Lab 1
3 Dimensional Array

May work in groups of up to 2 people

This project will require you to understand arrays. The goal of the project will be to use a 3 dimensional array to store data that can be used to calculate totals in multi-dimensions. You will be required to write this code in the C language and work individually or in a group of up to 2 people on the project, but use the online forum for collaboration on anything related to the assignment for help (keep general questions on the forum, and contact me for class related issues, or if the forum is not helping).

You will have flexibility in what you want to hold in your 3 dimensional array, and you will need to create a header file that you will #include that contains the raw data. An example of an array would be as follows:

```
float cashIn [12][7][10];
```

This is an example of a data container that will hold float values as money. The dimensions in the array represent [12] months in a year, [7] days in a week, and [10] cash registers. NOTE: Keep the array dimension sizes small due to the fact that you will need (first dimension size) * (second dimension size) * (third dimension size) of data in your header file. Once you have your container defined, you will need at least 3 functions that use the data in the 3 dimensional array that will output a calculation from the function. One function will require an additional prompt for user input as in the example below for sumLane.

An example of the 3 functions could be:

```
float sumTotal(const float[][][]);            //Sum all 3 dimensions
float sumWeekday(const float[][][]);      //Sum 2nd dimension for values 1-5
float sumLane(const float[][][], int n);   //Sum 3rd dimension for passed lane #
```

NOTE: but these types of function prototypes are only necessary for non-file scope defined arrays. If the variable is already at file scope, sending the variable through the function call should not be used.

In each of these functions you will need to use looping to iterate through a dimension (or multiple dimensions) as a set, and calculate a value based on an algorithm defined for the function. The algorithm chosen will require you to use nested for loops to understand how data passes through different blocks of code. Your algorithm will need to be clearly defined in your design document, as well as inputs and outputs per function. The calculations will need to use arithmetic in for loops to walk through the set and return a calculated value.

Your program can have either command line arguments passed to invoke the 3 functions, or you can utilize a menu system that will allow users to select the functions. An example of the menu for the above functions could look like:

MENU

------

1. **Total for the entire year**
2. **Weekday totals for the year**
3. **Lane totals for the year (NOTE: will need another prompt for lane #)**
4. **Exit**

Enter Selection : _____

A menu can be created easily with a while (true) loop that only exits if someone presses 4 and then the program will call exit().

```
while (1) {
        printf(“MENU\n”);
        printf(“=====\n”;
        …….
        printf(“Enter selection : “);
        sel = getchar();
        /* now use a switch statement to evaluate and if sel is not 1, 2, 3, 4, then
print an error message and show menu again
        */
}
```

If you decide to not use a menu you will need to have a “usage” statement to let the user know how to use the command. An example would be if you allowed the user to enter:

```
$ calcCash
```

would print:

```
USAGE: calcCash [ showYear | showWeekday | showLane LaneNum ]
```

Requirements

Design document

Please create design doc in text format - not MS Word/etc

You are required to start with a design document that will define your program. The document will need to define at least the following elements:

- Program description paragraph
- Usage statement (menu or command line)
- Data structure definition for 3 dimensional array
- Each function defined with
- Input type(s) (if any)
- Output type
- Algorithm description
- Listing of files required and what type they are (.h header, .c c code, etc..)

C program

Header file that holds data set (could also hold your 3 function definitions)

3 functions that will calculate a value from the 3 dimensional array. The functions should use the array as a read-only variable using the const keyword in either the data definition in your header file, or in the function definition (and use).

Main function should only hold usage statement (if command line) or menu and from there you should just call the functions. Keep main clean with only usage/menu and calls to functions.

Grading Criteria

You will need to submit a single file (zip or tarred) inclusive of all of your source code including header files (or any additional files), your design doc, and a test script.

Your test script will be the output of the command “script”. Script is a command line program that once run, will copy your output to your screen to a file. An example of how to use script is as follows:

```
$ script Project1.txt
```

NOTE: This will start the program and everything that happens on your screen from now until you type “exit” will be included in the file named Project1.txt. If you do not provide an argument to “script” it will create a file called typescript. When you type “exit” you will not be exiting your shell, rather the script program that is copying all of your output to your file. So the proper use of script for project output would be the following algorithm.

1. Be ready to run through all test cases knowing they work
2. Enter the script program
3. Run through all test cases of your project
4. When done, type in exit to leave the script program
5. Include the script output file in project submission

```
/* Header file with data */

float cashIn[12][7][10] =

{

 {

  {3.05,23.41,44.21,23.12,2.10,2.34,12.23,10.90,44.31,30.12},
  {3.12,44.21,32.50,76.45,6.23,6.25,11.32,12.44,9.08,1.02},
  {3.45,23.41,44.21,23.12,2.10,2.34,12.23,10.90,44.31,30.12},
  {9.12,32.12,33.12,45.23,9.87,4.56,12.49,20.33,4.12,86.23},
  {3.12,44.21,32.50,76.45,6.23,6.25,11.32,12.44,9.08,1.02},
  {9.12,32.12,33.12,45.23,9.87,4.56,12.49,20.33,4.12,86.23},
  {3.12,44.21,32.50,76.45,6.23,6.25,11.32,12.44,9.08,1.02}
 },
 {
  {3.15,23.41,44.21,23.12,2.10,2.34,12.23,10.90,44.31,30.12},
  {3.12,44.21,32.50,76.45,6.23,6.25,11.32,12.44,9.08,1.02},
  {3.45,23.41,44.21,23.12,2.10,2.34,12.23,10.90,44.31,30.12},
  {9.12,32.12,100.90,45.23,9.87,4.56,12.49,20.33,4.12,86.23},
  {3.12,44.21,32.50,76.45,6.23,6.25,11.32,12.44,9.08,1.02},
  {9.12,32.12,33.12,45.23,9.87,4.56,12.49,20.33,4.12,86.23},
  {3.12,44.21,32.50,76.45,6.23,6.25,11.32,12.44,9.08,1.02}
 },
…..
 {3.12,44.21,32.50,76.45,6.23,6.25,11.32,12.44,9.08,1.02}
 }
};


/* You can simply copy the first array and paste the values for the rest */

#include <stdio.h>

#include “myData.h”

int main() {

        USAGE or Menu

        Function calls

}

float function1(….) {
}
float function2(…) {
}
float function3(…) {
}
```

Grading Requirements Rubric

Grading Topic	A Level Work (80-100%)	B Level Work (60-80%)	C Level Work (40-60%)	D Level Work (20-40%)	Points / 100
README.txt	**Clear concise** explanation of functionality / design supported with information on a breakdown of modules / functions to support program and thoughts on how to achieve goals. Defined with all team members, email, ID. Document NOT in MS Word format	**Messaging** explaining functionality of program and supporting algorithms / thoughts on how to achieve functionality	Messaging **explaining functionality** on program.	**Lack** of definition, and/or **little** description of functionality.	10 pts
Total function	**All** functionality is complete with function definition in a header file or at the top of the program. Return values defined and tested from calling function. All passed variables are done properly.	**Most** components required complete execution.	**Some** functionality of function properly working.	**Function** defined to accept input	20 pts
(2) prompt functions	**All** functionality is complete with function definition in a header file or at the top of the program. Return values defined and tested from calling function. All passed variables are done	**Most** components required complete execution.	**Some** functionality of execution of programs - possible program failure.	**Prompt** for user input and achieve output to user	30 pts

	properly.				
Correct use of for loops	**All** components of the for loops are setup to be both efficient as well as complete so that no input would break the calculation of the total output. Total output has been clearly defined and tested.	**Loops** are used in a consistent manner with initial start, iterator and how to complete the loop.	**Proper** use of variables in for loops	**For** loops defined	10 pts
Testing output with script file	Script file **clearly shows all** use cases and defines for another tester / quality engineer how to achieve the same results. All test are complete with both successful and failure conditions to ensure full functionality	**Most** of the use cases are displayed and repeatable by instructor	**Achieve** a small % of use cases	Script file **demonstrates** output of your program	10 pts
Data definition	**Array** fully defined and documented for complete understanding of how it is both defined and used	**Array** properly defined with data and able to be readable in the header file.	**Array** has data in it	3 dimensional array defined	10 pts
Menu or prompt	**Complete** use of a menu inclusive of taking in wrong input and prompting user for proper input. Clear way to enter functions as well as how to exit the program clean.	**Most** of the input function completed with the proper use of a case statement to understand input from customer	**Correctly** takes input and then calls the appropriate functions	Accepts input from a user	10 pts