b;
215:   *b = tmp;
216:   return;
217: }
```