20COA202 Coursework

F028415

Semester 2 2020-2021

# Introduction

How to use the home system:

1. Using the left and right buttons go through the different menu options (the arrows indicate where you can go). The first set of menu options will be for which floor to choose. Press the select button to choose a floor e.g. first floor.
2. You can select the option show data which doesn’t go to a new floor but prints stored data to the serial monitor.
3. Next the menu options will be for what room to select and this is the same as the previous in the sense that you navigate using the left and right buttons. The down button can be pressed to go back to the previous menu options floors.
4. The next menu options is for what type setting to choose: light, lamp… this follows the same implementation as the previous.
5. Once a type setting is seleceted you can then select which device name to choose.
6. The next menu option is for which action to choose and so then select an action setting
7. Once selected the current value for that action will be displayed, this can be increased and decreased using the left and right buttons, to set and save the new value press select to not set and save press down meaning back.
8. To back to the top of the menu press back repeatedly to go back through the menu to the beginning or press reset as the values will be saved and so it has no damaging effect.
9. At any point in the program the command q or m can be entered into the serial monitor to perform a query or memory.

# 2 Your code - base implementation

## 2.1 Data structures

The house descriptors will have their values stored in char Arrays:

* floorRoom={'F','G','O','S'}
* firstFlrRoom={'1','2','B'}
* groundRoom={'K','H','L}
* outsideRoom={'G','R}
* typeSetting={'L','A','H'}
* gardenSetting={'L','A','H','W'}
* deviceName={'M'}
* actionSetting={'1','0','L}

Each possible option that can be selected from the menu has a unique key within its own Array.

Another key data structure that is used, but briefly is the char\*\* house variable which is a pointer that points to the other char Arrays:

House {floorRoom, firstFlrRoom, …}

The char arrays will hold the key letter for each descriptor, this method was chosen for memory basis as opposed to choosing words stored in strings. The amount of memory the address’ used is known and it can also be manipulated easily.

**char\* house[9]={floorRoom,firstFlrRoom,groundRoom,outsideRoom,typeSetting,gardenSetting,deviceName,actionSetting}:** This variable holds all the char arrays for the menu key house descriptors. This allows the different char arrays to be accessed with loops

**typedef enum state\_e** **{floors=0,first,ground,outside,type,gardenType,names,actions,values,data,memory}** **state\_t**: This enum is declared for the different states of the menu system.

## 2.1.2 Variables

**int HOUSE\_LEN:** This variable stores the length of the house pointer variable. This makes accessing the house using loops easier because that variable is used for the loop length and it can be changed once to account for all times it declared and used.

**char floorRoom[5]=""**

**char firstFlrRoom[4]=""**

**char groundRoom[4]=""**

**char outsideRoom[3]=""**

**char typeSetting[4]=""**

**char gardenSetting[5]=""**

**char deviceName[2]=""**

**char actionSetting[4]=""**

These char arrays are declared as empty. They will store the house descriptor keys that are read from the EEPROM.

**char\* setting:** This variable is used to store any one of the setting char arrays, either typesetting or gardenSetting. It is declared using malloc with the size of a char times 5. This is used in order to pass the setting variable as a parameter of either gardenSetting or typesetting making it more dynamic.

**char\* room:** This variable is used in the same way a setting variable but is used to store room charArrays. It has the same memory allocation as setting.

**int written:** This variable holds the value read from the first address in the eeprom and stores either 1 or 0, 1 meaning written and 0 meaning not written. This determines whether the writeEE function (defined below) is called or not called.

**static int address**: This variable contains the unique memory address that is created in order to store the action value in the eeprom in order for it to be remeberd. The value is declared at 0 and then is increased by multiplying the (**choice index +1)** by 200 i.e. 1\*200 or 2 \*200 depending on the floor chosen. It is then increased by 0, 50, 100 … depending on the room chosen for a given floor. The calculate address function is the called to return a unique value that will be added to the address to give a unique number for each value.

**static int sumTotal:** Stores the value returned from the function sumIterations.

**static int choice:** Stores the value returned from the function navSettings which will be the index of the house descriptor key in the char array data structure for a given menu selection, i.e. floors, typeSetting…

**static int typeChoice:** Stores the choice variable that saves the index of the typesetting or GardenSetting, in order for it to be used later and not overwritten when choice is used again for a different char array.

**static int nameChoice:** Follows the same concept as typeChoice but for deviceName menu selection.

**static int startingAddress:** This variable stores the first value of the address as mentioned above. This is so that when going back in the menu the address can revert back.

**static int addressAtType:** This follows the same concept of the starting address whereby the address at the state type (this is the state whereby the menu is selecting a type setting such as light, lamp etc) is saved so that when going back in the menu the address at that point is remembered.

**static int uniqueAddressValue:** this stores the value of the generated memory address returned from the function calculateAddress. This value from this variable will be added to the variable address to form the final unique address.

**static int value:** this stores the value stored and read from the EEPROM using the variable address to locate it.

**static int opt:**  short for option this a variable used in the state data that stores the number 1, 2 or 3 depending on the floor which is multiplied 200 in order to replicate the generation of a memory address as explained in the variable address. It is also used in the printSettingsFunction to specify the case that will be called, which specify which settings will be printed.

**static int setNum:** short for settingsNumber, used in the function printSettings function to specify what case as there is a separate case to print out the settings for gardenSetting and typesetting.

**static int addressLoop:** The same as the variable address but this is used in the data state which prints out the value of all stored data. This is done by using for loops and so addressLoop is the unique address generated each iteration.

**static bool chosen:** This bool variable is used in the state values. It is used as a control variable for a loop whereby chosen is false until a value is selected.

**static state\_t state:** This variable stores the self defined enum structure state\_t. Each enum value in the structure will represent a state of the system as defined by the variable name.

**static state\_t backToRoom, backToType, backToName, backToAction:** These variable are all used in the same way and as they are named. backToRoom will store the enum state that is was on before moving to the next state as a result of moving along the menu. This allows the the system to know what room to go back to when going back. There is no back to floor because all rooms go to the same floor menu. Room refers to a state whereby a room can be selected rather than a floor.

**static char actionChar:** This stores a key descriptor from the action char array using the choice index. This variable is used in an if selection statement that is used to determine whether the action is level or time and thus data is formatted accordingly as well as ranges. If the actioChar is an ‘L’ the system know it is dealing with level as opposed to time.

**char command:** This variable stores a char value read from the serial monitor and if the char is a valid command the will switch to the according state.

## 2.1.2 Functions

Void writeEE():

This is a function is intended to write the values from the data structures into the eeprom (if the data has not been written already) so that whenever the program is run, the data is read from the EEPROM directly and does not need to be reloaded. The function takes no parameters and returns no value hence the declaration as void. The function first instructs the eeprom to update address 0 with the value 1 , this is to signify that the eeprom has been written to and if so(if it has been written to ) then there is no need to call the write function.

Using hard coded char arrays that contain the house descriptors, i.e. **floorRoom[6]={'F','G','O','S','-'}** and **actionSetting[5]={'1','0','L','x'}**, the function stores each of these char arrays in the **char\*\* variable ‘house’** (defined above) and loops through this data structure in order to write each single char along with a consecutively increasing address into the eeprom.

Void readEE(char\*\*):

A function designed to read data from the eeprom and store it inside multiple char arrays. The argument required is a **char\*\*** which will again be the **house** variable that stores empty house descriptor char arrays.

The pre defined char arrays that exist in the **writeEE** function, and have their values written to the EEPROM, make use of either ‘-‘ or ‘x’, meaning end of char array but not the final char array to read in the structure, and final char array to read in the data structure respectively. This allows the nested loop to know, when reading from the EEPROM, a dash ‘-‘ means the loop shoud move to the next char array i.e house[i][j] to house[i+1][j], and an ‘x’ means means the end of read and so the loop can break. The char values read from the EEPROM that represent house descriptors will subsequently be written into the empty char Arrays.

void printLcdSettings (char\* setting, int option, int num\_setting)

The purpose of this function is to print out the full house description to the lcd based on the key. Using switch statements each case has hard coded Strings to print to the lcd depending on the key. i.e for case 6 is the key is equal to ‘H’, the lcd is to print Heat. The arguments taken are: **char\*** for the char array that will contain the house descriptors, **int option** which is the index of a house key of parameter **char\*** **setting**, and **int num\_setting** which specifies a switch case number.

void printSettings(char\* setting,int option,int num\_setting)

This function is the same as the above in its operation and how it works. This function is used to print the house descriptors the serial monitor.

int navSettings(char\* setting, int num\_setting)

This function is used to navigate the menu. It takes arguments **char\* setting** and **int** **num\_setting** , these are passed into the function **printLcdSettings** where their purpose is explained above. Using left and right buttons on the Arduino the function is able to navigate the menu by incrementing or decrementing an **int variable select** respectively, which also acts as an index and so when the function **printLcdSettings** is called, the index is passed as the parameter **int option**, the char array is passed and **char\* setting** and the enum value of current menu options, i.e. **floors**, is passed as **int num\_setting**. This is run in a while loop and so with the function **printLcdSettings** it creates the menu navigation system. Once the select button is pressed the loop ends and the index of the chosen options house description key

is returned. The while loop use **bool variable chosen** as a control variable. If select is pressed then chosen is set to true and the choice index is returned. If a down button is pressed this returns the number 50 which is an arbitrary number I chose to represent going back in the menu. If a value is entered into the serial monitor the number 20 is returned which I chose to represent calling serial monitor functions.

In order to create the button system we first create the variable **static int old\_buttons.** This variable will store the result of **lcd.readButtons()** which returns the int value of a button. Then we create the variable **buttons** which will also store the int value of the button currently being pressed. Next the variable int variable **changes** is created which stores the bitwise AND of **old\_buttons** and the complement of **buttons.**  This is so that changes will always return a 0 if both **old\_buttons and buttons** are the same and return an int value that is not 0 if they are different thus telling the system that a button has been pressed and let go. This follows by the select statement if **changes** is true (not zero) then perform the following select statements for the buttons that are: if the button selected is the right button and the current **select variable value** is less than the length of the house desc char array it is using to print then selects will be increased by 1 and **printLcdSetting** will be called which will print to the lcd menu, the current select item for the given **setting** char array based on the index which is the variable **select.** if the left button is selected this is just the opposite and so **select** is decreased by 1 and must not be less than 0 and so will print its menu item to the lcd accordingly. If the button SELECT is pressed then **chosen** will equal true and the loop will end and return the current **select** index.

**Old\_buttons** is set to equal **buttons** this is so that **old\_buttons** will hold the button first pressed and so will then be compared by the new **buttons** when the loop starts again.

int sumIterations(char\* firstIter,char\* nestedIter,char\* secondNestedIter)

This function takes 3 char \* arguments. The reason for this function is to create a value by summing the indexes within a triple nested loop. This value is used in the creation of a unique address by dividing the result value by the length of the char array of the first loop (first iter). In this home system that will be the **typeSetting** char array: water, heat… This provides a unique range for each type setting and so the creation of this value is a crucial step in creating a unique address.

int calculateAddressValue(char\* setting,int sumTotal,int typeChoice,int nameChoice,int actionChoice)

This function creates the unique address. The argument **char\* setting** is used the divide the return value of **sumIterations** by its length thus giving a value to create a unique range as aforementioned in the previous function explanation. Other arguments namely int **sumTotal, typeChoice, nameChoice,** **actionChoice** are used in the generation of the unique address. **typeChoice** is multiplied by the created range in order to create the range for that given choice.

Then this formula is used:

modValue=sumTotal/strlen(setting)

modProduct=(typeChoice\*modValue)

(modProduct+(typeChoice+1)+(nameChoice+1)+ actionChoice) **mod**  (modValue\*(actionChoice+1)+1)

This creates a unique value for the memory address.

freeMemory():

This function works by calculating the space between the stack and the heap in order to calculate the amount of SRAM data used in bytes. It does this by returning &top - \_\_brkval which subtracts the bottom of the heap from the top of the stack

void showData():

This purpose of this function is to print out the stored data action values to the serial monitor. Printing the stored data means that the function will only print out data where the value has been set. If the value has not been set then it will not print out that particular data. The function takes no arguments. The functions prints out that data of all stored data by using nested for loops in order to go through each possible option. It uses a select statement to determine which floor the loop will go through and thus what room the loop will go through and retrieve data values. It does this be setting the **room** variable to one of the char array house descriptors depending on the floor based on the current loop index. i.e. **if floorRoom[a]=='F'** the **room** will equal **firstFlrRoom.** This selection statement is run in the first for loop which is the loop for each floor. Depending on the floor chosen **opt** is set to either 1,2 or 3. This function also needs the calculate multiple address each iteration and so at the loop the first part of the address is created by multiplying **opt** by 200 which is stored in **addressLoop**.

The second loop (nested loop) uses the given **room** for the length of the loop and as the char array the loop and index will go through. This loop uses a selection statement in order to set the **setting** variable for the third loop. This is because setting can either be typeSetting or gardenSetting (as mentioned in 2.1.1). As the loop goes through the given **room** if the char value is ‘R’, then setting will become **gardenSetting** and **setNum** is set to 5. This is because **setNum** will be used for the case value for **printSettings** which will print out the settings for the specific setting in this case **gardenSetting.** If the char value is not ‘R’ then setting will be equal to **typeSetting** and **setNum** will equal 4. The address is further calculated by adding 50 \* the current index to **addressLoop.** The **sumTotal()** is also calculated. It is done in this loop because the variable **setting** can be determined and **sumTotal()** uses this variable in its calculation (refer to 2.1.2).

The next 2 nested loops include the loop for the **setting** char array and the **deviceName** char array respectively. These do not contain any other code apart from the for loops that they nest. Their purpose is to go through every single option the menu can select.

The next loop is the loop for the action setting. This is also the last loop and so is the loop where all the values get printed. First of all the unique address for the current iteration is created and is added to **addressLoop** in order to form the final unique address. the variable value is declared and stores the eeprom value gotten from reading the eeprom using the unique address. a selection statement is declared to check the char of **actionSetting**. If the char is an ‘L’ this tells the system that we are dealing with **level** and so when printing its value it is formatted accordingly. Before the value is printed with a format a select statement is performed whereby if the value stored in the eeprom is greater than the range for a particular actionSetting (in this case the actionSetting is ‘L’ and so the range is 100) then **printData** will be set to false to indicate the eeprom does not have a value that has been set by the menu but rather is holding a default value. Once the value is retreieved and **printData**  is set to true then the printSettings function is called for each house descriptor key char array using the index of each loop as the value for the parameter **int option** (in print settins, see 2.1.2) . When accompanied by a slash for the formatting the function prints out all the possible menu outcomes with their associated values. i.e.

First Floor/1st Bedroom/Light/Main/On Time: 05:00.

void showMemory():

This function uses the function defined in 2.1.2 **freeMemory()** to calculate and print the amount of memory free and the amount of memory used. It subtracts the return value of **freeMemory()** from the size of the SRAM (2048) in order to print the amount of memory in use.It then prints the amount of memory that is free by printing the return value of **freeMemory().**

## *Diagram Description automatically generated*2.2 FSMs

floors:

In this state the variable **address** is declared as 0. The function **navSettings** is called and this function is what prints the out the menu and allows the navigation of the menu using the buttons and returns the choice index. Once a floor setting is chosen, the address is increased by 200 \* (choice+1), this is the first step for creating the address (this is done so that each floor has a different address that it starts at, 200 for first etc). the variable **startingAddress** will store the value address, this is so that in the next state the previous address was remembered just in case you want go back. Depending on what number choice is the state will change to a new floor, i.e. If choice is 0 then state will change to first. Or the function **showData** will be called.

first, ground:

Depending on what choice is set in the state **floors** the system will move to one of these states. These states generally have the same implementation. The address is initially declared as **startingAddress** this is so that ever time the state is called the address is set to a saved starting value. The function nav settings is called for the given house descriptor char array and stores the choice. The address is increased from its current value by **choice** \* 50.

If choice Is 50 (if the back button is pressed) then the state will be set to **floors** allowing the menu to go back to the floors selection. Once a room is selected then the state is set to the next menu selection being the state **type** and backToRoom is set to the current state, this is so that the system knows what room to go back to.

Outside:

The state outside is the same as **first** and **ground** with the difference that a select statement is used to check what room is selected from **outside**. This is because if the **garden** is selected there is a different set of type settings and so it has its own state **gardenType**. And so the select statement checks this by checking the key for the house descriptor.

type, gardenType:

address at **type or gardenType** is stored by taking the value of the current address. **typeChoice** stores choice, this is because it will be used later in the generation of memory address and so the value needs to be stored before its overwritten by the next state. **setting** is also saved by storing the char array of the current house desc menu selection, i.e. either **gradenSetting** or **typeSetting**. **sumTotal** is stored by calling the **sumIterations** function. Once a type setting is selected the state changes to names and **backToType** will store the current state. And if the down button (go back) is pressed then state will equal **backToRoom**.

names:

This state is basic and has the same philosophy as the others. **navSettings** is called which will print the new menu and store **choice**. **nameChoice** stores choice in order to save it and use it later in the unique address calculation. Once a name is selected the state changes to the state **action**.

actions:

In this state address is set by the variable addressAtType and uniqueAddressValue is set to 0. navSettings function is called and the menu selection for action settings is printed to the lcd. Once an action setting is selected (on time,off time,level) then the uniqueAddressValue will be calculated using the function calaculateAddressValue. Once calculated this value is added to address to form the unique address.

actionChat will save the char gotten using the actionSetting char array and the choice index thus storing the char of the menu option selected. State will then move onto values.

values:

In this state the variable value will store int value gotten from calling eeprom read using the generated unique address as the parameter.

The select statement checks the actionChar in order to determine how to format the value that it got from the eeprom and will print to the lcd. If the action char is ‘L’ and value is greater than 100 this indicates an invalid value and most likely a default eeprom value and so it will be updated and set to 0.

The value is then printed to the lcd. If the action char is not ‘L’ (for level). Then the system will then assume that it dealing with time (on time, off time), and so it an invalid value will be greater than 23. Furthermore when printing the value it will be formatted to look like a time and so will look like 0:00 when printed and so on. After the value is printed the value is then able to be increased and decreased with the use of the left and right buttons (the implementation is the similar to **navSettings** with a few differences). If the system is dealing with **‘L’** and so level then the value will get increased and decreased by 10 using left and right buttons within the range 100 and printed to the lcd to show value increasing. And when dealing with time the range is 23 and it increases and decreases by 1. Once the select button is pressed the eeprom will be updated using address and the new value and chosen will equal true and the state will go to the previous which is **backToAction.** If the down button is pressed then it will do the same without updating the eeprom.

## 2.3 Testing and testing

*Describe your approach to debugging. Include any code you have included to assit with debugging.*

*Describe any code that exists purely to test other aspects of your program.*

My approach to debugging is using #ifdef DEBUG statements that print to the serial monitor to check the correct values are being written and read by the program.

1. As my data structure for the house descriptors will be char arrays with the first letter of a descriptor being used as the char identifier i.e ‘F’ =Floor ,and so i have created a for loop to print out the char array alongside the memory location to check if it was written correctly so that when it is stored in the eeprom the correct data goes in the correct place.
2. Once the data has been read from the EEPROM via the function readEE(), there is a #ifdef DEBUG statement that will print out the char arrays to check if data was entered correctly.

# 3 Extension Features

## 3.1 LAMP

Lamp setting is implemented by adding the house key descriptor to the char array **typeSetting** and **gardenSetting**. It is implemented the exact same way as the other type Settings and so follows the same documentation and is used with the same functions. The key for lamp is an ‘A’.

## 3.2 OUTSIDE

char outsideRoom[3]

outside is implemented by creating a new char array called **outsideRoom**. It is implemented the exact same way as the other rooms. It is first written to the EEPROM using the function **writeEE()** which writes the hard coded char array **char outsideRoom[4]={'G','R','-'}.** It is implemented the same way as the other rooms and so follows the same documentation. It is read using the function **void readEE(char\*\* house).** **outsideRoom** is defined in the **house** variable.

## 3.3 QUERY

This feature is realized by first adding code to **navSettings** the **values** state. I added a check for if the serial monitor is available then **chosen** will be set to true in order the end the loop and choice will equal 20. The char variable **command** will then run and store a serial read to check the value entered into the serial monitor. If the value entered is a char q then **showData()** will run which performs the query function.

## 3.4 MEMORY

This feature is called by the function **showMemory.** It uses the function defined in 2.1.2 **freeMemory()** to calculate and print the amount of memory free and the amount of memory used. It subtracts the return value of **freeMemory()** from the size of the SRAM (2048) in order to print the amount of memory in use.It then prints the amount of memory that is free by printing the return value of **freeMemory().**

## 3.5 SOFT

*NOT IMPLEMENTED.*

## 3.6 EEPROM

The way the eeprom was implemented for this system was by storing the house key descriptors and the storing the action values. The house descriptors are written to eeprom with the use of the function writeEE (explained in 2.1.2). The house key descriptor are read and stored in char arrays with the use of the function readEE(explained in 2.1.2). The action values are stored in the eeprom by generating unique address values using the function **calculateAddress** thus allowing them to be stored and read from the eeprom without any overwriting issues and locating issues. When writing to the eeprom the function update is used rather than write which saves the amount of times that the eeprom can be written to. The action values of the house menu are printed using the eeprom read function with the address.

# 4 Conclusions

*Reflect on what is fully working and what only partially working. Include a description of the parts of the code that are particulary worth mentioning.*