# SECTION 2: ASSIGNMENT #2 INFORMATION

You have been approached by a company to create a front-end system for a vending machine written in C. The vending machine serves all kinds of treats such as candy bars, soft drinks and other tasty snacks.

In this assignment you are to implement an application that will perform the above tasks as a single user model – which will then be used as a prototype for how the new system would look.

You will need to demonstrate your understanding and programming ability, with respect to more advanced C programming principles. The concepts covered include:

- Command line arguments.
- File handling.
- Dynamic memory allocation and linked lists.
- Modularisation and multi-file programs.
- Makefiles.
- Function pointers
- All concepts covered in Assignment #1.

Your assignment must compile and execute cleanly on the coreteaching servers in the fashion below:
Compile:

```
[s3344949@csitprdap02 ~]$ make
```

Execute:

```
[s3344949@csitprdap02 ~]$ ./vm [command line args]
```

## Startup Code:

The startup source code for vm will be provided on the shared drive:

```
/home/el9/E70949/shared/assignment-2/a2-skeleton.zip
```

(that is e-elle-9, not e-one-9)

You are expected to be submit a modularized solution with multiple source files for this assignment. This includes creating your own data structures and .c and .h modules.

**Unlike assignment one you will be expected to make some changes to the skeleton code as follows:**

- You are able to move data structures and function prototypes/definitions into more appropriate modules.
- Unless attempting the "Optional Requirements" you are not permitted to alter function prototypes. A function prototype includes the name, return type and parameter list of a function.
- You are expected to use ALL given data structures and and functions.
- You may add elements to structs if appropriate.
- You may create typedefs for the provided data structures but you may not alter the function prototypes or the predefined types.
- You do not need to submit vm_type.h. You will be expected to move the declarations in this file into your own .c and .h modules as appropriate.
- You may change the library structure if it fits in with the design of your program (for example the #include statements).
- Keep in mind that we will mark for appropriate software design in requirement 17 (Hint: do not include libraries more than once).

Permission to change the startup code beyond this list is not normally given, unless there is very good reason (i.e., a bug). You will need to build upon the code that is provided instead. If you have any concerns about the startup code, please post your query to the Blackboard discussion forum.

# Functional Requirements

This section describes in detail all functional requirements of Assignment #2. A rough indication of the number of marks for each requirement is provided. You should make a conscientious effort to write your program in a way that is capable of replicating the functionalities of the vm program, described in the requirements below.

## Requirement #1 – Command-line arguments 2 marks

A user of your program must be able to execute it by passing in the names of two data files that the system is to use. You need to check that exactly 3 command line arguments are entered.

Your program will be run using these command line arguments:

```
[s3344949@csitprdap02 ~]$ ./vm <itemsfile> <coinsfile>
```
For example:
```
[s3344949@csitprdap02 ~]$ ./vm items.dat coins.dat
```

## Requirement #2 -- Load Data 8 marks

Your program needs to be populated with the data provided in the 2 data files whose names are conveyed via the command line. You will need to tokenize this data in order to load it into the system. Use the structure definitions provided in the startup code to store your system data. As part of this requirement you need ensure that the specified files are valid (that is, they exist and the data contained in them matches the specification for this assignment). You must also ensure that data is added to the list in sorted order by ItemName. You need to abort the program and provide a suitable error message if this is not the case.

Item File Format:
```
[ItemID]|[ItemName]|[ItemDesc]|[Price]|[NumberOnHand]
```

Please note that the Price is stored as numbers delimited by the '.'. The number to the left of the '.' is the dollars and the number to the right will be the cents, for example:

```
I0001|Coke|375 ml Can of coke|3.50|50
```

Where the 3 represents the dollar component and the 50 represents the cents component of the price. A missing '.' from the price should be considered a data error.

Money data file format:
```
[denomination],[quantity]
```

That is, there will be a row for each value of money that exists and the system will have an amount of each denomination. For example:

```
1000,3
500,4
200,20
100,30
50,5
20,3
10,40
5,20
```

This means that the system currently has 3 x 10 dollar notes, 4 x 5 dollar notes, 2- x 2 dollar coins, etc.

Note that the above denominations are the only valid denominations for your vending machine. The vending machine does not accept $20 or $50 notes. A valid file will always contain exactly 8 denominations. If you wanted to initialise the vending machine with no change then the valid way to do that would be:

```
1000,0
500,0
200,0
100,0
50,0
20,0
10,0
5,0
```

Please note that you cannot assume that the data contained in these files are valid. Some examples (this is not a complete list) might be, there may be lines with too many or too few fields, and the data in each field may not be of the correct type, range and/or length.

Sample (valid) data files have been provided with the startup code. Make sure that your program works with at least these files. We will also be testing your program with invalid data in the data files.

We recommend that you get your program working for the (valid) provided data files first. Then spend any remaining time on your assignment testing for invalid files after you have completed most/all of the remainder of the assignment.

## Requirement #3 Implement Main Menu Structure 6 marks

The main menu data structure is to be implemented as an array of struct menu_item.
A menu_item is defined as follows:

```
struct menu_item
{
     char name[NAMELEN+1];
     BOOLEAN (*function)(struct vm*);
};
```

Where `name` is the text to be displayed for a menu item, eg: "Display Items" and `function` is the function that implements that option which in this case would be `BOOLEAN display_items(struct vm*)`. Your task in this requirement is to implement the function `init_menu(struct menu_item* menu_items)` which will initialise the menu array. The elements of the menu_items array will be as specified in the next requirement. Please note that you are expected to initialise the code in such a way that the code will be easy to maintain. Do not hard code values, and do not use magic numbers for array indexes etc. You are expected to use good coding practices at all times.

## Requirement #4 Display Main Menu 4 marks

Your program must display an interactive menu displaying 9 options. An example of what your menu should look like is displayed below.

```
Main Menu:
   1. Display Items
   2. Purchase Items
   3. Save and Exit
Administrator-Only Menu:
   4. Add Item
   5. Remove Item
   6. Display Coins
   7. Reset Stock
   8. Reset Coins
   9. Abort Program
Select your option (1-7):
```

Your program must print out the above options which will be stored in the menu_items array and then allow the user to select these options by typing the number and hitting enter. Upon selection of an option, appropriate function will be called via the function pointer of the item selected. Upon completion of all options except "Exit", the user is returned to the main menu. You can assume that customers can only see the first two menu options, and the administrator can see all of them.

**Under No Circumstances are you to password protect the administrator functions of this program. If the marker cannot access parts of your application, they cannot mark it and you will get zero for those components.**

The behaviour of these menu options is described in requirements 5-10.

## Requirement #5: Display Items – 4 Marks

This option allows the user to request a display of the items available for purchase. This is the data loaded into the linked list in the requirement 2. This requirement should behave as follows - from the main menu, the user selects 1 and data should be displayed in the following format:

```
Items Menu
----------
ID    |Name                 |Available |Price
-----------------------------------------------
I0001 |Coke                 |12        |$ 3.50
I0003 |Lemon Cheesecake     |439       |$ 4.00
I0005 |Lemon Tart           |5         |$ 3.75
I0004 |Mars Bar             |3         |$ 3.00
I0002 |Pepsi                |0         |$ 3.50
```

## Requirement #6: Purchase Item – 7 Marks

This option allows the user to purchase an item from the menu. This function is called from the main menu when the user has finally decided to purchase an item. This function allows the user to pay for their item by collecting money from them, and adjusting balances within the system as appropriate. Change is then given back to the customer, if appropriate. The number of items on hand should also be deducted. You should not allow an item to be purchased if there are 0 or less of that item on hand.

For example:
From the main menu the user selects 2

```
Purchase Item
-------------
Please enter the id of the item you wish to purchase: I0001
You have selected "Coke – 375 ml Can of coke". This will cost you
$3.50.
Please hand over the money – type in the value of each note/coin
in cents.
Press enter or ctrl-d on a new line to cancel this purchase:
You still need to give us $3.50: 200
You still need to give us $1.50: 300
Error: $3.00 is not a valid denomination of money.
You still need to give us $1.50: 500
Thank you. Here is your coke, and your change of $3.50: $2 $1 50c
Please come again soon.
```

After which the user would be returned to the main menu.

When refunding money, you must display each note or coin separately used in the refund and you must ensure that prior to the sale that there is sufficient denominations in the system so that the customer is given the correct change. You must also subtract these coins from the coins array if a sale can take place. Note that coins entered to pay for an item can form part of the change that is given to the customer if that is required to give the customer the smallest amount of change possible. If the vm cannot give correct change then the sale should not occur, and your program should display an appropriate message explaining why.

If the user presses enter or ctrl-d on a new line, refund all the coins/notes they have entered so far and return them to the main menu.

## Requirement #7: Save and Exit – 5 Marks

You must save all data to the data files that were provided on the command line when the program loaded up. When the saving is completed, you must have the program exit. The specifications mentioned for each file must be maintained and the program must be able to load up your files as easily as it loaded up the files that we have provided you with. Also, at this point, once you have implemented dynamic memory allocation, you must free all memory allocated and exit the program. A program which does everything but exit the program will still lose marks.

## Requirement #8: Add Item – 5 Marks

This option adds an item to the system. When the user selects this option, the system should generate the next available item id and associate that with this item. The user should then be prompted for the Name and Description and Price (a valid amount of money in dollars and cents). The item should then be allocated the default "on hand" value specified in the startup code. The new item id shall have an ‘I’ prepended to it and will be 5 characters long. For example:

```
This new meal item will have the Item id of I0006.
Enter the item name: Shish Kebab
Enter the item description: a dish consisting of small cubes of
meat threaded on a skewer that are grilled or roasted.
Enter the price for this item: 8.00
This item “Shish Kebab – a dish consisting of small cubes of meat
threaded on a skewer that are grilled or roasted.” has now been
added to the menu.
```

Please note that the price entered for a item must have a dollars and a cents component as above. Items should be added to the list in sorted order by ItemName.

## Requirement #9: Remove Item 5 marks

Remove an item from a category and delete it from the system, including free memory that is no longer being used.

Example:

```
Enter the item id of the item to remove from the menu: I0001

"I0001 – Coke - 375 ml Can of coke" has been removed from the
system.
```

## Requirement #10: "Reset Stock Count" 2 marks

This option will require you to iterate over every stock in the list and set its 'on hand' count to the default value specified in the startup code.

## Requirement #11: "Reset Coin Count" 2 marks

This option will require you to iterate over every coin in the coin list and set its 'count' to the default value specified in the startup code.

## Requirement 12: "Display Coins" 3 marks

This option will require you to display the coins as follows. In particular, the counts of coins should be correctly aligned:

```
       Coins Summary
---------------------
Denomination   | Count
---------------------
5 cents        |    20
10 cents       |    40
20 cents       |    3
50 cents       |    5
1 dollar       |    30
2 dollar       |    20
5 dollar       |    4
10 dollar      |    3
```

## Requirement #13: "Abort" 2 marks

This option should terminate your program. All program data will be lost. You should also be freeing memory at this point as well.

## Requirement #14: Return to menu functionality 3 marks .

Your program should allow the user to return to the main menu at any point during these options. The user can do this by either hitting enter or pressing ctrl-d on an empty line. If the user is in the middle of a transaction, that transaction should be cancelled.

## Requirement #15: Makefile 5 marks

Your program must be compilable using a makefile. All compile commands must include the "ansi -Wall -pedantic" compile options and compile cleanly with these options. Your makefile needs to compile your program incrementally, i.e., use object files as an intermediate form of compilation.

Also include a directive called "clean" that deletes unnecessary files from your working directory such as object files, executable files, core dump files, etc. This directive should only be executed when the user types "make clean" at the command prompt.

Have a look at the courseware on blackboard for examples of makefiles. There are also many examples of makefiles provided in the lecture material.

## Requirement #16: Memory leaks and abuses 5 marks

The startup code requires the use of dynamic memory allocation. Therefore, you will need to check that your program does not contain memory leaks. Use the "valgrind --leak-check=full --show-reachable=yes <command> <arguments>" to check for memory leaks. Marks will only be awarded for this requirement if the feedback valgrind provides reports zero memory leaks and no other memory related problems.

Another common problem in is memory abuses. These are inappropriate accesses such as reading from uninitialized memory, writing to memory addresses you should not have access to and conditional statements that depend on uninitialized values. You can test for these again by using valgrind: valgrind --track-origins=yes <command><arguments>

## Requirement #17: Software Design Choices – 6 marks

As part of the implementation of this assignment you will have to make a range of design decisions. We have only provided you with the skeleton code to implement the basic assignment. Your task here is to create additional source and header files to manage the various data

structures in this assignment as separate modules. Each module should be self-contained and complete in its own right.

## Requirement #18: Proper Use of an ADT – 6 marks.

In this assignment, you are implementing an Abstract data type – a list. One list is implemented as an array and the other list is implemented using a linked list. For this requirement you will need to propose a list of interface functions for each list and implement these. All reference to these types should be via these interface functions.

## Requirement #19: General requirements – 15 marks

You must read the "Functional Abstraction" "Buffer Handling" and "Input Validation" requirements of the "COSC1076 Advanced Programming Techniques General Assignment Information" document. These requirements are going to be weighted at 7 marks, 4 marks and 4 marks respectively.

## Requirement #20 – Assignment Demonstration 1 – 5 marks

You are required to attend an assignment demonstration in week 10. You will be required to demonstrate requirements 1-4 in this demonstration. You will be required to compile and run your program on saturn, jupiter or titan. Please attend your regular scheduled lab for this.

We have provided you with a tester object file that you will link with your program to demonstrate that the data has been loaded into memory successfully. Linking with this object file will only work on linux 64 bit computers. After the demonstration you will be required to submit your code to weblearn.

## Requirement #21: Assignment Demonstration 2 – 5 marks

You are required to attend an assignment demonstration in week 11. You will be required to demonstrate requirements 5,6, 8 and 9 in this demonstration. After you have demonstrated your program working you will need to submit your files to weblearn.

# Optional Requirements

These requirements outline some interesting extensions to your assignment that show you some ways that you can extend what you have learnt in this course. These requirements will only be assessed if you get 80% or more on the rest of the assignment so if you are struggling with this assignment, you are not encouraged to try these out during the pressure to get the work for assignment 2 done.

If you are going to attempt these sections you MUST submit one zip file containing two directories. The first directory must be named BASIC and contain the code for the core specifications listed above. The second directory must be named OPTIONAL and contain the code for the the optional requirements. You can and should include a second readme.txt in the OPTIONAL directory with details of your changes to the skeleton code and listing which of the optional requirements you have implemented. If you are not attempting any of the optional requirements then do not submit any directories in your zip file as the standard penalties will still apply.

However, these requirements are intended to both give students who want more of a challenge than is already provided a chance to test out more ideas and to further your C programming skills. If you are a strong student who has done well in the first assignment and you have another idea for an extension to your assignment, do not hesitate to discuss this with your lecturer.

If you are not such a strong student, I encourage you to look at these ideas as possible ways to brush up on your C code and prepare for your next semester of C. The skills you will learn through performing these tasks are important skills you will be required to know in later C programming courses such as COSC1285 - Algorithms and Analysis, COSC1112 - Operating Systems Principles and COSC2406 - Database Systems.

If you get over 100% in this assignment due to these bonus marks, these marks may be applied to your mark in other parts of this course. Also note that these requirements involve more effort than would normally be allocated for this amount of marks. They are meant to be challenge tasks for those interested in furthering their learning.

Even if you don't get time to complete these tasks during the semester, we recommend you look at implementing these over the summer break. While we have listed these as optional requirements for this assignment, these skills are essential for your further progress as C developers.

## Binary I/O (Bonus 5 marks)

For this requirement you will be required (on top of the other requirements in this assignment) to use fread() and fwrite() to be able to read data for your linked list from a binary file and write it back to a binary file. So, instead of storing the struct data in its ascii representation, you will

write functions which will allow the data stored in each vm_node to be written to disk directly as binary. If you are looking for a way to start on this requirement, have a look at the fwrite() example in week 6's lecture. For this requirement, you would also need to provide a way that the use can specify from the command line to use binary I/O for either saving or loading rather than ascii I/O.

You will only get marks for this section for a successful implementation. If you attempt this section and it does not work there will be no marks allocated. The marks will be allocated as follows:

Successful writing of the binary file: 2.5 marks.
Successful reading of the binary file: 2.5 marks.

## Generic Programming (Bonus 5 marks)

The linked list described in this assignment is for a particular type – a vm node. The problem with this is that if we want a different kind of linked list, we would need to make many modifications to allow for this. The task in this requirement is to use void pointers instead for the data pointers and provide functions for comparison of the type. Note that this is a fairly advanced task so you should only attempt this if you are a fairly comfortable C programmer.

Please note that you will not get the marks for this are only attainable if you submit a correct implementation. There are no half marks for this requirement. With this in mind, we encourage you to submit two versions of your assignment if you are aiming for these marks – a version that uses generics and a version that does not in case the generic version does not work as expected, we don't want you to fail the assignment.

## Pointer to Pointer Implementation (Bonus 5 marks)

The linked list used in the normal startup code version of the assignment passes around a pointer to a vm_list. But you will find that an alternative is to pass around instead a pointer to a pointer to the head of the list so we can change the headnode. To get these marks, you need to alter the startup code so that you are passing around a pointer to the head node pointer rather than a pointer to the vm_list.

# Submission Information

**Submission date/time:**

Submission details for Assignment #1 are as follows. Note that late submissions attract a marking deduction of 10% per day for the first 5 days. After this time, a 100% deduction is applied.

| Category | Due Date/Time | Penalty |
|---|---|---|
| On Time | Friday 17th Oct, 9.30 pm | Not Applicable |
| 1 Day Late | Saturday 18th Oct, 9.30 pm | 10% of total available marks |
| 2 Days Late | Sunday 19th Oct, 9.30 pm | 20% of total available marks |
| 3 Days Late | Monday 20th Oct, 9.30 pm | 30% of total available marks |
| 4 Days Late | Tuesday 21st Oct, 9.30 pm | 40% of total available marks |
| 5 Days Late | Wednesday 22nd Oct, 9.30 pm | 50% of total available marks |
| 6 or more Days Late | Not Accepted | 100% of total available marks |

**Submission Content**

For assignment #2, you need to submit at least the following files:

**vm_main.c**

This file will contain your main function.

**vm_menu.h**

**Contains code for the initialisation and display of the menu**

**vm_menu.h**

Header file for vm_menu.c.

**vm_options.c**

This file contains functions for each of the 8 major menu options.

**vm_options.h**

Header file for vm_options.c.

**vm_utility.c**

This file will contain additional code for the running of your program. For example, this can include your own functions that help you collect and validate user input. It may also contain functions to load the data files and initialize your system to a safe state.

**vm_utility.h**

Header file for vm_utility.c.

**Makefile**

This file will compile your program in an incremental fashion.

**vm_readme.txt**

This file will contain important additional information about your program that you wish your marker to see. Examples include incomplete functionality, bugs, assumptions, and so on.

It is recommended that you ask about any assumptions you want to make in the Blackboard discussion forum first before you make them.

You will also be expected to add your own files. Each data structure / section of the project should have its own .c and .h files – that is, you are required to modularise your program and show good software design in your choices for modules. If you think about things from an object oriented design perspective, anything that should be a class in java or c++ should be a separate module in C.

**Submission instructions:**

- Create a zip archive using the following command on the server:

```
[e70949@csitprdap02 ~]$ make archive
```

- This command would generate a zip file named "username-a2.zip" with your nds username substituted for "username".
- Submit the archive to weblearn, which is accessible via the learning hub.