

1 Introduction

Each exercise should be its own separate project.

Remember to follow the coding and documentation standards for the class listed on the MyClasses pages.

When you are ready to submit your work create a folder called Homework09 in that folder have separate folders for each project, one folder per project. Put all the code files needed for that project in its respective folder. Do not include the files that the IDE creates, I just want the code files. Zip the entire Homework09 folder up into a single zip file and submit it.

2 Exercises

In this homework you will be writing several array manipulation functions that are described below. These can all be put into one program (project) and you are to create a main that tests each of these functions. For each of these functions you are to use pointer manipulation only to accomplish the task, no use of bracket notation outside the initial creation of the new arrays. That is, to create the arrays you would of course use something like `int *A = new int[size];` but all other code must use pointers only. The main is to create all testing arrays dynamically, that is, all of them to be stored in the heap. It must also free the memory that is allocated so that there are no memory leaks.

1. Write a function that accepts an int array and the array's size as arguments. The function should create a copy of the array, except that the element values should be reversed in the copy. The function should return a pointer to the new array. For example, if the argument array is

1 2 3 4 5 6 7 8 9 10

the resulting array is

10 9 8 7 6 5 4 3 2 1

2. Write a function that accepts an int array and the array's size as arguments. The function should create a new array that is twice the size of the argument array. The function should copy the contents of the argument array to the new array and initialize the unused elements of the second array with 0. The function should return a pointer to the new array. For example, if the argument array is

1 2 3 4 5 6 7 8 9 10

the resulting array is

1 2 3 4 5 6 7 8 9 10 0 0 0 0 0 0 0 0 0 0

3. Write a function that accepts an int array and the array's size as arguments. The function should create a new array that is one element larger than the argument array.

The first element of the new array should be set to 0. Element 0 of the argument array should be copied to element 1 of the new array, element 1 of the argument array should be copied to element 2 of the new array, and so forth. The function should return a pointer to the new array. For example, if the argument array is

1 2 3 4 5 6 7 8 9 10

the resulting array is

0 1 2 3 4 5 6 7 8 9 10

4. Write a function that accepts an int array and the array's size as arguments. The function should create a new array that is the same size as the argument array. The first element of the new array should be set to the last element of the argument array. Element 0 of the argument array should be copied to element 1 of the new array, element 1 of the argument array should be copied to element 2 of the new array, and so forth. The function should return a pointer to the new array. For example, if the argument array is

1 2 3 4 5 6 7 8 9 10

the resulting array is

10 1 2 3 4 5 6 7 8 9

5. Write a function that accepts two int arrays and their sizes as arguments, sizes can be different. The function should create a new array that is the sum of the sizes of the argument arrays. The first element of the new array should be set to the first element of the first argument array. The second element of the new array should be set to the first element of the second argument array. The third element of the new array should be set to the second element of the first argument array. The fourth element of the new array should be set to the second element of the second argument array, and so forth. That is, the new array is an interlacing of the two. When one array runs out of space the remaining elements of the other are put on the back of the new array. The function should return a pointer to the new array. For example, if the argument arrays are

1 2 3 4 5 6 7 8 9 10

and

83 86 77 15 93 35 86 92 49 21 62 27 90 59 63

the resulting array is

1 83 2 86 3 77 4 15 5 93 6 35 7 86 8 92 9 49 10 21 62 27 90 59 63