Week 10, Homework 08	Weight: 5%	Due: Monday week 12, 09:00am (via sync)

Pre-homework Preparation:

- **Lab: Week 10**
- **Lectures: Weeks 01, 02, 03, 04, 05, 06, 07, 08, 09, 10**

Homework Activities:

Obtain the homework files using **sync**. For each question's source file, be sure to fill in the file comment header accurately!

Remember, all work submitted must be unique and your own!

We are watching you! We are not *even* kidding.

Question 1: Video game library

Add code to **`hw08q01.c`** to declare a structure which describes an individual video game.

Video games have a title, genre, platform, developer, year of release, lower age limit, a price and whether or not they support in-app purchases. You will need to pick appropriate data types of each piece of information to be stored in the structure. Name the structure: **`Video_Game`**.

An example video game is as follows:

```
Title: Candy Crush Saga
Genre: Match-Three Puzzle
Developer: King
Year of Release: 2012
Platform: Android, iOS, Windows Phone
Lower Age Limit: 7
Price: $0.00
In-app Purchases: Yes
```

Another example video game is as follows:

```
Title: Halo 4
Genre: First-Person Shooter
Platform: Xbox 360, Xbox One
Developer: 343 Industries
Year of Release: 2014
Lower Age Limit: 16
Price: $69.95
In-app Purchases: No
```

Declare three video game structure variables locally in the **`main`** function, called **`game1`**, **`game2`** and **`game3`**.

Assign to the members of **`game1`**, the details for the Candy Crush Saga (King, 2012) example above.
Assign to the members of **`game2`**, the details for the Halo 4 (343 Industries, 2014) example above.
Assign to the members of **`game3`**, the details for your favour game… if you don't play games, check your smart phone… surely you play something on there… if not, check the relevant app store top charts and find a game that could become your new favourite!

Next, declare a function called **`print_video_game_details()`** which takes in a parameter that is a pointer to video game structure declared earlier. In this function, print out the details of the game passed to the function, in the style shown above for the Candy Crush Saga and Halo 4 examples.

Next, call the **`print_video_game_details`** function three times from **`main`**, passing in the address of **`game1`**, **`game2`** and **`game3`**, after the members have all been set.

Question 2: A linked list of int nodes

Add code to **hw08q02.c** to declare a **Node** structure which describes an individual node within a linked list. Each node will have a pointer to the next node in the linked list, and an **int** member which holds that data stored in the node.

```
struct Node
{
    Node* p_next;
    int data;
};
```

Declare a global variable, called **g_p_first**, which holds the pointer to the first node in the linked list. Set the pointer to zero to begin with. Place this declaration at the top of your **hw08q02.c** file. Below the include headers, but above all the functions.

```
struct Node* g_p_first = 0;
```

Next, add a prototype for the following **add_new_node** function below the global variables. Then define the function below the prototypes.

```
void add_new_node(int data)
{
    // Insert your code here...
}
```

You will need to add code to the **add_new_node** function such that it creates a new **Node** on the heap, and assigns the data passed into the function as the data member of the new node. The newly created node should become the new first node in the linked list. Ensure the pointers are correctly set, such that the new node points to the old first node, and the global first node is updated to be the newly created now. This will effectively add a new node to the head of the list each time **add_new_node** is called.

```
void remove_node(int data)
{
    // Remove the node which contains data...
}
```

Add a prototype and definition for the **remove_node** function as above. Next, write code such that the **remove_node** function searches the linked list for a node with data matching the data passed into the function. If found, remove the node from the linked list. This will require you to reassign the pointers in the linked list such that the node to be removed is no longer in the list. Also, ensure that the node to be removed is deallocated from the heap.

```
void print_all_nodes()
{
    // Iterate the list, print the contents of all nodes.
}
```

Next, add a prototype and definition for the **print_all_nodes** function. This function must traverse the linked list, from the first node, to the end, printing out each node's data, and what where it points to next. The output should be as follows:

```
[N1: data is 10] -> [N2: data is 20] -> [N3: data is 30] -> null
```

You may find it helpful to also print the addresses of the nodes in memory, and the addresses stored in each node's **p_next** pointer.

Below is a sample main function, which could be used for testing. This function makes calls to add new nodes, remove nodes, and prints out the contents of the linked list. You can modify your main as necessary for testing your implementation of **add_new_node**, **remove_node**, and **print_all_nodes**.

```
int main(int argc, char* argv[])
{
    printf("1) Print the nodes: \n");
    print_all_nodes();

    add_new_node(10);
    add_new_node(20);
    add_new_node(30);

    printf("2) Print the nodes: \n");
    print_all_nodes();

    add_new_node(40);
    add_new_node(50);
    add_new_node(60);
    add_new_node(15);
    add_new_node(25);
    add_new_node(35);

    printf("3) Print the nodes: \n");
    print_all_nodes();

    remove_node(20);

    printf("4) Print the nodes: \n");
    print_all_nodes();

    remove_node(50);
    remove_node(15);

    printf("5) Print the nodes: \n");
    print_all_nodes();

    return 0;
}
```

You will need to work incrementally. Compile and run your program often. Do small changes to your code and ensure each new feature added works. Be methodical in your testing approach. Use debug printing with `printf` to help ensure your logic functions correctly inside each function.

Question 3: A linked list with nodes that point to structures

In this exercise, we will combine the code from question 1 and question 2. Using Linux, concatenate **`hw8q01.c`** and **`hw8q02.c`** into a single file, **`hw8q03.c`** and save this the **`q03`** directory.

Open your **`hw8q03.c`** file in the editor, and remove the duplicate header and library includes. You will want to organise your prototypes such that they are near the top of the file, after the includes. Ensure all the includes are grouped together!

Modify the **`Node`** structure by renaming it to **`Game_Title_Node`**, and changing the data if stores from an **`int`** to a pointer to a **`Video_Game`** structure.

Change the **`add_new_node`** function such that it accepts a pointer to a **`Video_Game`**, and then have the function add the new title to the linked list based upon alphabetical order of the titles.

Modify the **`remove_node`** and **`print_all_nodes`** functions appropriately to now also work with **`Video_Game`** structures, instead of the **`int`** type.

Write a function called **`add_new_game_title_to_library()`** which queries the user for a new video game title which is to be added to the library. This function must allocate the new title on the heap, and populate its members based upon the users input. Users must be queried for a title, genre, platform, developer, year of release, lower age limit, a price and whether or not they support in-app purchases.

Modify the `main` function to present the user with a menu. The menu must present the following to the user:

```
Welcome to the "Video Game Library" System
==========================================

1) View game titles in library.
2) Add game title to the library.
3) Exit.

Please make a selection:
```

If the user selects 1, the program should output each game title, in a similar format to the `print_video_game_details` function from question 1.

If the user selects 2, the program should ask the user to enter details of the new game. Creating a new node, and setting its members based upon the user's desires.

If the user selects 3, the program should end.

Question 4: Write the linked list out to disk

Copy `hw08q03.c` from question 3 into the `q04` directory, and rename the file to `hw08q04.c`.

Add additional functionality to save and load the linked list of video games to a file on disk.

When the program starts, it should load the linked list from file on disk.

When the program exits, it should save the linked list to file on disk before returning to the operating system.

Homework Submission:

Run the `sync` command to submit your completed homework.

Marking Criteria:

Have you completed each of the following?

Marking Criteria:	**Week 10, Homework 08 Weight 5%**	**Maximum Possible Mark:**	**Mark achieved?**
Q1:	Video_Game structure accurately declared?	3	
	main function has three local Video_Game structure variables declared and initiliased correctly?	3	
	print_video_game_details function implemented?	3	
	main calls print_video_game_details three times passing in the local structure variables?	1	
Q2:	Node structure correctly declared?	1	
	g_p_first node declared globally and assigned zero?	1	
	add_new_node implemented correctly?	10	
	remove_node implemented correctly?	16	
	print_all_nodes implemented correctly?	12	
	Main function has test code that thoroughly experiments with the usage and functionality of adding, removing and printing nodes?	5	
Q3:	Game_Title_Node structure correctly declared?	1	
	add_new_node operates with Game_Tile_Node structures?	4	
	remove_node operates with Game_Tile_Node structures?	4	
	print_all_nodes operates with Game_Tile_Node structures?	4	
	add_new_game_title_to_library function queries user for a new game to add, and then adds the new game to the linked list?	6	
	When run, the program presents the user with a menu interface, allowing for adding, viewing and exiting?	6	
Q4:	Linked list is loaded from file upon running hw08q04 program?	10	
	Linked list is saved to file upon exiting the hw08q04 program?	10	
Total:		**100**	