OrderedDataBuffer subclass to DataBuffer

In this assignment, you are to create a subclass of the `DataBuffer` from the prior assignment. A subclass uses the original implementation and extends it in some way. In this case, the subclass you are to create will add the property that the data in the buffer is always in order, and therefore is called `OrderedDataBuffer`. In order to make the original `DataBuffer` class be usable by a subclass, you must make only one change to your original `DataBuffer` class. That change is to change the declarations of the buffer array and length (or count) integer in DataBuffer.h from "private" to "protected." Everything else in your `DataBuffer.h` and `DataBuffer.cpp` files should be exactly the same as in the buffer assignment. You should use the same code from the previous `main()` as well, to show that the `DataBuffer` class still works as before.

The `OrderedDataBuffer` subclass must add one new public member function, called `add`, that takes a single integer argument called `value`. The `add` method will add the new `value` to the data buffer, thus increasing its `length` by one. However, the new value cannot just go at the end. The insert method must find the correct position within the array where the value can be added, while preserving the property that the array is in smallest to largest order, and then insert the value at that location. It must also take care not to let the buffer get longer than its size.

In order to insert a value into the middle of an array, every element starting at that location and continuing until the end of the array, must be moved one location higher in the array. Create a separate "helper" function, called `shiftUpOne`, to perform that task. The `shiftUpOne` method should take one argument which is the index of the location to be vacated so that the new value can be stored at that location. Since the `shiftUpOne` is a helper function and should not be invoked by itself, make the `shiftUpOne` member function protected, rather than a public. When you implement the `shiftUpOne` operation, make sure that you do not overwrite a value in the array before it has been moved to a new location. Each move has to be from an occupied location to a safely unoccupied location in the array. In other words, it matters in which order you do the moving. After it is done moving data, the length should be increased by 1.

Once you have the new `add` member function working correctly, then you should also write a new version of the `copyData` member function. In this version of the function instead of just copying the data over, it should use the `insert` method to put each new value into ordered positions. Taking an unordered array and copying it into an ordered array in this manner is called *Insertion Sort*.

In all of the new operations, remember the buffer properties from the Buffer assignment. The buffer is a reserved amount of space allocated as an array. So there are two separate measures of the length. One is for the amount of space available (the size of the array) and the second is for the amount of space that is occupied (the count or length of the data). Also, note that because the data in the buffer is never not sorted, there is no need to implement a separate "sort" function.

For 1 extra credit point, add one more new method called `remove`, which takes one argument for the value. The `remove` method will remove all elements from the buffer that have the given value. To implement the remove function, you will need another helper function, called `shiftDownOne`, to move data into the cell being vacated before shortening the count by 1. Implementing this method to vacate a single element is not complicated. However, it becomes a bit more complicated when more than one element of the array has that value. Your code must

take into account that the array itself changes after each remove operation. You will see that when you run the test code below. If you do not implement the extra credit, comment out that section of the `testOrderedDataBuffer` function, below.

```
void testDataBuffer(int arr[], int length);
void testIterableDataBuffer(int arr[], int length);
void testOrderedDataBuffer(int arr[], int length);

int main() {
    const int ARR0_LEN = 2;
    int arr0[ARR0_LEN] = {-2, -1};
    const int ARR1_LEN = 10;
    int arr1[ARR1_LEN] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    const int ARR2_LEN = 25;
    int arr2[ARR2_LEN] = {2, 4, 6, 8, 10, 12, 14, 16, 7, 6, 22, 8,
        9, 16, 5, 2, 7, 8, 12, 2, 0, 14, 17, 19, 22};
    testDataBuffer(arr0, ARR0_LEN);
    testDataBuffer(arr1, ARR1_LEN);
    testDataBuffer(arr2, ARR2_LEN);
    testOrderedDataBuffer(arr1, ARR1_LEN);
    testOrderedDataBuffer(arr2, ARR2_LEN);
    return 0;
}

void testDataBuffer(int arr[], int length) {
    DataBuffer buf;
    buf.copyData(arr, length);
    cout << endl; // blank line
    data.print();
    cout << "Sum " << buf.sum() << endl;
    cout << "Min " << buf.min() << endl;
    cout << "Max " << buf.max() << endl;
    cout << "Range " << buf.range() << endl;
    cout << "Mean " << buf.mean() << endl;
}

void testOrderedDataBuffer(int arr[], int length) {
    OrderedDataBuffer obuf;
    obuf.copyData(arr, length);
    cout << "\n Ordered data buffer" << endl;
    obuf.print();
    obuf.add(0);
    obuf.add(5);
    obuf.add(13);
    cout << " Ordered data buffer after inserts" << endl;
    obuf.print();
    // section below for extra credit
    obuf.remove(2);
    obuf.remove(8);
    cout << "\n After extra credit removes" << endl;
    obuf.print();
}
```

Turn in all 5 files in a single zip folder. The files are the .cpp with your main(), DataBuffer.h, DataBuffer.cpp, OrdereredDataBuffer.h, and OrderedDataBuffer.cpp. Remember that the OrderedDataBuffer files should be implemented as a subclass of the DataBuffer class and only contain the bits that are new or overridden (2 public member functions and 1 protected helper function without the extra credit, 3 and 2 with the extra credit). Look in the book for how to implement a subclass. In Visual Studio, the add -> class form has a field where you specify the "base class". (Other than that, you declare and implement just the new code.)

The output from the above code is shown below.