EX2: Inventory Management System

Introduction:

The aim of this assignment is to practice implementing an inventory management system using linked lists and dynamically allocated data structures in programming. You are tasked with implementing several functions as requested in the assignment, while also having the freedom to define additional functions as needed. The provided skeleton file (***main_template.c)*** contains some of the necessary functions you must implement, including a main function that orchestrates the program's operation. It is crucial not to alter the main function or the function signatures provided.

Task goal

The goal in the task is to associate items to different warehouses. Each of the items and warehouses has a textual name and a positive numerical code. The code is unique for each item and each warehouse (the code may be the same for the item and the warehouse). There may be several different warehouses that store the same item (same name and same code). Conversely, there may be several different items stored in the same warehouse.

For the benefit of the program the basic structure of the main function is given as well as the necessary structures. In addition, there are a number of functions that you must not change and that <u>you must use</u>.

DEBUGON variable

In addition, for the sake of convenience, a ***DEBUGON*** variable has been defined which, if it is defined in the program, will print debugging printouts, some of which have been written for you, and you can similarly add your own printouts. Please note, when submitting, you must include the definition of the ***DEBUGON*** variable in the comment, thus blocking any printing that is not according to the task definition.

Printout:

In any case of an error, you must call the ***print_error_message*** function. This function can receive an arbitrary number of error codes that you can choose for debugging purposes. But pay attention to submission, when ***DEBUGON*** is not define the function will only print one fixed error message. (on submission you must to remark out the ***DEBUGON*** – undefine it)

All printout format must be the same as in the given example (see the ***output02.txt*** file).

Structures (see the given template file ***main_template.c***):

This assignment revolves around managing inventory items. There are 4 structures involved, representing information about items and warehouses:

```c
/* structures */
typedef struct item {
    char* name;
    int  id;
    struct wlst* warehouses;
} item;

typedef struct warehouse {
    char* location;
    int  code;
    struct itemlst* items;
} warehouse;

typedef struct itemlst {
    item* data;
    struct itemlst* next;
} itemlst;

typedef struct wlst {
    warehouse* data;
    struct wlst* next;
} wlst;
```

Basic Operations (see the given template file ***main_template.c***):

You will implement the following basic operations:

i - new item (if the id exist it will return the same item without any changes)
if the code of the item already exists, it will not generate it or rename it, it will simply use the existing one as is.

w - new warehouse (if the id exist it will return the same item without any changes)
if the code of the item warehouse exists, it will not generate it or rename it, it will simply use the existing one as is.

a - assign an item to a warehouse

u - unassign an item from a warehouse(do not delete the item or the warehouse!)

p - print status (see output.txt)

g - generating and assigning 100 items to 10 warehouses

1. 100 items must be produced with an ID code from 0 to 100 (not including 100)
2. 10 warehouses with ID code 0 to 10 (not including 10) must be produced
3. The random number must be locked using the command: ***srand(1948)***
4. A random warehouse must be selected for each of the 100 items using the command: ***int randomCode = rand() % 10***
5. Note that if the code of the warehouse or the item already exists, it will not generate it, it will simply use the existing one.

q – quit

General instructions

1. You received a run code named task02.out, input02.txt, output02.txt where Your code must behave like the given task02.out runtime code.
2. **In any case of a contradiction between the instructions in the question and the task02.out running code, the behavior of the task02.out prevails.**
3. Your running code name should be “ex02.out”
4. You have to write a new C source code file base on the given ***main_template.c*** file. The name of your C code should be ***ex02.c*** and you must submit only one C code file.
5. You must use the following command in your Linux terminal to compile your C code: ***gcc wx02.c -o ex02.out***
6. Run your running code: ***./ex02.out***
7. Check your code using your keyboard and the printout on your Linux terminal screen.
8. Before submitting to the model your files you should check your running code with the following command line:
   ./ex02.out< input02.txt> myOutput02.txt
9. If your code is running OK then ***myOutput02.txt*** should be the same as the given ***output02.txt***
10. You should have at least 2 files:
    a. Your C code file ***ex02.c***
    b. ***gpt.txt*** file, this file will include a copy of all your conversation, with your LLM (including the code that it produces for you). D) other files that might be requested in the question. The names of the files will be as indicated in each question, making sure to use upper- and lower-case letters.
    c. Optional file name ***t02.pdf***. Only one PDF file, containing special certificates. Approvals such as reserve approval. It is mandatory to attach to the approval a letter explaining the relief or request and the appropriate approval including specifying relevant dates.
11. Make sure that the program works properly both in the gcc compiler and that the program ran properly in the Linux environment. The ***input02.txt*** file provided to you must be used as a series of input commands and compared to the expected result shown in the ***output02.txt*** file provided to you. The tester that will check your submission will test your code with other input files and output files and he will perform a much wider series of tests, including edge case and error tests.
12. You have to follow **VPL** instructions in order to submit the task.
13. Remarks:
    a. It can be assumed that every input ends with pressing the Enter key.
    b. Numerical Input Validation: Check inputs for validity (e.g., positive, negative, zero, whole, fractional). Invalid inputs include anything other than numbers, spaces, and tabs at the start or end.
    c. Normal numerical input is considered input with consecutive digits, spaces and tabs, when the spaces and tabs are either at the beginning or at the end of the input (or both) in sequence and the number should be in the requested range, any other input will be considered invalid.
    d. In any case of an error, or any unexpected response. The program will always use the given function ***print_error_message()***
    e. There must be no situation where the program crashed or performed an illegal operation without this notification.
14. You can get compilation warnings on commands: ***scanf , strlen , strcpy , strcat , strcmp , gets*** .
15. You can get compilation warnings on ***while(1)*** and also on single line warnings.
16. No other compilation warnings or errors are allowed.

17. Allowed Material: Use only material studied until the publication of the work, including conditions, loops, arrays, strings (with studied functions), and two-dimensional arrays.
18. When entering names and code numbers, make sure that the user does not enter more than 100 characters in the input.
19. In order to test your code use the given input file and check accordingly the output file. Your output file should be exactly the same as the given output file.
20. Submission in pairs, ID numbers must be written in a comment inside the submission file. Everyone must submit separately in the model to receive a personal grade.