CS2050 – Lab 8.5

Summer 2024

Concepts to Practice

- Recursion
- Makefiles

Submission Information

Submit this assignment by the mucsmake command to submit lab8.5.c:

```
mucsmake 2050 lab8.5 lab8.5.c
```

Key point: You cannot use any looping in any code that you write in this assignment. You may not use for, do while, or while. You may not use goto (didn't know that was a thing, right 😊?). You have got to do everything in this assignment using recursion! If you use a loop, you will not get credit!

Description

To get started on this lab, type the following while logged in to the hellbender.rnet.missouri.edu terminal:

```
cs2050start lab8.5
cd lab8.5
make
./a.out
```

For the lab assignment, you are to start with the starter code provided in the lab8.5 directory. This starter code is an outline of what you need to do. The only file you should change is lab8.5.c. Notice that you already have tests provided in lab8.5main.c this time (though, you can write additional tests if you want).

For full credit, all you have to do is write a function called mypow(). The mypow() function takes a base and raises that base to a power and returns that result. For example, 3 raised to the 2nd power is 9. Notice that a power may be zero or negative. Notice that a power may not be fractional (must be an integer). Check out the prototype for mypow() in lab8.5.h as well as the tests in lab8.5main.c.

Bonus

For extra credit, you can also implement PrintBaseBallPlayers() and SortBaseBallPlayers(). You should write PrintBaseBallPlayers() first and only try SortBaseBallPlayers() if you finish PrintBaseBallPlayers(). If you want to try these, you should use the provided functions in baseball.h.

PrintBaseBallPlayers() should print all of the players in the provided array as shown in the sample output. To print a single player, you can use the provided PrintBaseBallPlayer() function (see baseball.h).

SortBaseBallPlayers() should sort the provided array by batting average in descending order. You can use the AVG() function (see baseball.h) to get the batting average for a given player. Consider that you can sort an array by finding the largest item in the array (highest batting average in this case), placing it at the beginning of the array, and then sorting the rest of the array. This sketch of an algorithm is basically the same as "bubble sort". If you want to swap two players in the array, you can use the provided SwapPlayer() function (see baseball.h).

When I wrote SortBaseBallPlayers(), I found it convenient to write a FindHighestAvg() function first that returns a pointer to the player with the highest average. I called that function in my SortBaseBallPlayers() function. Don't forget: You can't use loops so . . . you might want both of these functions to be recursive!

Another tip on the bonus problems: You don't know the implementation of the BaseBallPlayer structure. So, the only way to get an array of sample players is to call the provided GetSamplePlayers() function (see baseball.h). You don't need to know the implementation of this structure because you have accessor functions (as we have discussed in class).

Notice:

1. All of your lab submissions must include documentation in the form of code comments to receive full points. In addition, your program is expected to have **a comment header** at the top that includes your name, pawprint, the course you are taking, and the lab that you solved.
2. All of your lab submissions must compile under GCC using the *−Wall* and *−Werror* flags to be considered for a grade (note that the Makefile should take care of this for you in this case).
3. Do **NOT** change any files other than lab8.5.c. Additional definitions/helper functions must be placed in your lab8.5.c file.

Rubric:

- mypow()
  - works properly for a power of zero – 2 points
  - works properly for positive powers – 3 points
  - works properly for negative powers – 3 points
  - works properly for negative bases – 2 points
- PrintBaseBallPlayers() – up to 2 points for printing all of the players in the array properly
- SortBaseBallPlayers() – up to 3 points for sorting all of the players in the array properly

Sample Output (output from just the starter code)

```
jer676@hellbender-login lab8.5]$ ./a.out
**** Regular Stuff:
2^-10=    0.000
2^ -9=    0.000
2^ -8=    0.000
2^ -7=    0.000
2^ -6=    0.000
2^ -5=    0.000
2^ -4=    0.000
2^ -3=    0.000
2^ -2=    0.000
2^ -1=    0.000
2^  0=    0.000
2^  1=    0.000
2^  2=    0.000
2^  3=    0.000
2^  4=    0.000
2^  5=    0.000
2^  6=    0.000
2^  7=    0.000
2^  8=    0.000
2^  9=    0.000
2^ 10=    0.000
-1^-2=    0.000
-1^-1=    0.000
-1^0=    0.000
-1^1=    0.000
-1^2=    0.000
-1^3=    0.000


**** Bonus Stuff:
```

Sample Output (completed code)

```
jimr@jimrsurfacepro9:~/CS2050/SS2024/labs/lab8.5$ ./a.out
**** Regular Stuff:
2^-10=    0.001
2^ -9=    0.002
2^ -8=    0.004
2^ -7=    0.008
2^ -6=    0.016
2^ -5=    0.031
2^ -4=    0.062
2^ -3=    0.125
2^ -2=    0.250
2^ -1=    0.500
2^  0=    1.000
2^  1=    2.000
2^  2=    4.000
2^  3=    8.000
2^  4=   16.000
2^  5=   32.000
2^  6=   64.000
2^  7=  128.000
2^  8=  256.000
2^  9=  512.000
2^ 10= 1024.000
-1^-2=    1.000
-1^-1=   -1.000
-1^0=     1.000
-1^1=    -1.000
-1^2=     1.000
-1^3=    -1.000


**** Bonus Stuff:
Paul Goldschmidt - 0.268 avg, 0.362 obp, 0.447 slug
Nolan Arenado - 0.266 avg, 0.316 obp, 0.459 slug
Tommy Edman - 0.248 avg, 0.300 obp, 0.399 slug
Wilson Contreras - 0.264 avg, 0.342 obp, 0.467 slug
Lars Nootbar - 0.261 avg, 0.367 obp, 0.418 slug
Jordan Walker - 0.276 avg, 0.335 obp, 0.445 slug
Nolan Gorman - 0.236 avg, 0.325 obp, 0.478 slug

Jordan Walker - 0.276 avg, 0.335 obp, 0.445 slug
Paul Goldschmidt - 0.268 avg, 0.362 obp, 0.447 slug
Nolan Arenado - 0.266 avg, 0.316 obp, 0.459 slug
Wilson Contreras - 0.264 avg, 0.342 obp, 0.467 slug
Lars Nootbar - 0.261 avg, 0.367 obp, 0.418 slug
Tommy Edman - 0.248 avg, 0.300 obp, 0.399 slug
Nolan Gorman - 0.236 avg, 0.325 obp, 0.478 slug
```