Lab 9 [Structures]

In this workshop, we will continue our development of a text based game "Code Quest!". This workshop focuses on enhancing the game by adding strings for player and item names.

LEARNING OUTCOMES

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

- design a structure type
- access the members of an object of structure type
- pass an object of structure type to a function

SPECIFICATIONS

Write the following two structs which will contain the required information for ***items*** and ***players***. In addition, write the following three functions which will make use of these structs:

`Struct Player:` 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:` 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).

`struct Item loadItem():` This function defines a variable of type **`Item`**. The function, prompts the user for an item name, item type and item power, storing the values in the variable of type **`Item`** and returning it.

`struct Player loadPlayer():`This function defines a variable of type **`Player`**. The function, prompts the user for all required attributes of the defined variable (Level, Strength, Speed, Defense, Intelligence, Luck, Max HP, Current HP, and Name). This function then fills the defined variable with the input and returns the struct.

`void printData(struct Player p1, struct Item itemList[], int listSize):` This function takes a variable of type **`struct Player p1,`** an array of **`struct Item itemList[],`** and an integer **`listSize`** representing the number of items in **`itemList`**. This function displays all information for **`p1`** and all information for the list of items **`itemList`**. Check the output below for formatting.

Once the above code has been implemented, write a program that allows simple tests of the code. This program should allocate space for a single player `struct Player` and an array of size 2 of `struct Item`.

First, the `loadPlayer()` function is called to receive information for the player, the returned value is stored in the local `Player` struct.

Then, the `loadItem()` funciton is called twice, storing each returned value in a different part of the item array.

Finally, the `printData` function is called to display the information that was collected.

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

```
Please enter the following player info:
Name: Ariel
Level: 2
Strength: 20
Speed: 10
Defense: 10
Intelligence: 30
Luck: 10
Max HP: 100

Please enter the following Item info:
Name: Potion
Type: 3
Power: 10

Please enter the following Item info:
Name: Elixer
Type: 4
Power: 5

Collected Input:

Name   Lvl   Str   Spd   Def   Int   Lck   MHP
Ariel  2     20    10    10    30    10    100

Name   Type   Power
Potion 3      10
Elixer 4      5
```

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.

Copy the structures from this lab into your CodeQuest source code (the functions will not be needed). At the beginning of the game, instead of allocating separate variables for the player's stats, instead create a single player struct. Change all functions relating to the player to accept the player struct instead of its individual values. Change the inventory section from storing items in parallel arrays (quantity array is still needed) to storing them as a single array of item structs. Change all item related functions to use the item struct instead of individual values. Change the item resolution function so that it makes use of the item's power and type when resolving the item's effect (specify this as you'd like).

Finally change the battle sequence function so that it takes two player structs, one for the player and one for the enemy. The enemy should still be randomly generated as normal. When the battle or training finishes, check if the player's EXP has raised above a specified threshold (you may define this limit), if so, raise the level of the player and increase the player's stats by a certain amount (again, you define this). This should allow the player to become stronger the more battles and training they do.

SUBMISSION

Prepare a Typescript

Create a typescript on the remote Linux host using the following commands:

```
+ At the prompt, type: script w9.txt
+ At the prompt, type: whoami
+ At the prompt, type: cat w9.c
+ At the prompt, type: gcc w9.c -o w9.out
+ At the prompt, type: w9.out
+ At the prompt type: exit
```

This will produce a typescript named **"w9.txt"** in your current directory. Transfer a copy of this file to your local computer.

Upload your typescript file to BlackBoard:

- Login to BlackBoard
- Select your course code
- Select Workshop 9 under Workshops
- Upload **`w9.txt`**
- Write a short note to your instructor
    - Under “Add comments”, add a sentence or two regarding what you think you learned in this workshop in the notes textbox
    - press "Save Changes"
- When ready to submit, press "Submit". Note you can save a draft until you are ready to submit.