1 Functions

A program that sets up and manipulates `DividendAccount` objects is in the directory `~cs203/Projects/Project4`. Copy the four files in this directory and make changes to the file `account.cc`. The main program in this file makes calls to two functions that you are to implement.

1.1 `getDouble()`

`getDouble()` takes a `string` argument, which is displayed as a prompt to the user. The function then reads a `double` from `cin` and returns it. `getDouble()` has the following prototype.

```
double getDouble(const string & inPrompt);
```

1.2 `getAccountInfo()`

`getAccountInfo()` has the following prototype.

```
void getAccountInfo(
    const string & inPrompt,
    string &       outName,
    double &       outBalance,
    double &       outDividendRate);
```

The function prints the first argument as a prompt for the three inputs, then reads in a `string` value, and two `double` values from `cin`, which are assigned to the second, third, and fourth arguments, respectively. Note that the second, third, and fourth arguments are reference parameters.

2 Methods

In this part of the assignment, you will add two methods to the `DividendAccount` class. One is a method `display()` that will display the name and balance of an account, and the other is `applyDividend()` which calculates the dividend and deposits it in the account.

2.1 `display()`

Add a method `display()` to the `DividendAccount` class that displays the name and balance for the account in the following form:

```
Account Name:    <name>
Account Balance: $<balance>
```

Note that `<name>` and `<balance>` refer to the data values stored in the `DividendAccount` object. For example, if an account named Walt has a balance of $3.19, the output should read

```
Account Name:    Walt
Account Balance: $ 3.19
```

The prototype of your `display()` method is

```
void display() const;
```

Here, the `const` means that this method may not change the state of the object.

Remember that adding a new method requires changes in two places:

- An entry declaring the method must be added to public part of the interface in `DividendAccount.h`.
- An implementation of the method must be added to the implementation in the file `DividendAccount.cc`.

2.2 `applyDividend()`

Add a `applyDividend()` method to the `DividendAccount` class. This method calculates the dividend and deposits it back into the `DividendAccount` object. Note that the `DividendAccount` class has a method for computing the dividend. Use this method when computing the dividend.

The prototype for this method is:

```
void applyDividend();
```

3 Notes

Note that we have provided you with a `Makefile` for this project. Use the `Makefile` to compile your program. To compile the program, type `make` in the Terminal window.

After you have completed the assignment, be sure to "`make clean`". You can do this by typing `make clean`. This will remove your `.o` and `.exe` files which you no longer need.

4 What to Hand In

Remember to include `Pre` and `Post` conditions.

Turn in a copy of your revised `account.cc`, `DividendAccount.h` and `DividendAccount.cc` files, and put your output in a file called `output.txt`. Print each of these files using `a2ps`.

Here is a sample run. Use this example and another of your choosing in your sample output.

```
Enter account name, initial balance,
    and dividend rate: Home 1000 10
Enter account name, initial balance,
    and dividend rate: Vacation 2000 6
Enter amount to withdraw from Home: 300
Enter amount to deposit in Vacation: 100
Account Name:    Home
Account Balance: $705.83
Account Name:    Vacation
Account Balance: $2110.50
```