Computer Science 1 - Program 1 (aka “Wake-Up Call”)
UCF KnightsBall Lottery
Assigned: 1/11/12
Due: 1/25/12 (Wednesday) BY 11:55pm (WebCourses time)

Objective
1. To review reading from a file.
2. To use records (an instance of a struct)
3. To use **Dynamic Memory Allocation**
4. To create and use a dynamically allocated array of structs
5. To use enumerated types in a useful manner (more info given on Discussion board!)

The Problem
Bad economic times has forced the UCF administration to think outside the box for alternative methods of income. Specifically, we now have a UCF lottery, KnightsBall. Your have been hired by UCF to decide who gets winnings for the KnightsBall Lottery. UCF has a system that automatically creates a file that contains the name of each ticket purchaser along with the combination of 6 ***distinct*** numbers picked by that person. Your job is to use this file in determining everyone's winnings. In particular, here is the payout for matching a certain number of values:

Numbers Matched	Winnings
3	$10
4	$100
5	$10000
6	$1000000

Your program will automatically read in an input file with the data for the ticket purchases. Then your program will ask the user for the winning combination of numbers. (When playing the KnightsBall lottery, you must pick 6 ***distinct*** numbers from the set {1,2,3,...,52,53}.) So 53 is the highest possible number.

The user MUST enter these numbers in ascending order. Once these have been entered, your program should print out the names of all the winners, along with how much money they have won. You are guaranteed that each person playing the lottery has bought EXACTLY one ticket and is listed once in the file.

In your implementation please adhere to the following guidelines:

1) Use a record (struct) to store information about each person/ticket.
2) **Dynamically allocate** an array of these structs to store all the ticket information based on the first number in the file, which represents the number of tickets purchased.
3) Use an enumerated type for the four possible winning values.

<u>Implementation</u>:

You MUST use the EXACT struct as follows (notice the typedef):

```
typedef struct KnightsBallLottoPlayer {
     char firstName[20];
     char lastName[20];
     int numbers[6]
} KBLottoPlayer;
```

<u>Helpful Reminder</u>: As mentioned in class, and on the notes, this struct is simply a "skeleton". When you type this into your program, the compiler understands that, at some point in the program, you may reference something called KBLottoPlayer. And when you do reference KBLottoPlayer, the compiler will remember that you made this skeleton and will then know what you are talking about. You will then be able to make (dynamically allocate) space for an instance of one KBLottoPlayer, or as required for this program, an array of these structures.

<u>Note</u>: Since this is typedef'ed, you can simply use "`KBLottoPlayer`" instead of "`struct KnightsBallLottoPlayer`" when referencing this structure throughout the program.

<u>Input File Specification (KnightsBall.in)</u>
You will read in input from a file, " KnightsBall.in". The name MUST BE "KnightsBall.in". Have this AUTOMATED. Do not ask the user to enter "KnightsBall.in". You should read in this automatically. (This will expedite the grading process.)

** If there is confusion over the *.in file extension, or the *.out file extension (see Output section below), refer to the discussion boards for my explanation.*

The input file your program will take in will have the following format:

The first line will contain a single integer n, the total number of tickets bought. Since you are guaranteed that each person playing purchased only one ticket, this integer n also represents the number of lottery players.

The next 2n lines will contain information about each ticket purchased. The information for a single ticket will be on <u>two lines</u>. The first of the two lines will contain the last name of the ticket purchaser, followed by a space, followed by the first name of the ticket purchaser. Both the first and last name are guaranteed to be 19 characters or less. The following line will contain the six integers chosen by that buyer, <u>all in ascending order</u>, separated by spaces.

<u>User Input Specification</u>

1. The winning lottery numbers will be valid and entered in ascending order.

Output Format
*****NOTE***: You should generate your output to a FILE that you will call "KnightsBall.out".**

You will print to the screen requesting the user to input the 6 winning numbers. However, all remaining output is printed to the file.

Your output should contain one winner per line. Each line of output should be of the following format:

```
First Last matched X numbers and won $Y.
```

where First is the first name of the winner, Last is the last name of the winner, X is the number of winning numbers the player picked correctly, and Y is the prize money won.

*****WARNING*****
Your program MUST adhere to this EXACT format (spacing capitalization, use of dollar signs, periods, punctuation, etc). The graders will use very large input files, resulting in very large output files. As such, the graders will use text comparison programs to compare your output to the correct output. If, for example, you have two spaces between in the output when there should be only one space, this will show up as an error even through you may have the program correct. You WILL get points off if this is the case, which is why this is being explained in detail. Minimum deduction will be 10% of the grade, as the graders will be forced to go to text editing of your program in order to give you an accurate grade.

Again, your output MUST ADHERE EXACTLY to the sample output shown below.

Input & Output Samples
Here is a sample input file:

```
5
Llewellyn Mark
1 15 19 26 33 46
Young Brian
17 19 33 34 46 47
Cazalas Jonathan
1 4 9 16 25 36
Siu Max
17 19 34 46 47 48
Balci Murat
5 10 17 19 34 47
```

Note that this is NOT a comprehensive test. You should test your program with different data than is shown here based on the specifications given. The user input is given in *italics* while the program output is in bold. (Note: The following output is based upon the sample input file shown above.)

```
Enter the winning Lottery numbers:
17 19 33 34 46 47
```

Output File: (saved to "KnightsBall.out":

```
Mark Llewellyn matched 3 numbers and won $10.
Brian Young matched 6 numbers and won $1000000.
Max Siu matched 5 numbers and won $10000.
Murat Balci matched 4 numbers and won $100.
```

Grading Details

Your program will be graded upon the following criteria:

1) Adhering to the implementation specifications listed on the first page.
2) Your algorithmic design.
3) Correctness.
4) **Use of Dynamic Memory Allocation. If your program does NOT use DMA, you will NOT get credit for the assignment.**
5) The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (We're not kidding here. If your code is poorly commented and spaced and works perfectly, you could earn as low as 80-85% on it.)
6) Compatibility to CodeBlocks (If your program does not compile in CodeBlocks, you will get a sizable deduction from your grade.)
7) Your output MUST adhere to the EXACT output format shown above.

Bonus Points Opportunity:

Note: There is a more efficient way to determine the number of matching tickets compared to the straightforward method of doing two for loops. This method will be discussed during the first week of class and is available on the class notes for that week. If you implement the quicker method, you get an extra 10 bonus points.

Restrictions

Name the file you create and turn in *KnightsBall.c*. Although you may use other compilers, your program must compile and run using CodeBlocks. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. You should also include comments throughout your code, when appropriate.

Deliverables

A single source file named *KnightsBall.c* turned in through WebCourses.