12.6 Practice Programs

Sample solutions to these problems will be included by the beginning of 2013 at the following website:

http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3223/book/

1) Adjust the hotel program so that the struct stores how many people are staying in each room. When asking a party to check in, ask them how many people are in the party, and put all of the people in the party in the next available room. Add a function that calculates the total number of guests staying at the hotel and add an option to the menu to print out the total number of current guests.

2) Adjust the program from #1 so that each room has a capacity. (You can set these to whatever values you feel are appropriate.) Now, when a party checks in, you must check them into a single room with enough capacity to store the whole party. If no such room exists, the party can't stay at the hotel.

3) Create a card struct, to store the suit and kind of a standard playing card. Then, create an array of size 52 of type card struct, and populate that array with one of each type of playing card. Create an appropriate print function that prints out a single playing card and have your program print out an entire deck of cards.

4) Building off the program for #3, create a second struct that stores a collection of up to 52 cards. This struct should have an array of size 52 and an integer that keeps track of how many cards are actually in the collection. Create functions for adding and deleting a card from this struct. Write a main function that tests these functions out.

5) Building off the program for #4, write a function that populates a full deck of cards appropriately in order and write a second function that shuffles a fully populated deck of cards. Test your program by calling the two functions in succession, and then printing out the full deck of cards.

6) Building off the program for #5, write a blackjack program that allows the dealer to start with a full deck and deal two random cards from the deck to two players each. To simplify the game, let Aces always count as 11 points. Continue to ask each user whether or not they want another card and deal them another card until they say no. Let user 1 get all of her extra cards before user 2 starts to get his. Report the winner of the game.

7) Edit the program for #6 so that when users have an Ace, it is automatically awarded 11 points unless doing so would force them to bust. In the latter case, set the offending Ace to 1 point. Namely, for all users with Aces, attempt to give them the greatest score less than or equal to 21. If this is impossible, even with all Aces counting as 1 point, the user will then bust.

8) Create a struct money that stores a dollar amount and a cent amount, both in integers. Write functions to facilitate money transactions, such as adding and subtracting two money structs. Test these functions by writing an appropriate main function.

9) Create a struct to store information about professional basketball players. Use a point system that "scores" a basketball player, based on his points, rebounds, assists, blocks and steals. Read in data from a text file and then sort the players in order, by score totals. One possible scoring scheme is as follows: each point is worth 1, each rebound and assist are worth 2 and each block and steal are worth 5. Write the output of sorted players to a file. Design the file format, etc. on your own.

10) Write a program that utilizes the employee struct from the first section of the chapter. Read in data from a file about several employees into an array of struct employee. Sort the data by hourly pay rates, from largest to smallest. If two employees have the exact same pay rate, sort them by last name in alphabetical order. If the employees have the same last name, break the tie by first name. No two employees will have the same first and last name. The input file format is as follows:

The first line of the input file will contain a single positive integer, *n (n < 100)*, storing the number of employees. The next *n* lines will be of the following format:

```
FIRST LAST ID PAY
```

where `FIRST` is the first name of the employee, comprised of only uppercase letters, `LAST` is the last name of hte emplyee, comprised of only uppercase letters, `ID` is an integer in between 1000000 and 9999999, and `PAY` is the emplyee's hourly pay rate, a double in between 7.00 and 300.00.

Sample Input File
```
5
Ben Johnson 1234567 9.99
Anna Smith 2222222 10.35
Jennifer Smith 3213213 9.99
Adam Johnson 1234568 9.99
Jason Delaney 3333333 20.00
```

Sample Output File
```
Jason Delaney 3333333 20.00
Anna Smith 2222222 10.35
Adam Johnson 1234568 9.99
Ben Johnson 1234567 9.99
Jennifer Smith 3213213 9.99
```