Lab Activity #8 – 2D Arrays and File Streams

Exercise #1:

Write a program that declares a two-dimensional array named `myFancyArray` of the type `double`. Initialize the array to the following values:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>14.12</td>
<td>17</td>
<td>85.99</td>
</tr>
<tr>
<td>6.06</td>
<td>13</td>
<td>1100</td>
<td>0</td>
</tr>
<tr>
<td>36.36</td>
<td>90.09</td>
<td>3.145</td>
<td>5.4</td>
</tr>
</tbody>
</table>

1. Create a function that will return the sum of the third column. Output this result.

2. Create a function that finds the ceiling [the closest whole number that is not smaller than that number. So `ceil(4.6)` is 5 and `ceil(4.2)` is 5, and `ceil(-3.4)` is -3] of each value in the array.

3. Create a function that will use a nested loop to display all the elements in the array to the screen.

Exercise #2:

Write a program that can be used to assign seats for a commercial airline. The airplane has 13 rows, with 6 seats in each row. Rows 1-3 are first-class seats, while rows 5-13 are economy seats.

Create a menu-driven program. Display a “map” of all of the seats on the airplane (display a star (*) to indicate the seat is available; display an X if the seat is occupied). Ask the user which class of ticket they are interested in (first-class or economy) and to select the specific seat(s) they wish to reserve.

You must create this program by writing the following functions: `displayMap` will display the current seating map of the entire airplane, `makeReservation` will let the user select the specific seat(s) they wish to reserve, and `validateFunction` will make sure that a) the user is not trying to reserve seats that are already taken and b) the user’s seat-class choice is appropriate for the specific seats they’ve selected.
Exercise #3:

You are the owner of a hardware store and need to keep an inventory that can tell you what different tools you have, how many of each you have on hand and the cost of each one. The data is organized as follows:

- The first column contains the tool name
- The second column contains the quantity of that product in inventory.
- The third column contains the cost per item or unit price of that product.

Use the following information to start your file:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hammer</td>
<td>76</td>
<td>11.99</td>
</tr>
<tr>
<td>Jigsaw</td>
<td>21</td>
<td>14.99</td>
</tr>
<tr>
<td>Wrench</td>
<td>10</td>
<td>4.49</td>
</tr>
</tbody>
</table>

Write a C++ program that will do the following:

- Read data from a data file you create called `hardware.dat` located on the hard drive. The inventory changes occasionally, so you do not know how much data there is to process.

- Call a function that will return the total value of the store’s inventory for any one particular product (unit price * number of items).

- Compute the total number of items in the inventory.

- Print the output to a file you create called `hardware.out` to display the results. The output in that file should be displayed as follows (make sure you include the necessary headings that are shown below):

```
MY HARDWARE STORE
--------------------------------
TOOL     QUANTITY     PRICE
Hammer   76           11.99
Jigsaw   21           14.99
Wrench   10           4.49

Value of all hammers in stock:   $ 911.24
Value of all jigsaws in stock:   $ 314.79
Value of all wrenches in stock:  $ 44.90
```
Exercise #4: Tic-Tac-Toe (courtesy of Dr. Huo)

The following program is the base for the game: tic-tac-toe. The main function and program structure is given, don’t modify them. Complete three functions as required. Compile and run this program, and then you can play the game. Have fun!

```cpp
#include <iostream>
using namespace std;

const int DIM = 3;
char chessboard[DIM][DIM];

void initChessBoard(char cb[][DIM])
{
    //set all the elements of the ChessBoard to blanks
    //Complete this part in the following:
}

void printChessBoard(char cb[][DIM])
{
    //print all the elements of the chessBoard with each row in one line
    //Complete this part in the following:
}

bool putChequer(char cb[][DIM], int i, int j, char x)
{
    /* if i and j are not out of bound (that is, i and j are in the range of 0 and DIM - 1) and
    cb[i][j] is not occupied (that is, the value of cb[i][j] is blank), set cb[i][j] to be the value of x and
    return true. Otherwise, return false. */
    // Complete this part in the following:
}

bool state(char cb[][DIM], int row, int col, char x)
{
   //judge the state of the game. The player has put x in the position (row, col).
   // If all the elements in this row are x, x wins
   // If all the elements in this column are x, x wins.
   // If x is in the main diagonal, and if all the elements in the main diagonal are x, x wins.
   // If x is in the opposite diagonal, and if all the elements in the opposite diagonal are x, x wins.
```
/* We declare four variables count1, count2, count3, count4 to represent the occurrence number of x in row number row, column number col, in the main diagonal, and in the opposite diagonal. If after calculation, count1, count2, count3 or count4 equals to DIM, return true (that is, x wins). */

int count1=0, count2=0, count3=0, count4=0;

for(int i=0; i<DIM; ++i)
{
    // Complete this part in the following:
    // if the element in position (row, i) is x, count1 is increased by 1.
    // if the element in position (i, col) is x, count2 is increased by 1.
    /* if x is in the main diagonal, and the element in position (i, i) is x,
    count3 is increased by 1.*/

    /* if x is in the opposite diagonal, and the element in position (i, DIM-1-i) is x,
    count4 is increased by 1.*/

    return (count1==DIM | | count2==DIM | | count3==DIM | | count4==DIM);
}

int main()
{
    int row, col;
    int blanks=DIM*DIM;
    initChessBoard(chessboard);
    printChessBoard(chessboard);
    char cur='O';

    cout<<"Input the position(row col), (-1 -1) for exit; It is the turn of "<<cur<<endl;
    cin>>row>>col;

    while(row!=-1 && col!=-1)
    {
        if(!putChequer(chessboard, row, col, cur))
        {
            cout<<"Invalid move"<<endl;
            printChessBoard(chessboard);
        }
        else
        {
            --blanks;
            printChessBoard(chessboard);
            if(state(chessboard, row, col, cur))
            {  

```
cout<< cur << " Wins"<<endl;
    return 0;
};
if(blanks==0)
{
    cout<< "Ties"<<endl;
    return 0;
}
if(cur=='X')
    cur='O';
else cur='X';
} 
cout<<"Input the position(row col), (-1 -1) for exit; It is the ture of "<cur<<endl;
cin>>row>>col;
}

Exercise #5: Complete Your Game!

If you have not already done so, add a loop to your game.

The loop can be an infinite loop (while (1)). The inside of the loop consists of the IF statement that gives the user the choice of playing a game, showing a demo of the game and exiting. When the user chooses to exit, the program ends.

Change the menu above so that instead of the three choices, you have the following choices:

- Play the Madlib game (from Lab #2)
- Play the Choose Your Own Adventure game (from Lab #3)
- Play Treasure Hunt game (from Lab #5)
- Play the Psychic game (from Lab #5)
- Play Tic-Tac-Toe (from this lab, Exercise #4)
- Exit

The WHILE loop will consist of the menu above and the IF statement.

However, this time, based upon the user choice, the main program will call a function that plays the game of choice. If the user chooses choice 1, the main program will call a function that plays the Math game, etc.

Make each game a function of its own!