Building on previous homework by modifying last week's source code

A. Class Point:
    a. Continue with declaration and implementation of class Point
        i. Make following changes to member functions:
            1) Change the member function name from *distance_to_orig* to *length*
            2) Remove the function *distrance_to_point*
        ii. Overload the following operators as <u>member</u> operators:
            1) Unary `operator-`
            2) Binary `operators: +=, -=`
        iii. Overload the following operators either as member or non-member
            1) Binary `operators: +, -`
            2) Binary `operator/`
               this operator takes two Point objects and returns the ratio of their length
            3) Binary `operators: >, <`
               these operators compare the lengths of two Point objects
            4) `operator<<`
               this operator streams out the coordinates of a Point object
            5) `operator>>`
               this operator streams in the coordinates of a Point object.
        iv. Non-member function:
            1) double dot(const Point&, const Point&);
               it calculates the dot product of two 2d vectors
    b. Test the operators you overloaded above in main function:
        i. Simple test:
            1) Use std::cin to read in 2 Point objects, p1 and p2, from console with the operator>> overload of Point. Use std::cout to print out the two points you just read in with the overloaded operator<<
            2) Print out the following information:
                a. `-p1` and `-p2` (use the unary operator-)
                b. `p1+p2`
                c. `p1-p2`
                d. `p1/p2`
                e. `dot(p1,p2)`
                f. result of Boolean expressions `p1>p2, p1<p2`
                g. the distance between p1 and p2 using only Point member function
        ii. Read and sort an array of Point objects according to their distance to origin:
            1) You are provided an array of x, y coordinates in "xy_data.txt". Read these x, y pairs (there are 10 pairs) into an array Point using while loop, make sure you use the overloaded operator>> to read one Point at a time.
            2) Sort the above array:
                a. Ask the user whether to sort these points according to their distance to origin in ascending or descending order.
                b. Use the sort algorithm you wrote in a previous homework to sort these points according to user's choice.

c. Print out the sorted array

3) Ask user to input another point, find the closest point in the array to the user input and print out both the coordinate and its distance to user input.

B. Class Stock:

a. Private data members with initialization:

i. `double *Prices=nullptr;` //dynamic array to hold price series

ii. `int nPrices=0;` //number of prices in price series

iii. `double *Returns=nullptr;`//dynamic array to hold calculated price return series

Mathematically, price return is calculated from prices as $r_i = 1 - \frac{P_{i-1}}{P_i}$

iv. `double avgReturn;` //stores calculated result

v. `double returnVol;` //stores calculated result

b. Special member overrides:

Inside the implementation of each of the following override, using a 'cout' statement to print out a short description of the special member. For example, inside the default constructor, add a statement: cout << "default constructor called\n";

i. Default constructor

ii. Parametrized constructor: use a user provided data series to

1) fill the Prices dynamic array
2) calculate and fill the dynamic Returns series
3) calculate average return and save the result in `avgReturn`
4) calculate the return volatility and save the result in `returnVol`

iii. Destructor

iv. Copy constructor

v. Copy assignment operator

vi. Move constructor

vii. Move assignment operator

c. Interface functions (make sure you use the keyword '`const`' properly)

i. Price series setter:

```
void setPrices(const double *p, const int& n);
//
```

1) copy values from a user supplied price series to internal prices
2) calculate and fill the dynamic Returns series
3) calculate average return and save the result in `avgReturn`
4) calculate the return volatility and save the result in `returnVol`

```
//
```

ii. Getters:

1) Read-only function `getNPrices` that returns the length of the price series
2) Read-only function `getPrices` that returns the price series in a read-only fashion.
3) Read-only function `getReturns` that returns the return series in a read-only fashion.
4) Read-only function getAvgReturn that returns avgReturn
5) Read-only function getReturnVol that returns returnVol

d. Overloading operators. You need to decide the nature of overload: (a) member, (b) friend non-member, or (c) non-friend non-member
    i. `operator<<` : to output the following data members:
        1) nPrices
        2) avgReturn
        3) returnVol
        4) information ratio: avgReturn/returnVol
    ii. Binary `operator-` : calculate the over/under-performance of one stock over another, i.e. the difference between the avgReturn of two stocks
    iii. Binary `operator+` : generate a synthetic stock where the price history is the sum of the two stock prices. It returns the synthetic stock as a Stock object
e. Test your class with provided data series:
    i. Read data series TS_A and TS_AMZN into your program (use the data provided in previous hw)
    ii. Initialize a Stock object *A* and *B* with data series TS_A and TS_AMZN respectively
    iii. Initialize a Stock object *C* with A
    iv. Declare an empty Stock object D, assign C to D
    v. Declare an empty Stock object *E*, assign synthetic stock *A+B* to E, i.e. E=A+B
    vi. Initialize a Stock object *F* with *std::move(C)*
    vii. Use cout statement to print out the following:
        1) Object *A*
        2) Object *B*
        3) Result of *A-B*
        4) Object *A+B*
        5) Object *E*

Compile your source code, run your programs, take screen shots. Submit everything online.

Happy coding and Happy Thanksgiving!