CS2250 PROGRAMMING PROJECT 2:

In this project you will be working with arrays and pointers in order to become more comfortable dealing with them together.

This project is divided up into parts, with some dependency between them (certainly with the first part), so work on it incrementally.

Note: In this project all arrays need to be created INSIDE functions and then passed to the outside through the return statement or through reference pointers. I want no arrays created without dynamic allocation. *I will give you function prototypes that I expect you to use, note that you can re-use functions from previous parts.*

Task 1:
Your program should call a function that prompts the user for a number (1-10) and then dynamically allocates an array of that size. It should then take from the user integers to fill up that array. It should check to see if the original number is in the range of 1-10 and it should make sure that the following numbers are in the range [1-100] (1 and 100 are allowed). This function should then return that array through the return statement to the main body and the size of the array through a reference parameter.

In the main body, the array should be displayed on the screen. To do this, you should create a function to display that array.

When displayArray is called, it should display the array on one line, by itself, with a space following each number, surrounded by square brackets:

```
[1 5 13 3 7 ]
```

Prototypes for task 1:

```
int * generateAndFillArray(int & size);
void displayArray(const int *arr, int size);
```

Task 2:
Your program should call a function to create a copy of the array generated in part one. This function would take in an array and then inside dynamically allocate an array equal in size to it and then copy over all the elements. It should then return that array through the return statement. Once this is done, output both arrays to the screen using displayArray (yes, even the array you passed to it to get a copy, in order to verify that it was not changed accidentally).

Prototypes for task 2:

```
int * copyArray(const int *arr, int size);
```

Task 3:
Pass the array created in part 2 (the copy) that should sort ascending every EVEN element (so elements 0,2,4,…) using a modified selection sort that does not change the odd elements. When this is done, output the "sorted" array using your display array function. Then pass this array to a function that should bubblesort every ODD element, but this time sort descending. Then display the array after this also.

Prototypes for task 3:
void selectionsortAscendingTheEven(int arr[], int size);
void bubblesortDescendingTheOdd(int *arr, int size);

Task 4:
You just created an array that sorted the even elements and odd elements differently. Now I want you to pass this array to a function that will "split" the array up into two arrays. This will require that you pass the original array to a function and then return TWO arrays out through reference parameters (and their associated sizes). It should start by dynamically allocating an array large enough to hold all the odd elements, then putting the odd elements [1,3,5,7…] into its elements [0,1,2,3,…]. Then it should dynamically allocate an array large enough to hold the even and then putting the even elements [0,2,4,6,…] into its elements [0,1,2,3,…].

End this task by displaying all 3 arrays in the main body.

Prototypes for task 4:
void splitAnArray(const int *originalArray, int originalSize, int ** oddArray, int &numOdd, int ** evenArray, int &numEven);

Task 5:
You have many arrays at this point, but I want you to write a function that will "merge" the original array that you started with and the evenodd sorted array you created in task 3. So dynamically allocate in a function an array large enough to hold all the elements in both of these arrays. Then fill that array up with elements from those arrays, alternating. So element 0 of the array takes it from the very original array, then the next element from the "sorted" array, then the next element from the original array and so on. Return that array to the outside through the return statement. The size of this array can be calculated on the outside.

Display all 3 arrays, labeled as such, on the outside.

Prototypes for task 5:
int * mergeArrays(const int *firstArray, int firstSize, const int * secondArray, int secondSize);

Standards: Make sure to pay attention to proper programming style. Check the programming style supplement before beginning on this project. 20% of your final grade on this project is dependent on style and comments, do not ignore this.

Submit through canvas by attaching your source file. Do not copy/paste the text into the submission box.