**Preface**

When HW2 is done, you will have a shop that you can purchase items from. You will then equip those purchased items to your chosen heroes, and continue to save these progressions. Much of the rest will be added flavor and features. The shop will utilize two separate linked lists that you will manage wrapped in one structure. See the .h file for more information.

Once again, read this entire document, and pay close attention to the comments section. You also should know by now that time management is crucial to complete these projects.

**NOTICE: There is a bonus opportunity in this homework that will affect how you implement your functions. See the grading criteria for more information.**

**Function Prototypes**

The following functions have been updated to accommodate the expansion:

* **playGame(…)**
* **loadSavedData(…)**
* **loadInitialData(…)**
* **printHeroes(…)**
* **saveFileData(…)**
* **loadGameStateData(…)**

If you are going to use your HW1, you will want to get the newer versions of these functions. You can either copy these functions into your version, or copy the HW1 functions you wrote into the template. You will find them in various .c files according to their scope at which they are used.

**Functions You Will Implement:**

**Shop\* loadShop()**

**Input: None**

**Return: The complete loaded linked lists wrapped in the shop, loaded from the input file**

You will be creating the shop and loading all of the items from the designated items file described below. The shop will hold two pointers: one for the head of each linked list. Load each shield in one at a time and add it to the list as normal. Then, when the shields are done, do the same for the swords.

(Continued on next page)

Return this created shop when you are done. **The number of swords and shields is given to you in the header file.** **You may hardcode the name of the item file. DO NOT INSERT THE NODES IN SORTED ORDER. INSERT THE NODES IN THE ORDER GIVEN IN THE FILE.**

**void destroyShop(Shop\*)**

**Input: The shop**

**Return: None**

The shop has the beginning of two linked lists. You are to free the linked lists, **then free the shop itself.** Recall the process for freeing a linked list. See notes below for a comment on this function.

**The functions below are paired together, as they are identical except for the variable types handled.**

**void printSwords(Sword\*)**

**void printShields(Shield\*)**

**Input: The head of the linked list**

**Return: None**

Print out the items in the given linked list. Make sure to print them out in a nice manner using menu options that the user can see and select from. **The menu numbers must behave the same way as the sample output shows, with numbers 1-N and the numbers staying aligned after an item is removed.**

**Sword\* removeSwordFromList(Sword\*\*, int)**

**Shield\* removeShieldFromList(Shield\*\*, int)**

**Input: A pointer to the head of the linked list, and the number of the selected item in the list**

**Return: The newly removed item (not the head of the list)**

Remove the selected item from the linked list, and return it. **Note the use of the double pointer here.** You must make sure that the list is never broken, regardless of the item that was chosen to remove.

**void sortShields(Shield\*\*)**

**void sortSwords(Sword\*\*)**

**Input: A pointer to the head of the linked list**

**Return: None**

By far the hardest two functions in the homework. You will be sorting the given linked list by price. This is up to you how you want to do this; you are not restricted based on what sort to use. Be careful when you swap nodes around, however!This implementation will require some thought.

**The following functions are given to you:**

**void freeHeroesAndItems(Hero\*)**

Frees the chosen heroes and their equipped items.

**void goShopping(Hero\*, Shop\*)**

The controller for the shopping experience. Calls other functions accordingly.

**int findSwordPrice(Sword\*, int)**

**int findShieldPrice(Shield\*, int)**

Finds an item’s price in the list, and returns it.

**File Format**

**Items.txt:**

<Name of shield 1> | <price of shield> | <defense value>

<Name of shield 2> | <price of shield> | <defense value>

<Name of shield 3> | <price of shield> | <defense value>

…

<Name of sword 1> | <price of sword> | <attack value>

<Name of sword 2> | <price of sword> | <attack value>

<Name of sword 3> | <price of sword> | <attack value>

…

**Items of Note**

* **All work you will do will go in shop.c and/or game.h. No other files need to be coded for this HW.** If you are going to use your code from HW1, make sure to separate out your functions into separate .c files as demonstrated in the template given to you. If you don’t follow the template exactly, you might get compile errors! If you are to make extra functions/structures, put the definitions or prototypes in the .h file.
* To compile your code you will run the command “make” in your project directory. See the sample output for a demonstration. Learn how the makefile works, even though you don’t have to write it. You will need to add to it for HW3.
* If there is a segmentation fault from code that you didn’t write, chances are you broke your linked list. Check your code again before contacting Daniel.
* Using GDB gets harder now that you have multiple files. Remember the debugging lab? The GDB commands get more relevant now.

**Sample Output**

See the PowerPoint slides for pictures and notes.

**Grading Breakdown**

Your code must compile with no warnings and no errors. Failure to do so will result in a **ZERO**, no exceptions. Any global variables will also result in a **ZERO**, no exceptions.

**ANY** segmentation fault or other runtime error will result in a minimum deduction of 50%. This will be increased at your TA’s discretion. **We will be attempting to break your code, so make sure your code handles any corner cases that come with working with linked lists.**

Just as with HW1, if you find a bug, let Daniel know immediately. You might receive extra credit for it.

**USE OF ARRAYS ANYWHERE IS PROHIBITED. ANY USE OF ARRAYS WILL RESULT IN A ZERO, NO EXCEPTIONS.**

**BONUS OPPORTUNITY: if you make your linked lists doubly linked (that is, use a previous pointer and a next pointer), you will receive 6 points extra credit. You are only eligible to get this extra credit if the rest of your program works perfectly. For this implementation, you must do the following:**

* Setup the linked lists with the previous pointers as well as the next pointers.
* Take the previous pointer into account when removing a node from the list or sorting. Make sure the list doesn’t break with regards to the previous pointers.
* Find a place in the homework to print out a message stating that you used both directional pointers in your HW.

**(Potentially) Useful statistics from Daniel’s solution (# of lines of code):**

* Printing: <10 lines of code per function
* Sorting: ~60 lines of code per function
* Destroying: ~20 lines of code
* Loading: ~60 lines of code
* Removing a node: <30 lines of code
* Total: ~250-300 lines of code
* Total with bonus: ~300-350 lines of code

If you are heavily over these numbers, you might want to rethink your design.

**20 points: loadShop() and destroyShop()**

* loadShop() correctly creates a linked list from the items in the input file in the order given
* The items are added in the order that they are read in from the input file
* destroyShop() frees each item in the given linked list correctly

**25 points: sortShields() and sortSwords()**

* The linked lists are sorted in place by price ascending
* The linked list stays intact throughout the sorting algorithm

**10 points: removeSwordFromList() and removeShieldFromList()**

* The correct item is removed from the list based off of the item chosen from the user
* The list stays intact regardless of the item’s position in the list

**5 points: printSwords() and printShields**

* printSwords() and printShields() prints the items in a neat manner according to the sample output, and according to the linked list passed in
* The output is dynamic to account for items being removed from the list
* The output has the correct options for each item that the user can choose from

(Continued on next page)

Failure to follow these guidelines will result in deductions (At your TA’s discretion):

* All prewritten code, prototypes and files given to you are unmodified (any tampering with prewritten code without Daniel’s consent = major point deductions). If you do not follow the prototypes you are told to use, you will receive a zero.
* Code is formatted in an acceptable and readable manner
* Comments are present explaining logical reasoning (HINT: more meaningful/thoughtful comments = more likely to get partial credit)