Data Buffer

The next three assignments are linked. In this assignment you will create a buffer class that uses arrays and provide some basic operations on arrays. In the next two assignments you will add more methods and implement the ability to sort the contents of the buffer, using pointers.

First, read the note about arrays in C++. Pay special attention to the concept of a buffer. A buffer is an area of memory, typically implemented as an array, that is used to hold data when the size of the data is not known ahead of time. The buffer must be larger than any expected data. With a buffer, you need two length variables, one for the physical size of the buffer, typically called `bufferSize`, or just `size`, and one for the `length` of the data in the buffer (sometimes also called "`count`"). Operations on the data use the latter, length variable. Operations that load data into the buffer must not let the length exceed the size.

Implement a class called `DataBuffer`. The `DataBuffer` class has three properties (also called member variables), a constant for the size of the buffer called `BUFFER_SIZE`, an array of ints, called `buffer`, that can hold `BUFFER_SIZE` integers, and a variable called `length` for the amount of data in the buffer. Make `BUFFER_SIZE` 256. Note that the buffer array is defined directly in the class. There is no "`new`". Give the class a constructor, and the following 7 member functions: `copyData`, `print`, `sum`, `mean`, `maxValue`, `minValue`, `range`. The `copyData` function should return a `bool`. In C++ the Boolean type is called, bool. `Sum`, `maxValue`, `minValue`, and `range` should return ints. Mean should return a double.

In the implementation of the constructor, set the length to 0.

The `copyData` member function takes two arguments, an array of ints and the length of the array. It must contain a loop that copies the values from the elements of the passed array to elements of the buffer. It must also update the length (or count) value to indicate the amount of data now in the buffer. If some of the data was not copied, `copyData` should return false. Otherwise, it should return true. Also, note that the `copyData` should replaces any data in the buffer. It should not add to existing data.

The `print` method prints the values in the array to the console, with white space between the values being printed. The print method should print the values in rows, with ten values in each row - less if it is the last row. Do not use C-style print commands. You must use `cout` from the iostream library, and `setw()` from the iomanip library to create uniform column widths.

The `sum`, `maxValue`, and `minValue` methods should return the sum, highest value, and lowest value in the array. Each should return an int. In all three cases, the method uses a result or helper variable to hold the computed value as it evolves. If you have not seen these functions before, note that the way to find the max is to start with an initial value and compare it to each value in the array. Every time you find a value that is higher, replace the known max with the new value. Don't forget that for the min and max, start with an initial min or max value from the array. You cannot start with 0 and assume that the maximum value will be positive. When you create a variable for the sum, you must initialize it to 0. C++ does not initialize variables by default.

The mean and range methods should not contain any loops. Instead, they should have expressions using calls to sum, min, and/or max to calculate the mean and range. The range should return an int, while the mean should return a double.

All methods that return a result should return 0 when the buffer is empty (length is 0).

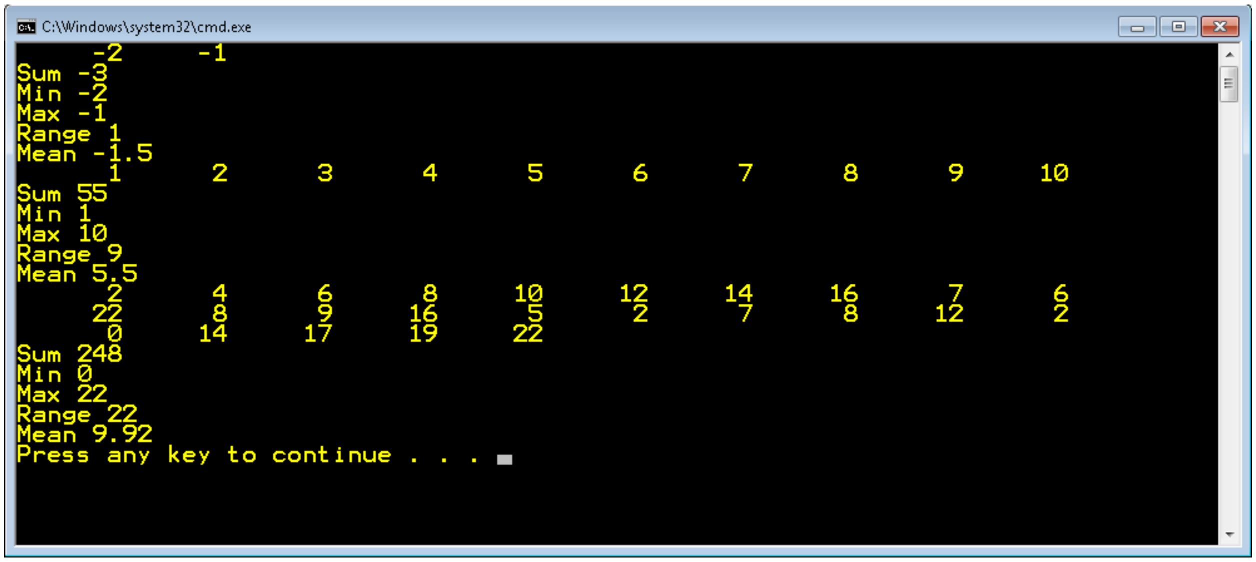
Please use descriptive names for variables (except i) and use ++ and += operators where appropriate (not `i = i + 1`).

Here is an implementation of the main and sample output.

```
void testDataBuffer(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);
    return 0;
}

void testDataBuffer(int arr[], int length) {
    DataBuffer buf;
    buf.copyData(arr, length);
    buf.print();
    cout << "Sum " << buf.sum() << endl;
    cout << "Min " << buf.minValue() << endl;
    cout << "Max " << buf.maxValue() << endl;
    cout << "Range " << buf.range() << endl;
    cout << "Mean " << buf.mean() << endl;
}
```



```
     -2     -1
Sum -3
Min -2
Max -1
Range 1
Mean -1.5
      1      2      3      4      5      6      7      8      9     10
Sum 55
Min 1
Max 10
Range 9
Mean 5.5
      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
Sum 248
Min 0
Max 22
Range 22
Mean 9.92
Press any key to continue . . .
```