CSci 1113
Lab Exercise 6 (Week 7): Arrays!

Arrays

Arrays are our first real look at *data abstraction*. An array is an *ordered collection* of data values, but it can be described as if it were a singular "thing". So, for example, a "deck" of cards might be represented using a 52-element *array* of characters or integers. We could subsequently pass this "deck" to a function that might "deal" the cards one by one.

Arrays or *lists* of data are integral to problem solving in every programming language. In this lab, we begin exploring this often-used and important concept.

Mystery-Box Problem

Here is your next mystery-box problem. What is the output produced by the following code segment?

```
int someArray[4][4], i, j;
for( i = 0; i < 4; i++ )
    for( j = 0; j < 4; j++ )
        someArray[i][j] = i + j;
for( i = 0; i < 4; i++ )
{   for( j = 0; j < i; j++ )
        cout << someArray[i][j] << " ";
    cout << endl;
}
```

Warm-up

1) **Simple 2D array**
Represent the following matrix using a two-dimensional array and write a nested for loop to initialize the elements **(do not initialize the array in the declaration):**

```
10   9   8

7    6   5

4    3   2
```

2) **Declaring/initializing arrays**
Download partner.cpp from the website (under the lab section). Currently, there are two variables for names (fullName1 and fullName2) along with two variables for heights (height1 and height2). Rewrite the code to use a single array for each of these pairs of variables (a single array for all names and another array for all heights). The code should give the use the same input/output as before.

3) **Passing arrays**
Copy-paste your code from warm-up 2 into a new file as a starting point for this problem. You probably have your cout statement in main() along with the array declaration. Make a new function for the cout statement while keeping the array declaration in main(). (Hint: Cut and paste the cout statement into this new function and pass the arrays from main into this function.)

Stretch

1) **Partially filled arrays**
Copy-paste your code from warm-up 3 into a new file as a starting point for this problem. Modify your main() function to allow any number of names and heights to be entered (instead of just 2). The output should tell you all the people's names entered and their combined height (in the same way the previous problem did it for two people). You will need to use a partially filled array to do this. (You can assume that there will not be more than 100 people.)

Example (user input is underlined):
How many people? 5
Please enter full name: AB
Please enter AB's height: 1
Please enter full name: CD
Please enter CD's height: 2
Please enter full name: EF
Please enter EF's height: 3
Please enter full name: GH
Please enter GH's height: 4
Please enter full name: IJ
Please enter IJ's height: 5
If AB and CD and EF and GH and IJ stand on top of each other, their combined height will be 15.

2) **Bubble Sort**
Sorting a list of numbers is an important Computer Science problem that has been extensively studied. One of the simplest methods is known as "bubble sort". Given a list of numbers:

3 5 2 8 9 1

the basic idea is to compare the first two numbers in the list and swap them if the first one is larger than the second (assuming you wish to sort from low to high). Next, the second and third numbers are compared and swapped if necessary, then the third and fourth, fourth and fifth, and so on until the entire list has been examined. After the first pass through, the list would look like this:

3 2 5 8 1 9

Note that the largest value will *always* end up in the last position after the first pass.

Next, we repeat the process, "bubbling" the larger values up in the list on each pass. Note however, that for the second pass we don't need to examine the last value because it's guaranteed to be the largest. After the second pass, the last *two* values need not be examined, and so on.

The process ends when an entire pass through the list results in no values being swapped.

Write a program that will implement the Bubble-Sort algorithm. For this problem you need to do the following:

- Declare an integer array named *list* that contains 50 values.
- Using a loop, initialize *list* with the values 100 , 99, 98, … (in decreasing order)
- Construct a void function named `bsort` that implements the Bubble Sort algorithm as described above. Your function should accept two arguments: the integer array to be sorted and the number of elements in the array.
- Call the `bsort` function to arrange the elements of *list* in increasing order.
- Print out the elements of *list*, 5 per line as follows:

Example:

```
51      52      53      54      55
56      57      58      59      60
        (and so on…)
```

[Hint: This problem will be much easier if you attack it in stages. First, write a `swap` function that will swap two integer values, then write another function that will make a single pass through the array, calling the `swap` function to correct any out-of-order elements. Finally, call this second function as many times as necessary to sort the array]

Workout

1) Steady State Temperature in a Thin Metal Plate

Imagine a thin metal plate surrounded by heat sources along each edge, with the edges held at different temperatures. After a short time, the temperature at each location on the plate will settle into a steady state. This can be modeled by dividing the plate into a discrete grid of cells and simulating the change in temperature of each cell over time:

100	100	100	100	100
90	*T*(1,1)	*T*(1,2)	*T*(1,3)	95
90	*T*(2,1)	*T*(2,2)	*T*(2,3)	95
90	*T*(3,1)	*T*(3,2)	*T*(3,3)	95
80	80	80	80	80

At each time step, the temperature in each interior cell (*i*,*j*) will be the average of four of its surrounding neighbors:

$$T(i,j) = (T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1))/4$$

a) Part 1

Develop a program to determine the steady state temperature distribution in the plate by representing the plate as a two-dimensional array with `NROWS` rows and `NCOLS` columns. `NROWS` and `NCOLS` should be declared as global *constants* (suggestion: use 20 for both values).

Your program should do the following:

- Declare 2 separate two-dimensional arrays named `temp` and `old.` Array `old` will be used to maintain the *current* values of the grid temperatures and array `temp` will be used to compute the *new* values obtained at each successive time step using the equation above.
- Prompt the user and enter temperature values for the top, bottom, left, right sides. Also prompt and enter an initial temperature T(i,j) for the interior cells. (For this lab, you may initialize all the interior cells to a single, user-input temperature.)
- Initialize `temp` using these values and display the initial contents of the `temp` array on the console.
- Obtain a convergence criterion from the user (say, 0.001)
- Use a convergence loop to continue updating the `temp` array until the temperature in every cell converges using the following method:
    1. Set a *boolean* variable named `steady` to `true`
    2. Copy `temp` to `old`
    3. Loop over all the *interior* cells (but not the edge cells!)

       `temp[i][j] = 0.25*(old[i][j-1] + . . .)`

       If `|temp[i][j] – old[i][j]| > convergence_criterion,`

       then set `steady` to `false`

    Repeat this 3-step process again and again until all the cells simultaneously satisfy the convergence criterion.

This is a longer program than usual, so here are a number of hints:

- Before writing any code think about what the code should look like: What will an outline look like? Will it involve loops? If so, how many, where, and of what type? Will there be if statements? If so, where and how many? What variables will your program need to keep track of? Which of these will be arrays? Think about these questions and discuss them with your partner before actually writing any code.
- Write, test, and debug your code incrementally, rather than all at once.
- Create a function named `void display(double temp[][NCOLS]).` You can use this function to print out that array at any time, and so the function will be useful in debugging your program.
- Make sure you can explain the difference between the `old` array and the `temp` array, and why your program needs to use both.
- Make sure you understand how to have your program loop through only the *interior* of the array.
- Make sure you understand what the convergence criterion test is, and what role it plays in your program.

b) Part 2 (optional)

Now you will *visualize* the steady state temperature data by creating a 3D plot using Matlab. Begin by modifying your program so that the final temperature data is written to a file named **`temp.dat`** instead of (or in addition to) the terminal screen. Then open Matlab by typing the following commands:

```
%module load math/matlab
%matlab
```

When the application opens, move the cursor to the Command window and enter the following Matlab commands in order and observe what happens:

```
nrows = 20
ncols = 20
[x,y]= meshgrid(1:nrows,1:ncols)
load temp.dat
mesh(x,y,temp)
surf(x,y,temp)
```

The mesh function produces a mesh or wireframe plot of the data. The surf function produces a surface plot. Try using the rotate button from the menu bar to have a good look at the data!

Check

Individually, list one important things you learned in lab today, one question you still have about the lab material, and one way arrays are used in your major. When you are done, share your list with your partner.

Challenge

Here is an extra challenge problem. You should try to complete the warm-up, stretch and workout problems in the lab. Try this challenge problem if you have extra time or would like additional practice outside of lab.

1) Tic-Tac-Toe, part 1

The game of Tic-Tac-Toe (or Naughts and Crosses) is an ancient game in which 2 players alternate turns placing either X's or O's on a 3x3 square grid. One player places X's on the grid and the opposing player places O's. The first player to place 3 consecutive X's (or O's) on any row, column or diagonal of the grid wins the game. If all 9 grid cells are occupied without either player winning, the game is declared a draw.

Write a C++ function that takes a 3x3 array of characters representing a tic-tac-toe game in progress and determines the current game state: player X has won, player O has won, the game is a draw, or none of those.

Your function should return a single character as follows: 'X' if player X has won the game, 'O' if player O has won the game, 'D' if the game is a draw (all cells occupied but neither player has won) and '*' if none of these conditions exist. (Hint: use a `for` loop that compares each element of the diagonal with the elements of its intersecting row and column, then check the diagonals)

Write a `main()` driver to verify that your function is correct. It should (1) declare a 3x3 character array, (2) use a loop to read in values for each of the 3 rows of the grid (use '-' to indicate a blank cell), (3) print out the contents of the grid as 3 rows of 3 values, (4) call your function with the array as an argument, and (5) output an appropriate message declaring the state of the game (as returned by your function).

Test your program using the 4 following cases (at a minimum):

```
X – O
X O –
X – O

O – X
X O –
X – O

- X O
X O –
X – O

O X O
X X O
X O X
```

2) Tic-Tac-Toe, part 2

Using the function from part 1, write and integrate several new functions to manage an interactive program that will play Tic-Tac-Toe against a human opponent.

You need to do the following:

a). Write a function named `displayGame` that takes the current game state array as input and displays it on the console as follows:

```
–  X  O
–  –  –
O  –  –
```

Make the output more "readable" by inserting a blank line after the last row of the game display.

b). Write a function named `opponentPlay` that takes the current game state array as input and returns the row and column of the next move that the opponent wishes to play. Your function should prompt the user to enter the row and column and verify that it is a legitimate move (1 <= row <= 3, 1 <= column <= 3). In addition, you should verify that the specified row/column has not already been played. The function should continue to solicit input until a valid row/column value has been entered.

c). Write a function named `nextPlay` that takes the current game state array as input and returns the row and column of the next move that the computer determines is the 'best' play. You may use any strategy you wish to determine the next move. A few possibilities:

a) Select the first unoccupied grid cell you find
b) Use the `pseudoRand` function you created in Lab 6 to select a random cell
c) Locate rows and columns that have not been played by the opponent
d) …etc

Note: ensure that you do not select a grid cell that has already been played.

d). Write a `main` program to play the game using the following algorithm together with the four functions you've created:

1. Input the opponent's move, update the board state array and display the updated board state.
2. Determine if the result of the opponent's move is 'win', 'draw', or 'continue'.
3. If opponent wins, then announce "Good game, you win", and end the program.
4. If 'draw', then display "The game ends in a draw" and end the program.
5. If 'continue', then determine the next computer move.
6. Display "I will play row *x*, column *y*" using the row/column values determined in the previous step.
7. Update the board state array with the computer's move and display it.
8. Determine the result of the computer's move.
9. If 'win', then announce "I win, nice try" and end the program
10. If 'draw', then announce "The game ends in a draw" and end the program
11. If 'continue', then repeat steps 1-10 until the program ends.

Example 1:

```
Welcome.  I'll play X, you will be O
You may play first. Enter row and column (1-3):
4 2
Sorry, that is not allowed.  Try again
Enter row and column (1-3):
0 2
Sorry, that is not allowed.  Try again
Enter row and column (1-3):
1 1
O  -  -
-  -  -
-  -  -

I'll play row 1, column 2
O  X  -
-  -  -
-  -  -

Enter row and column (1-3):
1 1
That space has been played.  Try again
Enter row and column (1-3):
2 1
O  X  -
O  -  -
-  -  -

I'll play row 3, column 1
O  X  -
O  -  -
X  -  -

Enter row and column (1-3):
2 2
O  X  -
```

```
O  O  -
X  -  -

I'll play row 2, column 3
O  X  -
O  O  X
X  -  -

Enter row and column (1-3):
1 3
O  X  O
O  O  X
X  -  -

I'll play row 3, column 3
O  X  O
O  O  X
X  -  X

Enter row and column (1-3):
3 2
O  X  O
O  O  X
X  O  X

Game ends in a draw
```

Example 2:

```
Welcome.  I'll play X, you will be O
You may play first. Enter row and column (1-3):
1 3
-  -  O
-  -  -
-  -  -

I'll play row 1, column 1
X  -  O
-  -  -
-  -  -

Enter row and column (1-3):
3 1
X  -  O
-  -  -
O  -  -

I'll play row 2, column 2
X  -  O
-  X  -
O  -  -

Enter row and column (1-3):
3 3
```

```
X  -  O
-  X  -
O  -  O

I'll play row 2, column 3
X  -  O
-  X  X
O  -  O

Enter row and column (1-3):
3 2
X  -  O
-  X  X
O  O  O

Good game, you win!
```