Lab 1: DArray

The project **lab_01_darray** contains an implementation of the class **DArray**, which creates objects containing a pointer to a dynamic array and two integers denoting the capacity and the number of elements in the array. Complete the project by implementing the functions listed below; write the **declarations** in the **DArray.h** file and the **definitions** in the **Functions.cpp** file.

- Function **emptyArray**
    - Resets the array to zero elements.

- Function **append**
    - **Parameters:** an object of the **DArray** class
    - Does not return a value.
    - Append all the elements of the dynamic array parameter to the calling object. If the calling object's capacity is too small to fit all the elements, delete the dynamic array to which the pointer of the calling object is pointing and re-create a new one.
    - <u>**Be efficient**</u>: Do not execute the body of the function if the array parameter has no elements.

- **Move constructor** and **move assignment operator**
    - The **Big Three** can be expanded to the **Big Five**, which include a constructor and an overloaded assignment operator that "steal" all the data from the parameter object.
        - The **move constructor** does not allocate new memory (like the copy constructor does), but it takes the memory already assigned to the parameter object, and it then sets the pointer stored in the parameter object to nullptr (this way, when the parameter object is eventually destroyed, the data to which the pointer stored in the calling pointer is pointing will not be destroyed).
        - The move assignment operator works in the same manner. The function deletes all the data to which the pointer in the calling object is pointing, and it re-sets it to point to the data to which the pointer in the parameter pointer is pointing. After "moving" the data, the pointer of the parameter object is re-set to nullptr.
    - **Parameters:** Since these two functions overload respectively the copy constructor and the overloaded assignment operator, you will need to make them different from the traditional ones by passing the parameters as double reference (&&)—note that the syntax for the implementation is the same as if it were single reference.
    - The **call statement** to these functions needs to use the move function from the STL header <utility> (the move function from the STL <algorithm> header overloads this function). Note that the move function does not actually move anything, but it acts as a helper function.
    - You may find some help by browsing the link below, but you will still need to think carefully about your implementation.
      https://msdn.microsoft.com/en-us/library/dd293665.aspx

Testing cases for all functions can be found in the Main.cpp file.

*** You may modify the Main.cpp file by adding more test cases), but do NOT modify the rest of the code that is provided. ***