CISC220 Lab 2:

Due Wed, Sep 26 at Midnight (110 pts)

For this lab you may work with a partner, or you may choose to work alone. If you choose to work with a partner, you are still responsible for the lab in its entirety, even if you partner is abducted by aliens and mind melded with a two-toed sloth.

For this lab, you should still only have one file. As with Lab 1, your file should start with comments including your name(s), etc., followed by your included files, and then your function declarations in the order in which the functions are written in the lab.

Your main should follow, calling each of the functions and, if needed or helpful, initializing variables before function calls and printing out values and arrays.

Following the main should be each of your functions in the order in which they are specified below.

Make sure your code is formatted and commented.

Functions:

Note that for functions 1 through 7, you only need one test case.

1. (2 pts) Write a function that takes no input parameters and returns nothing (void). Inside the function, create a variable of type int, and give that variable a value (I'm not picky here). Print out both the value and the address of the variable.
2. (2 pts) Write a function that takes as an input parameter an integer and returns nothing (void). Add 4 to that integer inside the function. Inside the function, print out both the value and the address of that variable.
   In main, create an integer variable, and give it a value (your choice). Print out both the value and the address of that variable. In your comments, note what this type of function call this is.
3. (2 pts) Write a function that takes no input parameters and returns an integer. Inside the function, create an integer variable, and set the variable to be a random number between 0 and 50 (excluding 50). Print out the value and the address of this variable. Return this value from the function.
   Now in your main function, set an integer variable to this returned value. Print out both the value and the address of this integer.
4. (2 pts) Write a function that takes as an input parameter an address of an integer and returns nothing (void). Inside the function, take the value at the address in the parameter and cube it. Print out the value at the address, the address in the parameter, and the address of the parameter.
   In main, create an integer variable (and give it a relatively small number for testing purposes). Print out both the value and the address of that variable. Then call the function. Repeat the process of printing out the value and the address of this variable. In your comments note what type of function call this is.
5. (2 pts) Write a function that takes as an input parameter an alias for an integer and returns nothing (void). Inside the function, create a variable and set its value to a random number between 0 and 10. Add that random number to the input parameter. Print out the random number, the new value, and the address of the input parameter.
   In main, create an integer variable (and give it a relatively small number for testing purposes). Print out both the value and the address of that variable. Then call the function. Repeat the process of printing out the value and the address of this variable. In your comments note what type of function call this is.
6. (3 pts) In main, create 2 integer variables, setting the first to 10 and the second to 20. Now write a function that uses call by pointer with 2 int addresses as parameters, with the function returning nothing (void). In the function, create a variable that is of type int address. Set the variable to the address in the first parameter, and change the value at that address to 32 using the local variable (the one created inside the function). Then change the address in the variable so it holds the address in the second parameter. Change the value at that address to 42 using the local variable.

Call the function using the address of the 2 integers. Print out the values in the 2 variables after you call the function.

7. (5 pts) In your main, create a bunch of character variables and use them to print out a word (like we did in class with "google"). Now write a function that uses call by value, call by reference, and call by pointer (it doesn't have to be in that order) to modify the characters to new characters. Back in main, print out your new word (like "code"). Note that "google" and "code" have already been used and thus can't be used for this assignment.

8a. (3 pts) Write a function that swaps 2 integer values if the first is larger than the second, and doesn't swap them otherwise. It should return a Boolean value indicating whether the values were swapped or not. If the values were swapped, they should remain swapped outside the function as well as inside the function. So the function should have 2 input parameters. Note that there is more than one way to do this function!

8b. (3 pts) Write another function that loops 20 times. Each time, it generates 2 random numbers between 0 and 25. It should then print out those 2 numbers. It should then call function 8a. and, if the values were swapped in 8a, it should print out the swapped values. Note that you might want to indicate that the second print is a swapped print, so you know when the values were swapped or not

8. (6 pts) Write a function that takes as input parameters a length parameter (an int), and an int parameter that will be modified using pass by reference. When the function is called the second parameter is initialized to -1 (before the function call). The function should should generate a random array the length of the length parameter, with the numbers between 1 and 50. The function should print the array, and then locate the smallest value in the array, modifying the third parameter to be the smallest value. Make sure you print this value after you've returned from the function.
   Note: to print the array in a way that is nicely readable, you'd use a tab or a ", " instead of an endl
   So your line of code would look something like this:
   ```
   cout << arr[x] << ", ";
   ```
   And then when you're done printing out the array (after the loop), you'd then print out the endl
   ```
   cout << endl;
   ```
9. (6 pts) Write a function that takes an input parameter an int and returns nothing. It then generates an array of random numbers the length of the int parameter. It fills the array with random numbers between 0 and 50. It then prints out the array.
   Without creating a new array, the function then reverses the array and prints out the reversed array.
   So if the first array was:
   22,10,8,5,3,18
   The reversed array would be:
   18,3,5,8,10,22
10. (10 pts) Write a function that takes as an input parameter an integer that will represent the length of the array and a second integer that represents the range of numbers the random number should generate (in other words, if the number is 50, the random number generator should generate numbers between 0 and 49 including 49.
    The function then sorts the list by traversing the list, locating the smallest number, and printing out that smallest number, then replacing that number in the array with the second parameter +1 (so in our above example, it would be 51.) Continue to do this until every number in the original array is printed out in order.
11. (3 pts) Write a function that creates a new variable on the stack. Give the variable a value of 3. Print out the value and the address of the variable within the function. Return the address of this variable, and make sure that main has a pointer set to the returned variable (int *x = func();
    Print out the address of the variable and the value in the variable in main. Did this work? Did it compile? Include a comment on those 2 questions.
12. (2pts) Write a function that takes an array of integers as an input parameter (the address of the first value of the array). It returns nothing. It prints out the array as a single line, with commas between each number, and when the array is finished being printed, it prints an endl; so that we flush the buffer and move to a new line. (I'm having you write this function because you'll be wanting to print out arrays a lot in the following exercises).

Note: you'll be using this function after running other functions to check out your results.

12b.(2 pts) In main, write a loop that generates an array of random integers on the stack. The length should be 25, and the integers should be between 1 and 10. Use this array to test your function that prints out an array.
Note: there are a few functions below that require as an input parameter an array of integers. This is how the arrays are created, although the length and range may differ.

13. (5 pts) Write a function that takes as input parameters (using call by pointer) 3 integers. It generates a random number between 25 and 50 (not including 50). It then creates an array on the memory heap of that length. It generates a random high number between 5 and 10 and a random low number between -5 and -10 and fills in the array iteratively with random numbers between the high and the low numbers*, and it returns that array. The input parameters should be modified so that it holds the length of the array, the high value, and the low value. In main, call the function 12 to print out the array.
**not including the high – in general when we specify a range, we include the first value but not the last. If I forget to say that in the future, you can assume that's what I intended.*
14. (4 pts) Write a function that is almost exactly the same as the function above, only it takes an input parameter an integer (pass in a number between 25 and 50). Inside the function create an array on the stack instead of the heap. Fill it with random numbers as above. Return the address of the first value of the array, and then in the main use function 1 to print it out. This should NOT work. In comments explain why.
15. (3 pts) Write a function that takes as an input parameter an array of integers (and the size of the array). The function should print out the address of every value in the array.
16. (3 pts) Write a function that takes as an input parameter an array of doubles. The function should print out the address of every value in the array. See how this works?
17. (5) Write a function that takes as an input parameter an array of integers and the size of the array as a pointer. This function should go through the array and create a new array on the heap that removes all double-numbers that occur next to each other. The function should modify the arraysize parameter to the length of the new array with the multiple occurrences of numbers next to each other removed, and it should return the newly-created array. In other words, if the input array is:
`5,4,4,3,6 ,6,6,8,9,5,1,3,8,8,1,8,9,9,3,2,2,2,6,1 ,2`
The returned array would be:
`5,4,3,6,8,9,5,1,3,8,1,8,9,3,2,6,1,2`
And the size would be adjusted to:18

HANNING WINDOW:

For this next part, you will be writing a low pass filter using a very basic hanning window. The idea behind a low pass filter is to get rid of high frequencies, or, in essence, smoothing out the massive outliers in an array. The way to do this is to take a window size, usually an odd number in length, in which you take the average of the values before and after a value in an array and replace that value with the average. So, for instance, if you had an array as follows:

3,8,2,5,1,4,6,0,2

And you had a window size of 3, you would go through the array in window sizes of 3 and replace each center value with the average. For those values at the beginning and the end, in which you can't have a window on both sides, you'd replace the values with a 0. Otherwise each value gets replaced with its window average.

The resulting array would be:

0 4 5 2 3 3 3 2 0

Which is a much smoother array with fewer outliers.

Now, an even better way of smoothing out the outliers is to weight the window. So, in the process of averaging, the values farthest from the center value in the window are weighted the lowest, whereas the values closest to the center of the window are weighted the highest. This is known as a Hanning window.

For instance, if you have a window size of 5, you might weight the values by multiplying the values at location 1 and 5 with 1, the values at location 2 and 4 with 2 and the values at location 3 (the center) with 3, and then dividing the sum by 9 to get the weighted average. So in this case, given the following array and a window size of 5,

3,8,2,5,1,4,6,0,2

Using a hanning window for a filter, you'd get:

0, 0, 4, 3, 3, 3, 3, 0, 0,

(Note that this is even smoother than the really simple filter we used above).

So you can see where this is going:

18. (5 pts) Write a function for the hanning window. For this function, you should take as input parameters a part of the array (remember – whenever you pass in an array, no matter what you pass in, it is always interpreted as the address of the first value of the array. So if I passed in &arr[3], when that address enters the function, it will assume that the 4th address is the first address in the array. So DO NOT make a copy of each window and send that into the function – that is just wasteful and makes no sense). Assume the window size is an odd number in size, Weight the values appropriately. Return the weighted and averaged value.
19. (10 pts) Write a function for filtering the array. This function should return a new array. You can't write over the old array (why not?). You will have to create a new array on the heap, and return that new array. This function should take as input parameters the original array and the size of the original array, and should create a new array that has been filtered using the hanning window function.
    Print out both arrays to see and check the difference.
20. (12 pts) Write a function that takes as input parameters an array, the length of the array, the highest value in the array, and the lowest value in the array. The function should print out a graph of the array by printing out a * for each value in the appropriate place (below is an example of my printout, which is probably easier than explaining it:

```
Random Array generated:
6, -2, -4, 5, -3, -4, -3, -1, 5, 2, -2, 0, -7, 2, -3, -4, -3, -1, -5, -3, 1, 7, 3, -7, -7, 3,
-8, 1, -5, -4, -2, -5, -8, 0, -4,
Graph of the above array (printed using the function in 6):
8:
7:                      *
6:*
5:   *    *
4:
3:                        *  *
2:          *   *
1:                     *      *
0:            *                      *
-1:         *         *
-2: *         *                     *
-3:    * *        * *  *
-4:  *  *          *              *    *
-5:                    *         *  *
-6:
-7:             *          **
-8:                           *     *
Filtered Array (using the hanning window):
0, 0, 0, 0, -1, -2, -1, 0, 1, 1, 0, -1, -2, -2, -2, -2, -3, -2, -2, -1, 1, 2, 0, -2, -3, -3, -
3, -2, -3, -3, -4, -4, -4, 0, 0,

Graph of the filtered array (printed using function 6):
8:
7:
6:
5:
4:
3:
2:                      *
1:         **          *
0:****   *  *           *          **
```

```
-1:    * *    *       *
-2:     *      **** **    *   *
-3:                *       *** **
-4:                              ***
-5:
-6:
-7:
-8:
```

21. (10 pts) Write a function that takes as input parameters two integer addresses (both call by pointer). It returns the address of a 2-dimensional array.
    In the function, generate a random number between 5 and 10. That will be the size of the array of addresses (call it x). Then generate a second random number between 4 and 8. That will be the size of each array of integers (call it y). Adjust the input parameters to hold x and y.
    Create the 2-dimensional array (of x, y in size), making sure the 2-d array is on the heap.
    Fill this array with 0's. Then generate 5 random number pairs (between 0 and x, and between 0 and y) and place a 1 in each of those random number locations.
    Make sure that there isn't already a 1 in that location (and if there is generate a new x and y random number set).
    Return the 2 d array.
    In the main, use a for loop (looping x times) to call the print function (function 12) to print out the matrix.