CS 202 – Assignment #3 **Fall 2023**

In investing, what is comfortable is rarely profitable. - Robert Arnott

(Quote source: https://trwsb.wordpress.com/2020/11/12/fear-greed/)

Purpose: Learn class inheritance, composition, multi-file class implementation, and make utility.
Points: 100

Assignment:

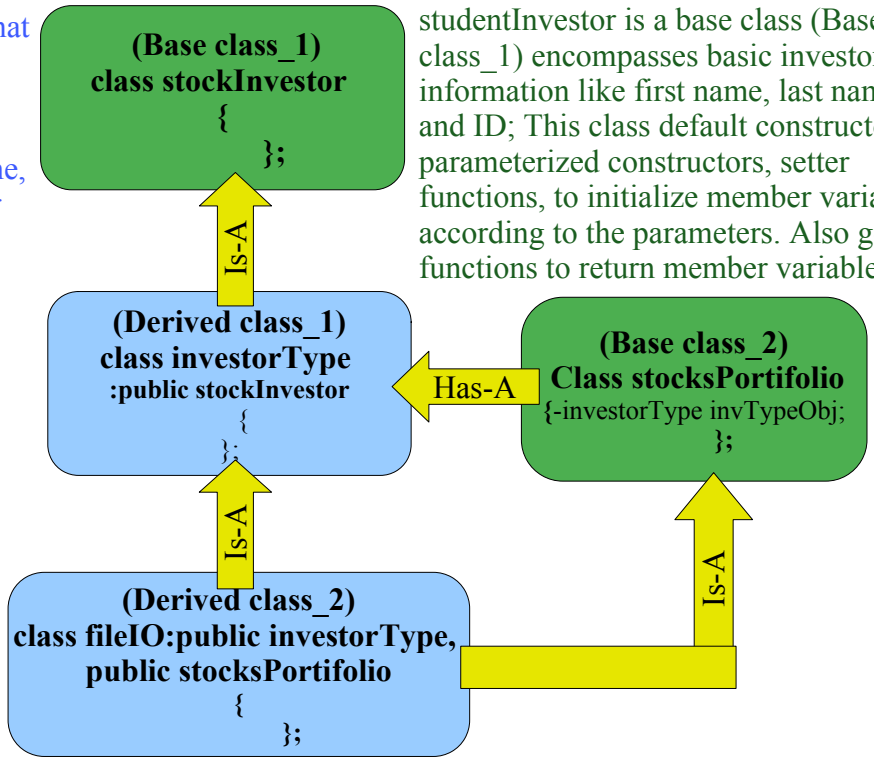
Design and implement four C++ classes- stockInvestor, investorType, stocksPortfolio, and fileIO. The assignment outcome is to generate an investors' portfolio report that shows their profit and or loss in their investments. In stocksMain.cpp, the objects of multiple classes are declared. With the use of constructors, setters, getters, we implement the functions to write the investors' data to a investorsData.txt file. During this process, the portfolio report is printed on terminal. Using fileIO class, read the data from investorsData.txt file, calculate profits and loss. Finally, write the portfolio reports to investorsPortfolio.txt file.

Image Source: https://www.dreamstime.com/vector-cartoon-stock-market-cycles-phases-investors-buy-sell-panic-funny-drawing-financial-graph-background-image165021067

Class Hierarchy

The following diagram should help understand the class hierarchy for this project.

investorType is a derived class (Derived class_1) that has functions like:
default constructor, constructor(parameters), setter (parameters) functions, getter functions(), and printData(ostream &). This class includes typeName, stockName, stockCount, and stockPrice as member variables. It inherits first name, last name and ID from the base class stockInvestor. This class has constructors and setters to set the base class and its own class private member variables.
writeDataToFile(ostream &, investorType &) use getter functions to write the investors data to investorsData.txt file.



studentInvestor is a base class (Base class_1) encompasses basic investor's information like first name, last name and ID; This class default constructor, parameterized constructors, setter functions, to initialize member variables according to the parameters. Also getter functions to return member variables.

stocksPortfolio is a base class (Base class_2) that has functions like: default constructor, constructor(parameters), setter (parameters) functions, getter functions(), writePriceChangesToFile(ofstream &), and printPriceChanges(ostream &). This class includes priceOne, priceTwo, priceThree, stockValue, an object (invTypeObj) of investorType (Composition: has-a relationship) as member variables. With composition, the object(s) of investorType class only can access to investorType member functions. The object of stocksPortfolio can not directly access the member functions of investorType class. You write the price changes (3 prices) of each investor's stock to investorsData.txt file. printPriceChanges(ostream &) prints all the three prices on the terminal or to the output file according to the parameter passed.

fileIO is a derived class (Derived class_2) that has a single function like:
unsigned int readWriteDataFile(ifstream &inFile, ofstream &outData); The function reads investorsData.txt as an input file; After reading the parameters from file, use investorType constructors(parameters), and stocksPortifolio constructors(parameters) to set parameters accordingly. Finally, calculate stockValues with original stock price, and three more price changes. Finally, write all the calculations, and the detailed report to investorsPortifolio.txt file. The function returns the number of entries(rows).

The classes we will implement are as follows:

- **stockInvestor Class**
  - Implement a basic ***stockInvestor*** class(Base class_1), will track investors basic information and provide standard support functions for all investors. This will include first name, last name, and ID. This class has a default constructor, parameterized constructor, setter and getter functions. To set the ID, you call setInvestorID(string id); In the implementation of setInvestor(ID) function, you call bool checkInvestorID(string tempID). The bool function returns true if the tempID meet the requirements else false. When the checkInvestorID(tempID) returns false to setInvestorID(string id), then this function prints an error message:
    Example: Error :- 318345 is invalid investorID for Dennis
  - *stockInvestor* will be the *base class* as this will apply to any investor (Type- Personal or Trader). The investorType class will inherit this functionality, so this class must be fully working before working on the investorType.
  - print(ostream &output): It prints InvestorID, firstName, lastName. It prints the above details on the terminal if the parameter is cout. It prints the above details to an output file if the passed parameter is output. See the sample output for more details. This function doesn't print stars and dashes.

- **investorType Class**
  - Implement a class, **investorType** (Derived class_1), that contains typeName, stockName, stockCount, stockPrice as member variables. This class can set all above using setter functions individually. Along with the above member variables, every investor have the following details such as firstName, lastName, ID. The parameterized constructor and setter functions are useful to set all the parameters of an investor. For this, you call base class (stockInvestor) constructors and setter functions as needed.
  - The getter function(s) return the member variable accordingly. As the investorType is inherited from stockInvestor, the object of investorType can access its own member functions and also member functions of stockInvestor(base class). In stocksMain.cpp, the objects of base classes and derived classes are declared. Few objects are declared with parameterized constructors and few are declared with no parameters. The object with no parameters can call setter functions to initialize investors data according to the parameters.
  - You write the data of each investor such as firstName, lastName, ID, typeName, stockName, stockCount, stockPrice to investorsData.txt file through its member function: writeDataToFile(ofstream &outFile, investorType &tempObj). [Hint: You call the getter functions with tempObj of investorType.
  - print(ostream &output): It prints firstName, lastName, ID, typeName, stockName, stockCount, stockPrice. It prints the above details on the terminal if the parameter is cout. It prints to an output file if the passed parameter is output. See the sample output for more details to match with stars and dashes.

- **stocksPortfolio Class**
  - This ***stocksPortfolio*** class is a base class (Base class_2). This class includes priceOne, priceTwo, priceThree, stockValue, an object (invTypeObj) of investorType (Composition: has-a relationship) as member variables. With composition, the object(s) of investorType class only can access to investorType member functions. The object of stocksPortfolio can not directly access the member functions of investorType class. You write the price changes (3 prices) of each investor's stock to investorsData.txt file.
  - printPriceChanges(ostream &) prints all the three prices on the terminal or to the output file according to the parameter passed.
  - stocksPortfolio(double pOne, pTwo, pThree, investorType invTemp) is a parameterized constructor. Initialize the member variables according to the parameters passed. The approach is similar for the setter function that has all the above parameters.
  - calculateStockValue(investorType invTemp, ostream &output): This is to calculate the stockValue on every price change
    - Multiply stockPrice with stockCount (Here stockPrice is the initial price of stock) (both from investor type; use getters)
    - Multiply stockCount with each new price and print statements accordingly
    - Finally print stockValue: Show Loss/Profit accordingly
    - call printPriceChanges(ostream &output)
    - Check sample output for more details
  - writePriceChangesToFile(ostream &output) const: Write priceOne, priceTwo, and priceThree to the investorsData.txt file. Check sample output, the prices should be in same line as id, name, etc.
  - printPriceChanges(ostream &output) const: It prints priceOne, priceTwo, and priceThree. It prints the above details on the terminal if the parameter is cout. It prints to an output file if the passed parameter is output. This function doesn't include stars and dashes. This function must be called in calculateStockValue(investorType invTemp, ostream &output).

- **fileIO Class**
  - Implement a small class, ***fileIO*** class(Derived Class_2). This class is inherited from investorType and stocksPortfolio through public access specifier. This class has no member variables and it contains only one member function.
  - unsigned int readWriteDataFile(ifstream &inFile, ofstream &outData): This function does read and write operations on files. Declare local variables as needed. It reads data from investorsData.txt file; declare objects of investorType and stocksPortfolio; call constructors or setters to set the parameters that are read from text file; Call the functions print(parameter) and calculateStockValue(parameter); These functions must write the investors' portfolio (report) to investorsPortfolio.txt file similar to the output that is printed on Terminal. This function must print the total entries of investorsData.txt file that includes invalid data lines. Also, this function returns that total entries.
  - In stocksMain.cpp, only one fileIO fObj is declared. You call the readWriteDataFile(parameters) with that object. If you observe here, you read every line from investorsData.txt file one after other; declare objects of investorType and stocksPortfolio then initializing parameters through constructors and setters accordingly; You use the same objects to read the next line and so you overwrite on the previous data. This saves more memory space. However, you don't have separate allocation for each investor and so you can not make changes flexibly, or you can't access the data of each investor dynamically. However, with the method we followed to declare objects in stocksMain.cpp, gives more flexibility to make changes on each investor. In short, both approaches have their own pros and cons.

Inheritance, composition and fundamental concepts are important in object oriented programming and it is essential to fully understand. Refer to the textbook and class lectures for additional information on inheritance and composition.

Development and Testing

In order to simplify development and testing, the project is split into four parts: *stockInvestor, investorType, stocksPortfolio,* and *fileIO* classes.

- The ***stockInvestor*** can be developed first and this class can be tested independently of the other class. This class sets firstName, lastName, and ID.
- The ***investorType*** (Inheritance: is-a relationship) must be developed next (before ***stocksPortfolio***). This class is inherited (public) from ***stockInvestor (base class).*** This class can be tested after implementing base class but has no dependency with other classes. This class has typeName, stockName, stockCount, and stockPrice as member variables.
- The ***stocksPortfolio*** (Composition: has-a relationship) must be developed next. This class has parameters like priceOne, priceTwo, priceThree, stockValue, an object (invTypeObj) of investorType (Composition: has-a relationship) as member variables. The above parameters are set using stocksPortfolio functions and the investorType member functions can be accessed with the object of investorType only (composition). So, you have to implement investorType before you implement this class.
- The ***FileIO*** (Inheritance: is-a relationship) must be developed last as this is derived from *investorType* and *stocksPortfolio* classes. This class has only one member function with two parameters. Parameter one is ifstream, to read from investorsData.txt file. Parameter two is ofstream, to write the investors' portfolio reports to investorsPortfolio.txt file.

Submission:

- All files must compile and execute on Ubuntu and compile with C++11.
- Submit source files
  - Submit a copy of the **source** files via the on-line submission.
  - *Note*, do ***not*** submit the provided header files and stocksMain.cpp (we have them).
- Once you submit, the system will score the project and provide feedback.
  - If you do not get full score, you can (and should) correct and resubmit.
  - You can re-submit an unlimited number of times before the due date/time.
  - Check the maximum attempts you can submit within an hour.
- Late submissions will be accepted for a period of 24 hours after the due date/time for any given lab. Late submissions will be subject to a ~2% reduction in points per an hour late. If you submit 1 minute - 1 hour late -2%, 1-2 hours late -4%, … , 23-24 hours late -50%. This means after 24 hours late submissions will receive an automatic 0(zero).

Program Header Block

All program and header files must include your name, section number, assignment, NSHE number, input and output summary. The required format is as follows:

```
/*
Name: MY_NAME, NSHE, CLASS-SECTION, ASSIGNMENT
Description: <per assignment>
Input: <per assignment>
Output: <per assignment>
*/
```

Failure to include your name in this format will result in a loss of up to 5%.

Code Quality Checks

A C++ linter[1] is used to perform some basic checks on code quality. These checks include, but are not limited to, the following:

- Unnecessary 'else' statements should not be used.
- Identifier naming style should be either camelCase or snake_case (consistently chosen).
- Named constants should be used in place of literals.
- Correct indentation should be used.
- Redundant return/continue statements should not be used.
- Selection conditions should be in the simplest form possible.
- Function prototypes should be free of top level *const*.

Not all of these items will apply to every program. Failure to to address these guidelines will result in a loss of up to 5%.

stockInvestor Class

Implement a basic *stockInvestor* class to provide basic information of an investor. The *stockInvestor* UML class diagram is as follows:

stockInvestor
-firstName, lastName, ID: string
+stockInvestor()
+stockInvestor(string fName, string lName, string id)
+setStockInvestor(string fName, string lName, string id): void
+setInvestorID(string id): void
+checkInvestorID(string investorIDTemp) const: bool
+setName(string fName, string lName): void
+getID() const: string
+getFirstName() const: string
+getLastName() const: string
+print(ostream &output) const: void

Function Descriptions:

The following are more detailed descriptions of the required functions.

- stockInvestor() is a default constructor that initializes firstName, lastName, and ID.
- stockInvestor(string fName, string lName, string **id**); This parameterized constructor is used to initialize the member variables according to the parameters in the constructor body.
- **void** setStockInvestor(string fName, string lName, string **id**); This setter function is used to initialize the member variables according to the parameters.
- **bool** checkInvestorID(string investorIDTemp) **const**;
  - return false if the ID is less than 6 characters

  Check the following conditions for an investors ID:
  - return false if the first character is not upper case
  - return false if the first character is not 'T' or 'P'.
  - return false if the remaining characters are not the numbers between 0 to 9.
  - return true if all the conditions are met.
- **void** setInvestorID(string **id**); if checkInvestorID(id) returns false print an error message and assign the id

  Example: Error :- 318345 is invalid investorID for Dennis
  if checkInvestorID(id) returns true assign id to ID(member variable)

- **void** setName(string first, string last); This function is to set firstName and lastName according to the parameters.
- string getID() **const**; This function is to return the ID.
- string getFirstName() **const**; This function is to return the firstName.
- string getLastName() **const**; This function is to return the lastName.
- **void** print(ostream &output) **const**; This function is to output the firstName, lastName and ID. Prints on terminal if cout is the parameter. Prints on to output file if the parameter is output.

investorType Class

Implement the Derived class_1 named ***investorType*** which is inherited from *stockInvestor* class. This class deals with investor details like ID, firstName, lastName, typeName, stockName, stockCount, stockPrice. The *investorType* UML class diagram is as follows:

investorType
-typeName: string
-stockName: string
-stockCount: int
-stockPrice: double
+investorType()
+investorType(string id, string fName, string lName, string tyName, string stName, int stCount, double stPrice, ostream &output)
+setInvestorInfo(string id, string fName, string lName, string tyName, string stName, int stCount, double stPrice): void
+setType(string tyName): void
+setStockName(string stName): void
+setStockCount(int stCount): void
+setStockPrice(double stPrice): void
+getID() const: string
+getFirstName() const: string
+getLastName() const: string
+getType() const: string
+getStockName() const: string
+getStockPrice() const: double
+getStockCount() const: int
+print(ostream &output) const: void
+writeDataToFile(ofstream &outFile, investorType &tempObj): void

Function Descriptions:

The following are more detailed descriptions of the required functions.

- investorType() is a default constructor to initialize all the member variables.
- investorType(string **id**, string fName, string lName, string tyName, string stName, **int** stCount, **double** stPrice, ostream &output) is a parameterized constructor to initialize the member variables according to the parameters in the constructor body. Expected Error messages for invalid ID, stockCount, stockPrice. Examples are given below:
  - `Invalid ID; No calculations for Dennis; ID 318345`
  - `James's stock count is invalid`
  - `Ryan's stock price is invalid`

- **void** setInvestorInfo(string **id**, string fName, string lName, string tyName, string stName, **int** stCount, **double** stPrice) is to set the member variables according to the parameters.
- **void** setType(string tyName) is to set the typeName member variable
- **void** setStockName(string stName) is to set the stockName member variable
- **void** setStockCount(**int** stCount) is to set the stockCount member variable
- **void** setStockPrice(**double** stPrice) is to set the stockPrice member variable
- string getID() **const** return ID
- string getFirstName() **const** return firstName
- string getLastName() **const** return lastName
- string getType() **const** return typeName
- string getStockName() **const** return stockName
- **double** getStockPrice() **const** return stockPrice
- **double** getStockCount() **const** return stockValue
- **void** print(ostream &output) **const;** Check sample output for more details. You will use few iomanip library for formatting. In this function print stars and dashes using the following instructions:

```
string stars;
stars.append(65, '*');
cout << stars << endl;

string dashes;
dashes.append(50, '_');
```

  - This function prints the following: typeName, firstName, lastName, ID, stockName, stockCount and stockPrice
  - If the parameter is cout, prints on terminal. If the parameter is output, prints to an output file.
- **void** writeDataToFile(ofstream &outFile, investorType &tempObj) This function writes ID, firstName, lastName, typeName, stockName, stockCount and stockPrice to investorsData.txt file. [Hint: call getter functions with tempObj]

stocksPortfolio Class

Implement the Base class_1 named ***stocksPortfolio.*** This class has priceOne, priceTwo, priceThree, stockValue, an object (invTypeObj) of investorType (Composition: has-a relationship) as member variables. The ***stocksPortfolio*** UML class diagram is as follows:

stocksPortfolio
-priceOne: double
-priceTwo: double
-priceThree: double
-stockValue: double
-invTypeObj: investorType
+stocksPortfolio()
+stocksPortfolio(double pOne, double pTwo, double pThree, investorType invTemp)
+setPortfolio(double pOne, double pTwo, double pThree, investorType invTemp): void
+calculateStockValue(investorType invTemp, ostream &output): void
+writePriceChangesToFile(ostream &output) const: void
+getInvestorPrice() const: double
+getPriceOne() const: double
+getPriceTwo() const: double

+getPriceThree() const: double
+getStockValue() const: double
+printPriceChanges(ostream &output) const: void

Function Descriptions:

The following are more detailed descriptions of the required functions.

- stocksPortfolio() is a default constructor to initialize all the member variables
- stocksPortfolio(**double** pOne, **double** pTwo, **double** pThree, investorType invTemp) is a parameterized constructor to initialize the member variables according to the parameters in the constructor body.
- **void** setPortfolio(**double** pOne, **double** pTwo, **double** pThree, investorType invTemp); is to set the member variables according to the parameters.
- `void printPriceChanges(ostream &output) const` prints the three price changes on terminal or to the investorsPortfolio.txt file according to the parameter passed
- **void** calculateStockValue(investorType invTemp, ostream &output)
  - Initial stockValue = Multiply stockPrice with stockCount (Here stockPrice is the initial price of stock; both from investor type; use getters)
  - Multiply stockCount with each new price (3 prices) and print statements accordingly
  - Finally print stockValue: Show Loss/Profit accordingly
  - Profit: If stockValue of priceThree is greater than initial stockPriceShow (priceThree stockValue – initial stockValue; if the result is positive then it is a profit, else loss)
  - Loss: If stockValue of priceThree is less than initial stockValue
  - call printPriceChanges(ostream &output)
  - You use iomanip functions, print stars and dashes in this function
  - Check sample output for more details
  - Special Conditions to consider:
  - If the investor's ID is invalid or stockPrice is negative or stockCount is negative, then show all the stock values with initial price, and other price changes to 0.00
  - You are allowed to declare local variables as needed.
- **void** writePriceChangesToFile(ostream &output) **const** writes the three price changes to investorsData.txt file. (Please check the objects declaration and function(s) calls in stocksMain.cpp to understand the flow)
- **double** getInvestorPrice() **const** return initial stockPrice. [Think: How to get it? This is composition]
- **double** getPriceOne() **const** return priceOne
- **double** getPriceTwo() **const** return priceTwo
- **double** getPriceThree() **const** return priceThree
- **double** getStockValue() **const** return stockValue

  Note: Following my suggestions on how to print stars and dashes are not mandatory. However, it is essential to match your output with the example output. You can move those stars and dashes to any function according to your logic. Codegrade ignores whitespace.

fileIO Class

Implement the Derived class_2 named ***fileIO*** (Inheritance: is-a relationship) which is inherited from multiple classes investorType and stocksPortfolio through public access specifiers. This class has no member variables. The *fileIO* UML class diagram is as follows:

fileIO
+unsigned int readWriteDataFile(ifstream &inFile, ofstream &outData)

Function Descriptions:

The following are more detailed descriptions of the required functions.

- **unsigned int** readWriteDataFile(ifstream &inFile, ofstream &outData); This function does read and write operations on files. Declare local variables as needed. It reads data from investorsData.txt file; declare objects of investorType and stocksPortfolio; call constructors or setters to set the parameters that are read from text file; Call the functions print(parameter) and calculateStockValue(parameter); These functions must write the investors' portfolio (report) to investorsPortfolio.txt file similar to the output that is printed on Terminal. This function must print the total entries of investorsData.txt file that includes invalid data lines. Also, this function returns that total entries.

The *investorType* and *stocksPortfolio* classes must be fully completed prior to *fileIO* class. Refer to the example executions for formatting. Make sure your program includes the appropriate documentation. Your implementation files should be fully commented.

Note: Even though you are not performing the calculations for invalid data, but when you write the investors data to investorsData.txt file, you have to pass the parameters exactly that are passed from stocksMain.cpp

Don't get confused with this:

Initially, you use investorsData.txt as an outputFile, you write every investor's data according to the parameters passed from stocksMain.cpp even though they are invalid. You would be expecting 13 entries that includes valid and invalid data of investors'.

After the declaration of fileIO fobj in stocksMain.cpp, investorsData.txt is considered as input text file. You must generate the investors' portfolio report that is similar to the one printed on terminal. However, you will miss Error :- 318345 is invalid investorID for Dennis (messages for all invalid IDs) in investorsPortfolio.txt output file. The reason is stocksInvestor class has no ostream& variable for setters or print() functions. Please observe that pattern carefully to fully understand the inheritance, composition, fileIO concepts.

Files Provided:

The following files are available to download for this assignment.

- makefile.
- stocksMain.cpp.
- stockInvestor.h
- investorType.h
- stocksPortfolio.h
- fileIO.h
- CS202_Assignment3_Fall23 (Instruction pdf)
- sample_output_on_terminal.txt
- sample_investorsData.txt
- sample_outputFile_investorPortfolios.txt

Submission:

Submit the following files only for this assignment.

- stockInvestorImp.cpp
- investorTypeImp.cpp
- stocksPortfolioImp.cpp
- fileIOImp.cpp

How to Compile?

The provided make file assumes the source file names are as described. If you change the names, the make file will need to be edited. To build the given classes, one main provided and it has good and bad data. To compile, simply type:

- **`make`** (Which will create the *main* for executables respectively.)

How to Run your code?

Use the following commands to run.

- ./stocksMain

Example: How the composition Works?

From ***investorType*** class
investorType(string **id**, string fName, string lName, string tyName, string stName, **int** stCount, **double** stPrice, ostream &output);

Example: investorType fred("T12345", "Fred", "Weasley", "Trader", "AAL", 15, 35.78, cout);

From ***stocksPortfolio*** class
stocksPortfolio(**double** pOne, **double** pTwo, **double** pThree, investorType invTemp);

Example: stocksPortfolio fredSP(37.89, 30.00, 25.66, fred);

From the stocksPortfolio parameters, if you observe fred (object of investorType) is passed in fredSP (object of stocksPortfolio). The other parameters should be initialized with priceOne, priceTwo, priceThree in the constructor body accordingly. In composition, only the object of investorType can access to investorType member functions.

```
investorType fred("T12345", "Fred", "Weasley", "Trader", "AAL", 15, 35.78, cout);
fred.print(cout); //prints the fred data
fred.writeDataToFile(outFile, fred); //You write all the above parameter to
                                     //investorsData.txt exactly even if the given data is
                                     //invalid such as ID, negative stockCount, negative
                                     //stockPrice,

stocksPortfolio fredSP(37.89, 30.00, 25.66, fred);
fredSP.calculateStockValue(fred, cout); // calculate stock Value and print
                                        //(check sample output)
fredSP.writePriceChangesToFile(outFile); //You write the priceChanges to
                                         //investorsData.txt file
```

Example Execution:

(Note: I highlighted some example output in color to emphasize errors and the function calls needed to print that information)

Your original text color for output should be black only.

Below is an example program execution output for this Assignment. Don't make any changes in header files and stocksMain.cpp. Your code/logic is acceptable only in implementation files.

Note: Below example execution format may have few extra white spaces on few pages. To show the output clean, I added a few white spaces in this example output. The codeGrade ignores the whitespace and so no need to be worried on that. Please check the example execution text file that shows expected formatting style.

```
kishore@kishore-virtual-machine:/CS202_Ast3_F23_Ubuntu$ make
g++ -Wall -Wextra -pedantic -std=c++11 -g -c stocksMain.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -c stockInvestorImp.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -c investorTypeImp.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -c stocksPortfolioImp.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -c fileIOImp.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -o stocksMain stocksMain.o
stockInvestorImp.o investorTypeImp.o stocksPortfolioImp.o fileIOImp.o
kishore@kishore-virtual-machine:/CS202_Ast3_F23_Ubuntu$ ./stocksMain

Investor ID: P23456
First Name: Andy                    stockInvestor andy("Andy", "Cartwright", "P23456");
Last Name: Cartwright               andy.print(cout);

*****************************************************************
Investor Type:
Type Name: Trader
Investor ID: T12345
First Name: Fred                    investorType fred("T12345", "Fred", "Weasley", "Trader", "AAL", 15, 35.78, cout);
Last Name: Weasley                  fred.print(cout);

Stock Name: AAL
Stock Count: 15.00
Stock Price: 35.78
_____________________________________________________

Stock Value with investor Price: 536.70
Stock Value with a priceOne change: 568.35
Stock Value with a priceTwo change: 450.00
Stock Value with a priceThree change: 384.90
Final Stock Value: -151.80          stocksPortfolio fredSP(37.89, 30.00, 25.66, fred);
Loss on an investment: 151.80       fredSP.calculateStockValue(fred, cout);
Price Changes:    37.89   30.00   25.66

*****************************************************************
Error :- 318345 is invalid investorID for Dennis    dennis.print(cout);
                                                    This is coming from stockInvestor class
                                                    setInvestorID(string id);

Invalid ID; No calculations for Dennis; ID 318345

          investorType dennis("318345", "Dennis", "Reynolds", "Personal", "IDXX", 200, 152.05, cout);
               Error message is from the investorType parameterized constructor
```

```
**********************************************************************
Investor Type:
Type Name: Personal
Investor ID: 318345
First Name: Dennis
Last Name: Reynolds

Stock Name: IDXX
Stock Count: 200.00
Stock Price: 152.05
_____________________________________________________

Stock Value with investor Price: 0.00
Stock Value with a priceOne change: 0.00
Stock Value with a priceTwo change: 0.00
Stock Value with a priceThree change: 0.00
Final Stock Value: 0.00
Profit on an investment: 0.00
Price Changes:   153.89  150.12  162.33

**********************************************************************
James's stock count is invalid

**********************************************************************
Investor Type:
Type Name: Personal
Investor ID: P29367
First Name: James
Last Name: Johnson

Stock Name: AVGO
Stock Count: -65.00
Stock Price: 163.41
_____________________________________________________

Stock Value with investor Price: 0.00
Stock Value with a priceOne change: 0.00
Stock Value with a priceTwo change: 0.00
Stock Value with a priceThree change: 0.00
Final Stock Value: 0.00
Profit on an investment: 0.00
Price Changes:   153.89  172.98  173.64

**********************************************************************
Ryan's stock price is invalid

**********************************************************************
Investor Type:
Type Name: Trader
Investor ID: T99988
First Name: Ryan
Last Name: Myers

Stock Name: DISH
Stock Count: 120.00
Stock Price: -53.81
_____________________________________________________

Stock Value with investor Price: 0.00
Stock Value with a priceOne change: 0.00
Stock Value with a priceTwo change: 0.00
Stock Value with a priceThree change: 0.00
Final Stock Value: 0.00
Profit on an investment: 0.00
Price Changes:    53.82   50.12   52.76

**********************************************************************

**********************************************************************
Investor Type:
```

James's stock count is invalid and so all the calculations results are 0.00

```
Type Name: Trader
Investor ID: T78564
First Name: Maria
Last Name: Garcia

Stock Name: CSCO
Stock Count: 252.00
Stock Price: 30.96
_____________________________________________________

Stock Value with investor Price: 7801.92
Stock Value with a priceOne change: 5876.64
Stock Value with a priceTwo change: 5178.60
Stock Value with a priceThree change: 7817.04
Final Stock Value: 7817.04
Profit on an investment: 15.12
Price Changes:    23.32   20.55   31.02

*************************************************************
Error :- P7888 is invalid investorID for Oscar

*************************************************************
Investor Type:
Type Name: Personal
Investor ID: P7888
First Name: Oscar
Last Name: Bluth

Stock Name: QCOM
Stock Count: -83.00
Stock Price: 61.95
_____________________________________________________

Stock Value with investor Price: 0.00
Stock Value with a priceOne change: 0.00
Stock Value with a priceTwo change: 0.00
Stock Value with a priceThree change: 0.00
Final Stock Value: 0.00
Profit on an investment: 0.00
Price Changes:    23.32   20.55   31.02

*************************************************************

*************************************************************
Investor Type:
Type Name: Trader
Investor ID: T55669
First Name: Robert
Last Name: Smith

Stock Name: NVDA
Stock Count: 538.00
Stock Price: 56.24
_____________________________________________________

Stock Value with investor Price: 30257.12
Stock Value with a priceOne change: 27497.18
Stock Value with a priceTwo change: 30848.92
Stock Value with a priceThree change: 34152.24
Final Stock Value: 34152.24
Profit on an investment: 3895.12
Price Changes:    51.11   57.34   63.48

*************************************************************
Error :- S31860 is invalid investorID for Selma
Invalid ID; No calculations for Selma; ID S31860

*************************************************************
Investor Type:
Type Name: Personal
```

```
Investor ID: S31860
First Name: Selma
Last Name: Bouvier

Stock Name: NTES
Stock Count: 46.00
Stock Price: 200.00
_____________________________________________________

Stock Value with investor Price: 0.00
Stock Value with a priceOne change: 0.00
Stock Value with a priceTwo change: 0.00
Stock Value with a priceThree change: 0.00
Final Stock Value: 0.00
Profit on an investment: 0.00
Price Changes:   199.25  180.67  203.17

*****************************************************************

*****************************************************************
Investor Type:
Type Name: Trader
Investor ID: T13502
First Name: Luke
Last Name: Skywalker

Stock Name: FOX
Stock Count: 512.00
Stock Price: 27.50
_____________________________________________________

Stock Value with investor Price: 14080.00
Stock Value with a priceOne change: 14080.00
Stock Value with a priceTwo change: 14085.12
Stock Value with a priceThree change: 7767.04
Final Stock Value: -6312.96
Loss on an investment: 6312.96
Price Changes:    27.50   27.51   15.17

*****************************************************************

*****************************************************************
Investor Type:
Type Name: Personal
Investor ID: P29367
First Name: James
Last Name: Johnson

Stock Name: AVGO
Stock Count: 65.00
Stock Price: 163.41
_____________________________________________________

Stock Value with investor Price: 10621.65
Stock Value with a priceOne change: 10002.85
Stock Value with a priceTwo change: 11243.70
Stock Value with a priceThree change: 11286.60
Final Stock Value: 11286.60
Profit on an investment: 664.95
Price Changes:   153.89  172.98  173.64

*****************************************************************

*****************************************************************
Investor Type:
Type Name: Trader
Investor ID: T99988
First Name: Ryan
Last Name: Myers
```

```
Stock Name: DISH
Stock Count: 120.00
Stock Price: 53.81

_____________________________________________

Stock Value with investor Price: 6457.20
Stock Value with a priceOne change: 6458.40
Stock Value with a priceTwo change: 6014.40
Stock Value with a priceThree change: 6331.20
Final Stock Value: -126.00
Loss on an investment: 126.00
Price Changes:    53.82   50.12   52.76

*****************************************************************

*****************************************************************
Investor Type:
Type Name: Personal
Investor ID: P78887
First Name: Oscar
Last Name: Bluth

Stock Name: QCOM
Stock Count: 83.00
Stock Price: 61.95

_____________________________________________

Stock Value with investor Price: 5141.85
Stock Value with a priceOne change: 1935.56
Stock Value with a priceTwo change: 1705.65
Stock Value with a priceThree change: 2574.66
Final Stock Value: -2567.19
Loss on an investment: 2567.19
Price Changes:    23.32   20.55   31.02

*****************************************************************

*****************************************************************
Investor Type:
Type Name: Personal
Investor ID: P31860
First Name: Selma
Last Name: Bouvier

Stock Name: NTES
Stock Count: 46.00
Stock Price: 200.00

_____________________________________________

Stock Value with investor Price: 9200.00
Stock Value with a priceOne change: 9165.50
Stock Value with a priceTwo change: 8310.82
Stock Value with a priceThree change: 9345.82
Final Stock Value: 9345.82
Profit on an investment: 145.82
Price Changes:   199.25  180.67  203.17

*****************************************************************
fileIO object declarations begin

*****************************************************************
Error :- 318345 is invalid investorID for Dennis
Error :- P7888 is invalid investorID for Oscar
Error :- S31860 is invalid investorID for Selma
Total Entries: 13
kishore@kishore-virtual-machine:
```

These messages are printed on terminal after reading Invalid IDs from investorsData.txt file. These are the only messages that are not printed in output file. All invalid ID's that has the following print message: `Error :- 318345 is invalid investorID for Dennis` are missed in outputFile.

Sample_investorsData.txt_after reaching the end of stocksMain.cpp

	ID	FirstName	LastName	invType	stockName	stockCount	stockPrice	pCh1	pCh2	pCh3
2	T12345	Fred	Weasley	Trader	AAL	15.00	35.78	37.89	30.00	25.66
3	318345	Dennis	Reynolds	Personal	IDXX	200.00	152.05	153.89	150.12	162.33
4	P29367	James	Johnson	Personal	AVGO	-65.00	163.41	153.89	172.98	173.64
5	T99988	Ryan	Myers	Trader	DISH	120.00	-53.81	53.82	50.12	52.76
6	T78564	Maria	Garcia	Trader	CSCO	252.00	30.96	23.32	20.55	31.02
7	P7888	Oscar	Bluth	Personal	QCOM	-83.00	61.95	23.32	20.55	31.02
8	T55669	Robert	Smith	Trader	NVDA	538.00	56.24	51.11	57.34	63.48
9	S31860	Selma	Bouvier	Personal	NTES	46.00	200.00	199.25	180.67	203.17
10	T13502	Luke	Skywalker	Trader	FOX	512.00	27.50	27.50	27.51	15.17
11	P29367	James	Johnson	Personal	AVGO	65.00	163.41	153.89	172.98	173.64
12	T99988	Ryan	Myers	Trader	DISH	120.00	53.81	53.82	50.12	52.76
13	P78887	Oscar	Bluth	Personal	QCOM	83.00	61.95	23.32	20.55	31.02
14	P31860	Selma	Bouvier	Personal	NTES	46.00	200.00	199.25	180.67	203.17

Note: Even invalid IDs, negative stockPrice, negative stockCount are saved exactly into the investorsData.txt file. Further, those invalids are set after investor "Selma" in stocksMain.cpp and so you can see profit/loss of those investors. Even invalid investors is considered as an entry and so you see 13 entries in total.