Writing and Debugging Your Program

It is hard for anyone to look at someone else's code and determine why it is not working. TA's and the instructor will not find bugs for you.

*** Approach the process of writing the program in a way that allows you to resolve bugs; write your code a piece at a time.

Maybe something like this:

• From Assignment P0; it is assumed that you are already familiar with how to edit, compile, and run a C program, and familiar with pointers.

  *Don't start this assignment before completing Assignment P0.*

• Make a function ERROR that takes as an argument a character string, prints that string, prints a few other key values (current size of queue, etc.) and then exits the program. So now you can call ERROR anywhere to check for things that should not be happening (e.g., removing an item from an empty queue).

• Add a type declaration for a position structure, create position structures containing the start and final arrays, write a function to print a position (that takes as an argument a pointer to a position structure and creates the arguments to make a call to the function to print a position), and then test it on the pointers to the start and final position structures.
• Add declarations to define the hash table, write a hash function, write a function to initialize the hash table, and write the functions that you will need to search for and insert positions. Then write a small test function to insert the start and end positions and maybe a few others and then print the contents of the hash table.

• Add declarations to define the queue, write a function to initialize the queue, and write the ENQUEUE and DEQUEUE functions. Write a function to print the positions currently in the queue. Then write a small test function to do some combinations of queue operations and printing of the queue to test that the queue is working.

• Write a function to generate reachable positions from the current position and test it.

• Write a function to print a solution. If you find the idea of reversing pointers confusing, get everything working with a simpler method first (using a stack or a recursive function as described in the questions), and then work on the full credit method.

• Write your main loop and if things don't work, start by testing it on a very small problem such as changing your final position to be one move away from the start position (with ENQUEUE, DEQUEUE, etc. printing positions as they are encountered).

• Clean up your program, put in good documentation throughout, and make a good high level description of how your program works for a large header comment.

• Answer the questions.
A Suggested Program Structure

*** Ideally your program might be a collection of files, but for grading purposes your program must be a single file. Start each section with white space and a substantial header comment that describes the section.

Constants
   Start position, end position, hash table size, queue size, etc.

Puzzle board manipulation functions:
   Make possible moves (store in temporary arrays).
   Provide a function to compare two boards.

Define a position type.

Hash table and its methods.

Queue and its methods.

Functions to output a solution and associated statistics.

Main Program:
   Implement the pseudo-code given in the assignment.
Example - a position could be a structure consisting of:

- hash table bucket link
- back pointer to the preceding position
- cost of this position (an integer equal to the number of steps from start position)
- piece that moved to this position (represented as a character or board index)
- direction moved to this position (N, S, E, W, or 0, 1, 2, 3)
- a character string representing the current board (or could use a 2-dimensional array)
- possibly other information

/*Position data type.*/
typedef struct pnode {
    struct pnode *next; /*next pointer for the hash bucket*/
    struct pnode *back; /*position from which this one came*/
    int cost;            /*number of moves to get to this position*/
    char piece;          /*piece that moved to this position*/
    int dir;             /*direction moved to enter this position*/
    char board[12];     /*this position's board*/
} PositionBody;

typedef PositionBody *TypePosition;
Example - allocating memory for a new position:

Allocation of a new position is similar to the allocation for a new linked list vertex in Assignment P0:

/*Allocate memory for a new position.*/
TypePosition NEWPOSITION() {
    TypePosition p = (TypePosition) malloc(sizeof(PositionBody));
    if (p==NULL) {
        printf("Malloc for a new position failed.");
        exit(1);
    }
    return p;
}