CS 305: HW 1
Fall 2019
Due Date: Sunday September 15th, NLT midnight.

Free tutoring is available to you for CS 305. JIT and Tutoring help now available in SH208 or @slack.

Your code files (.h and .c files), sample output file, and short answer questions in a pdf or docx file should be submitted as a single zip file to Moodle. Name your write-up username _hw1.txt where username is your UP login.

This assignment is meant to give you practice with C programming and to compare/ contrast a C implementation with a Java implementation (namely, HW 0). READ INSTRUCTIONS CAREFULLY, THERE ARE SOME DIFFERENCES BETWEEN HW0 AND HW1.

This assignment should be completed **individually**. Advice: start as soon as the homework assignment is posted, so you have plenty of time to get the code to compile, have time to test, have time to debug, have time to re-test, and have time to complete the summary report.

Specification

You will implement a C version of HW0 (items and inventory that keeps tracks of items) for this assignment.

The final code should include the following five files:

- `item.h` – defines the inventory struct and the function prototypes that operate on the items (needs to be completed, included in starter code)
- `inventory.h` – defines the inventory struct and the function prototypes that operate on the inventory data structures (needs to be completed, included in starter code)
- `main.c` – test program (done, included in starter code)
- `inventory.c` – implements the functions (you must create the file)
    - create_inventory
    - add_item
    - sold item
    - calc_min_price
    - calc_max_price
    - calc_inv_value
    - delete_item
    - print_inventory
    - free_inventory
- `item.c` – implements the functions (you must create the file)
    - create_item
    - print

To compile a C program that has multiple files:

```
gcc -o main inventory.c item.c main.c
```

To run it:

```
main
(or)
./main
```

Practical Notes:

i. Comment out everything in the main.c:run_test(void) and start writing your code with the first line only that creates inventory. This way you only test tiny coding increments

```
inventory* rei = create_inventory(-3);
```

ii. After the above works, implement the `free_inventory` function and uncomment the very last line:

```
free_inventory(rei);
```

iii. Run your code (executable) through `valgrind` and make sure you deallocated all the memory you allocated to run the program. The easiest way to check is to run your program inside of a `valgrind` as such:

```
valgrind ./main
```

you will see some additional printout on the screen before the code executes, then the printout of your code, and at the end you should see the following if you deallocated all memory.

```
ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

iv. incrementally implement the rest of your code and incrementally test it by uncommenting the lines in the `run_test` function.

1. Open `item.h`. Below are the `struct` members. Complete the `struct` definition `item` (use `typedef` so the type item can be used in your code).

item
price: double
name : char[MAX_NAME_LENGTH + 1]
id_number : int

2. Open `inventory.h`. Below are the `struct` members for `inventory`. Complete the `struct` definition for `inventory` (use `typedef` so the type `inventory` can be used in your code). Note that `items` is a pointer to `item` (aka, an array of `item` objects).

inventory
items : item *
stock: int *
num_items : int
max_items : int

3. Open `main.c` to see how the code can be used and tested. Similar to HW0, the `run_test` function creates two inventories and adds and deletes items and prints the inventory items. The printouts include `item`'s price, name, id, stock and the additional store's inventory statistics: total inventory value, cheapest and most expensive item in the inventory.

4. Create a new file called `item.c`. At the top, put a useful comment that at least includes your name. Then, `include <stdio.h>`, "`item.h`", and "`inventory.h`". If you need to see a syntax example of how to include header files, see `main.c`.

Implement the following functions (can copy and paste function prototypes from item.h):

- `create_item`
  - declare a `item` object
  - set parameters `price`, `name` and `id_number` to the `item`'s `struct` members that make sense; negative item ID numbers and negative price do not make sense, so you should set default values that do make sense
  - set the first `MAX_NAME_LENGTH` chars in the parameter name to the item object name and ensure that the name of the item is exactly `MAX_NAME_LENGTH` chars long
    - For example, if `MAX_NAME_LENGTH` is 10 and the name passed in as the parameter is "`Warm Hat`", then the time's name should be "`Warm Hat  `" (with 2 extra spaces to make the name exactly 10 characters long).
    - For example, if the item name passed in is "`Warm Hat The Warmest `", then the stored name should be truncated to "`Warm Hat T`".
    - This ensures that the printing of the i`nventory` has a fixed width field for the `name`, so the printing of the items lines up nicely.
  - set the name at position `MAX_NAME_LENGTH` to '\0' (the null-terminating character) to end the string correctly in C
  - return the `item` object
- `print` (routine implemented in `item` code)
  - prints the `item` passed in as a parameter as follows:
    `Item  10.25 WarmFingers                    1     10`
    (see example output for format; it should be item's price, name, ID and stock count with tabs in between.
    Hint: you can use %.2f in C to state that you want fixed (2) digits of mantissa showing.

5. Create a new file called `inventory.c`. At the top, put a useful comment that at least includes your name. Then, include `<stdio.h>, <stdlib.h>, "item.h"`, and `"inventory.h"`.

Implement the following functions (can copy and paste function prototypes from `inventory.h`):

- `create_inventory`
  - create an `inventory` pointer variable `items` and allocate memory for the size of the `inventory` object
  - set `num_items` to 0
  - set `max_items` to `max_it` (if positive)

    - set `max_items` to `10` (if `max_it` <= 0)
    - allocate memory for the array of `max_items` of `item` objects. Assign the `items` pointer to the allocated memory.
    - create an `int` pointer `stock` and allocate memory for the `stock` array of ints `max_items` large.
    - return the pointer to the `inventory`
    - (note: since the object is a pointer to a `inventory`, use -> instead of . to access data members)
- `add_item`
    - if the number of items `num_items` in the inventory is >= `max_items`, then print "`Cannot add another item to inventory. Max inventory size exceeded`". Return -1.
    - Else, iff the item `it`'s `id_number` does not exist in the `inventory`, store the `item it` in position `num_items` of the array `items` and update `num_items` of the inventory by adding one. Return 0.
    - if the `item` already exists, don't add it, print error: "`Cannot add item to inventory. Duplicate item ID number`"
- `sold_item`
    - if item's id `item_id` is found in the store's `inventory`, decrement its `stock` count by one and print message that the item was sold
    - if the `item` is found, but the `stock` count is 0, print error: "`Cannot sell item that has 0 inventory count`" and return.
    - If the `item` is not found in the `inventory`, print error: "`Cannot sell item that is not in the inventory`"
- `delete_item`
    - if the `num_items` in the `inventory` is <= 0, print error "`Item number ___ not found. Store is empty.`" Return -1.
    - Look for the first occurrence of the `item_num` in the `inventory` that matches the item's `id_number`. If such an `item` is found, delete it and shift all items in the `inventory` after it (on the right) by one position to the left. Update the `num_items` to be one fewer and return 0. If such an `item` is not found, print error "`Item w/ number ___ not found in the inventory.`" and return -1.
    - Note there should not be more than one instance of an item in the inventory. See add_item specifications.
- `calc_[min, max]_price`
    - find the `item` with the [lowest, highest] `price` in the store's `inventory`
    - return the value found
- `calc_inv_value`
    - calculates the overall value of the `inventory` as a sum of (`item's price` * the `stock`) for all items in the inventory
- `print_inventory`
    - prints the `inventory` as `item`, `price`, `name`, `id`, and `stock` for the store's `inventory` passed in as `inventory* inv`. Use the `print` function for `item` objects.
    - after printing the store's items, print the `inventory's` value and the price of cheapest and most expensive price in the `inventory`.
- `free_inventory`
    - should take a pointer to a `inventory` as a parameter
    - free the store's → `items`
    - free the store's → `stock`

- free the `inventory` object

Error-checking
All methods should do proper error-checking of the parameters. If a parameter value does not make sense (for example, a negative price or a negative id), then the code should set an appropriate value such as 0 or 1.

Documentation
The code should be well-commented, well indented, and include enough whitespace to be readable. Each of the .c files should have a block comment at the top that includes information about the author and included code.

You may want to review the questions on the summary report before you start coding. One question asks you to list at least 2 compiler error messages that you get, so you may want to log those while you are working on the assignment.

Example Output
See `sample_output.txt` on moodle.

Additional Enrichment
If you have time, feel free to add the components to your program. But, keep the original code saved!!! You can submit multiple folders of code. If you do any enrichment, please put the original code files in a folder called hw1 and put the enriched program in a folder called hw1option. Then zip both folders into a single file.

Please document the additional features in your code and your written summary. This is not extra credit, but a chance for you to do some exploration.

- Add more attributes to items, such as “type” (grocery, garden, home, jewelry, etc.). You may need to create new run_test functions to account for these new attributes.
- If you have items with attributes, print the inventory items according to category or out of stock instead of the order that the items are stored in the inventory.
- Add other features to the inventory to make it more closely resemble a real inventory such as dates, store location, etc.

Logistics

1. Download the starter code located on Moodle. You may use any IDE or just a text editor (emacs or another one) to write your code. However, your code *must compile* with gcc on the Shiley’s Ubuntu VDI machine.
2. When you are finished and ready to submit:
    1. Create a new folder called username_HW1.
        1. (For example, Martin’s would be cenek_HW1.)
    2. Copy the 2 header files and 3 code files into the folder.
    3. Copy your username_hw1.txt file into this folder.
    4. Copy your sample_output.txt file into this folder.
    5. Zip the folder by right-clicking and Send To->Compressed Folder (on Windows). There should also be a zip utility on Mac or Linux.
3. **What to turn in:** Submit your **username_HW1.zip** file to Moodle before the due date and time. After logging into learning.up.edu, navigate to CS 305.

You will find a link to submit this file. You do not need to print anything for this homework submission.

Grading Guidelines (total of 20 points)
Your files will be graded in three separate categories:

- (5 points) Code Quality: Design, commenting, whitespace, readability, proper indentation, consistent naming conventions
- (10 points) Code Operation: Does code do what is listed in the specification? Note: code that does NOT compile will receive 0 points for code operation.
    - 1 point item.h
    - 1 point inventory.h
    - 2 points item.c (1 pt per function)
    - 6 points inventory.c (1 pt per function, 2 points for delete_item)
- (5 points) Summary report: Completeness, correctness, and clarity

HW 1 Report Guidelines and Format – use the template provided below. Include the questions in your write-up.

Name:
CS 305 HW 1 Report

1. Questions (include these in your write-up):
1a. (.5 point) If your program does not meet the specifications, please note the differences and anything that is not working correctly. If it works, write "Meets all specifications."

1b. (.5 point) Include the output file called `sample_output.txt` in your zip file from executing your program with the original run_test() function.

You can create an output file of what is printed to the screen with the command line command (assuming your executable is called myinventory):
`./myinventory > sample_output.txt`

1c. (1 point) Show test cases (such as run_test2, run_test3, etc.) that demonstrate how you conducted other test cases (such as negative number input error-handing, removing items from an empty inventory, creating items with names longer than 30 characters, etc.). For this question, you should label the test case (what condition you are testing), provide the test code, and provide the program output.

2. The functions that operate on inventories take the inventory object as a pointer to a inventory when passed as a parameter. This is because the inventory object is modified by some of the functions and we want those modifications to remain in place after the function is finished executing (this is sometimes referred to as "side effecting"). Note that in Java, all object types are passed by reference (pointer to the object), so the object state can be modified by methods. However, in C, you have the choice to pass objects directly (no pointer) or to pass objects via pointers. Passing objects directly without pointers means that any modifications to the object are lost when the function is done executing.

2a. Why are the functions that operate on item objects defined without using pointers to the items as parameters?

2b. Why are the functions that operate on inventory objects defined using pointers to the inventory objects as parameters? Which methods change the contents of the inventory struct?

3a. In the run_test function, you will see comments about drawing picture 1 and picture 2. Draw the picture of the `rei` object at the point where the comment point to draw picture 1 is located. Use arrows to indicate pointers. The picture should include all components of the `rei` object.

3b. Draw the picture of `rei` at the point of execution where draw picture 2 is located in run_test.

4a. Describe a difference between C and Java that you have observed by completing HW 0 and HW 1. (I already described that Java uses pass by reference for all object types and C uses pass by value, so that difference cannot be used here.)

4b. Describe an aspect that is the same in C and Java based on HW 0 and HW 1.

5a. What compile error messages did you receive when working on this assignment? For at least 2 compiler error messages you received, write the compiler error message, describe what the message means, and describe how you resolved it in a table below.

Compiler error message	What it means	How I resolved it

5b. How much time did you spend in total on this homework assignment (including the report)? Was this more or less time than you spent on HW 0? *(Learning to use a new programming language takes time, so it is not surprising if HW 1 took considerably more time than HW0.)*

Appendix A: (copy this statement if it applies to you) I verify that the code and this write-up were authored by me. I have documented the help I have received in comments in the code files.

Appendix B: Copy and paste your item.h, item.c, inventory.h, and inventory.c code files here (use Courier New 8pt font so the characters line up correctly)