Lab 9 [Files]

In this workshop, we will conclude our development of a text based game "Code Quest!". This workshop focuses on providing the user a method of saving their game and resuming it later.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will be able to

- read data sequentially from a text file
- write data sequentially to a text file

SPECIFICATIONS

Write the following two structs which will contain the required information for ***items*** and ***players*** (same as previous lab). In addition, write the following two functions which will be used to save and load the game state:

`Struct Player`(used in previous lab)`:` This struct contains all the required data to represent a player's character. These components are: Level, Strength, Speed, Defense, Intelligence, Luck, Max HP, Current HP (all integers) and Name (string).

`Struct Item` (used in previous lab)`:` This struct contains all the required data to represent an inventory item. These components are: Item name (string), item type (int representing the effect of the item), item power (int representing the amount of change the item will produce).

```
int saveGame(struct Player player, int exp, int nextLvl, float days,
struct Item inventory[], int quantities[], int inventorySize)
```

This function will query the user for a filename to store information into. Once stored, the function attempts to open the file for writing, if it is unsuccessful an error message should print out and the function should end returning 0. If the file was successfully opened, the information received by the function should be written to the file one value at a time, separated by commons (structs should have each part printed out in series). Once the write is complete, the file should be closed.

```
int loadSaveGame(struct Player* player, int* exp, int* nextLvl, float*
days, struct Item inventory[], int quantities[], int inventorySize)
```

This function will query the user for a filename to read the save information from. Once the filename has been retrieved from the user, the function attempts to open the file for reading, if it is unsuccessful, an error message should print out and the function should end returning 0. If the file was successfully opened, the function should read the file's contents, identifying each individual value by the comma delimiter. The data read by this function should be loaded into the parameters that were passed into this function in the same order they were saved in the previous function.

Once the above code has been implemented, copy the following program which will test your code.

```
int main()
 {
 struct Player p1;
 strcpy(p1.name, "Tester");
 p1.str = 10;
 p1.def = 20;
 p1.spd = 30;
 p1.intl = 40;
 p1.lvl = 5;
 p1.mhp = 50;
 p1.hp = 15;
 p1.lck = 33;

 float days = 12;
 int nextLvl = 15;
 int exp = 24;

 struct Item it[2];
 strcpy(it[0].name, "item1");
 strcpy(it[1].name, "item2");

 it[0].type = 3;
 it[1].type = 2;

 it[0].power = 40;
 it[1].power = 20;

 int qnt[2] = {12,34};

 saveGame(p1, exp, nextLvl, days, it, qnt, 2);

 struct Player p2 = {0};
 struct Item i2[2] = {0};
 int q2[2] = {0};

 exp = days = nextLvl= 0;

 while(!loadSaveGame(&p2, &exp, &nextLvl, &days, i2, q, 2));

 printf("\n%s %d %d %d %d %d %d %d %d\n%s %d %d\n%s %d %d\
\n%d %d\n%d %d %.1f", p2.name, p2.str, p2.def, p2.intl, p2.spd,
p2.lck, p2.lvl, p2.hp, p2.mhp, i2[0].name, i2[0].type,
i2[0].power, i2[1].name, i2[1].type, i2[1].power, q2[0],
q2[1], exp, nextLvl, days);
 }
```

The output of a typical run-through of your program should look like this (user input highlighted in green):

```
Enter filename to write: test.sav
Enter filename to read: herp.derp
Invalid save file!
Enter filename to read: test.sav
Tester 10 20 40 30 33 5 15 50
item1 3 40
item2 2 20
12 34
24 15 12.0
```

The file's output should be similar to this:

```
Tester,10,20,40,30,33,5,15,50,2,item1,3,40,item2,2,20,12,34,24,15,12.0
```

If your program's output exactly matches the output shown above, given the provided inputs then your lab is complete and ready to be submitted (read below).

BONUS:

BUILDING THE GAME (OPTIONAL)

If you have completed all previous game labs, you may complete this section in order to merge your current lab with you previous labs.

Finally we are going to complete codeQuest!

Copy the two functions from the lab and add them to the codeQuest source code (prototypes and implementation). Add a new option to the overworld menu to save the current game progress. If selected, the save function should be called, passing to it all relevant information. Print an appropriate message after saving.

Change the load game function from the opening menu to load the game data from the file specified by the user. If the data is successfully load, continue the game from the point it was saved from. This should allow the user to save a game at any time and resume it after the program has shut down.

Finally, to finish off the game, alter the demon battle function so that it accepts a pointer to the player instead of a pointer to the player's HP. The function should create a Player struct instance for the demon which should have very high stats (around 100 battle stats or so, it’s up to you). Call the battle sequence function passing in the user player and the demon player. If the user wins the battle, display a win screen to the user and end the game (go back to the main menu).

If you've followed all instructions up to this point, you should be done CodeQuest! Check the attachment to download a binary version of CodeQuest v1.0 to help you ensure your game works correctly!

SUBMISSION

Upload your solution according to your instructor’s guidelines.