Part C - Encapsulation

Member Operators / Helper Functions

Workshop 6 V1.0

In this workshop, you are to overload operators to work with a class called Account.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities to

- overload operators as a member function
- overload an operator as a helper function
- overload an operator as a friend function
- Eliminate the need for a friend function by adding proper accessors to the class
- reflect on what you have learned in this workshop

SUBMISSION POLICY

The “in-lab” section is to be completed **during your assigned lab section**. It is to be completed and submitted by the end of the workshop. If you do not attend the workshop, you can submit the “in-lab” section along with your “at-home” section (a 20% late deduction will be assessed). The “at-home” portion of the lab is **due the day before your next scheduled workshop**

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to regularly backup your work.

ACCOUNT CLASS

Download or clone workshop 6 from https://github.com/Seneca-244200/OOP-Workshop6

Open Workshop6/in_lab directory and view the code in **Account.h** and **Account.cpp.**

Account is designed and coded to hold and display information about an Account in a bank. These information are: name and balance.

Account has three constructors and a display function:

The **default (no argument) constructor** sets the Account to a safe empty state.

The **one argument constructor** that sets the name to and empty string and balance to the incoming double value.

The **two argument constructor** set the name and balance to incoming values.

The **display** function, displays the contents of the Account and goes to newline if the "**gotoNewline**" argument is true;

Your task is to complete the code of the **Account** class or add helper functions to be able to work with following operators;

"+", "+=", "=" AND "<<"

The overload of the above operators should make the following code possible:

Member Operators:

Overload the **operartor+=** so the following is possible:
If A and B and C are Account objects:
A = **B += C**: adds the balance of C to B and returns the reference of B, so A will be to B afterwards.

Overload the **operator=** so the following is possible:
A = **B = "new name"**: Sets the name of B to "new name" and returns the reference of B.

Helper Operator Overloads:

A friend **operator+**:
Overload **operator+** as a friend so the following is possible:
A = B + C: this operator returns an account with an empty name and a balance of the sum of two balances of B and C.

A helper **operator<<:**
Overload **operator<<** so the following is possible:

cout << A << endl;

operator<< should call the display member function of A (no newline printed) and return the reference of **ostream**.

```
// OOP244 Workshop 6: operators
// File: w6_in_lab.cpp
// Version: 1.0
// Date: 2016/02/22
// Author: Fardad Soleimanloo
// Description:
// This file tests in-lab section of your workshop
/////////////////////////////////////////////

#include <iostream>
#include "Account.h"
using namespace sict;
using namespace std;
void displayABC(const Account& A,
                const Account& B,
                const Account& C){
  cout << "A: " << A << endl << "B: " << B << endl
    << "C: " << C << endl << "--------" << endl;
}
int main(){
  Account A;
  Account B("Saving", 10000.99);
  Account C("Checking", 100.99);
  displayABC(A, B, C);
  A = B + C;
  displayABC(A, B, C);
  A = "Joint";
  displayABC(A, B, C);
  A = B += C;
  displayABC(A, B, C);
  A = B += C += 100.01;
  displayABC(A, B, C);
  return 0;
}
```

Output Example:
(Your output should exactly match the following)

```
A: No Name: $0.00
B: Saving: $10000.99
C: Checking: $100.99
```

```
--------
A: No Name: $10101.98
B: Saving: $10000.99
C: Checking: $100.99
--------
A: Joint: $10101.98
B: Saving: $10000.99
C: Checking: $100.99
--------
A: Saving: $10101.98
B: Saving: $10101.98
C: Checking: $100.99
--------
A: Saving: $10302.98
B: Saving: $10302.98
C: Checking: $201.00
--------
```

IN-LAB SUBMISSION (70%)

To submit the in-lab section demonstrate execution of your program whit the exact output as example above.

If not on matrix already, upload **Account.h, Account.cpp** and **w6_in_lab.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit w6_in_lab <ENTER>
```

and follow the instructions.

AT-HOME SECTION: REMOVING FRIEND USING ACCESSOR METHODS(20%)

Eliminate the need for the “friend” access-right in the class by adding a proper accessor method (query) to the class and modifying the operator+ to use the accessor instead of using the class attributes directly.

Overload **operator+=** as a helper so the following is possible:
If “d” and “e” are double variables and A is an Account object.

d = **e** **+=** **A**;
The Balance or A should be added to the value of “e” and then the modified value returned

CLIENT MODULE

Here is a sample of implementation file for the `w6_at_home.cpp` main module that you should use to test your implementation:

```cpp
// OOP244 Workshop 6: operators
// File: w6_at_home.cpp
// Version: 1.0
// Date: 2016/02/22
// Author: Fardad Soleimanloo
// Description:
// This file tests At-Home section of your workshop
/////////////////////////////////////////////

#include <iostream>
#include "Account.h"
using namespace sict;
using namespace std;
void displayABC(const Account& A,
  const Account& B,
  const Account& C){
  cout << "A: " << A << endl << "B: " << B << endl
    << "C: " << C << endl << "--------" << endl;
}
int main(){
  Account A;
  Account B("Saving", 10000.99);
  Account C("Checking", 100.99);
  Account* AC[3] = { &A, &B, &C };
  double balance = 0;
  displayABC(A, B, C);
  A = B + C;
  displayABC(A, B, C);
  A = "Joint";
  displayABC(A, B, C);
  A = B += C;
  displayABC(A, B, C);
  A = B += C += 100.01;
  displayABC(A, B, C);
  for (int i = 0; i < 3; i++){
    cout << i+1 << "- " << (balance += *AC[i]) << endl;
  }
  cout << "Total Balance: " << balance << endl;
  return 0;

}
```

Output Example:
`(Your output should exactly match the following)`

```
A: No Name: $0.00
B: Saving: $10000.99
C: Checking: $100.99
--------
A: No Name: $10101.98
B: Saving: $10000.99
C: Checking: $100.99
--------
A: Joint: $10101.98
B: Saving: $10000.99
C: Checking: $100.99
--------
A: Saving: $10101.98
B: Saving: $10101.98
C: Checking: $100.99
--------
A: Saving: $10302.98
B: Saving: $10302.98
C: Checking: $201.00
--------
1- 10302.98
2- 20605.96
3- 20806.96
Total Balance: 20806.96
```

REFLECTION (10%)

Please provide brief answers to the following questions in a text file named **`reflect.txt.`**

1. Explain why if possible, we should avoid using friend helper functions.
2. Could the first helper operator+ (which accepts Accounts as left and right operands) implemented as a member operator? If yes, how?

3. In this line of your main function:
   ```
   A = B += C += 100.01;
   ```
   the under lined operator+= that accepts a double as right operand is never defined. Explain how is it, that the code compiles and runs correctly.

SUBMISSION

To test and demonstrate execution of your program use the same data as the sample output above.

If not on matrix already, upload **Account.h, Account.cpp** and **w6_at_home.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit w6_at_home <ENTER>
```

and follow the instructions.