Practice 9
Operator Overloading

2018 second semester
Computer Engineering Programming
CONTENTS

Explanation about assignment

Basic Operator Overloading

Friend Function
Operator Overloading Perspective

• Built-in operators
  – e.g., +, -, =, %, ==, /, *
  – Already work for C++ built-in types
  – VERY similar to overloading functions

• Always overload with similar "actions"!

• Example Declaration:
  const Money operator +(  
  const Money& amount1, 
  const Money& amount2);
Polymorphism about operator “+”

- Polymorphism is a feature of Object Oriented Programming that allows the object to behave differently in different conditions.

- Example about Operator “+”
  - Addition
    ```
    int a = 2, b = 6, c;
    c = a + b;
    ```
  - Concatenation
    ```
    string a = "Hello", c;
    c = a + "world";
    ```
  - Merge
    ```
    SortedArray a(2, 5, 9), b(3, 7, 10), c;
    c = a + b; // c={2,3,5,7,9,10}
    ```
Overloaded "+"

```cpp
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;

class Money {
public:
    Money(double amount);
    Money(int theDollars, int theCents);
    Money(int theDollars);
    double getAmount() const;
    int getDollars() const;
    int getCents() const;
private:
    int dollars;
    int cents;
};

const Money operator +(const Money& amount1, const Money& amount2){...}

int main() {
    double SumOfMoney;
    Money yourAmount(5,3), myAmount(10, 9);
    Money ourAmount = yourAmount + myAmount;
    SumOfMoney = (double)ourAmount.getCents()/100 + ourAmount.getDollars();
    cout << "Sum of our money is $" << SumOfMoney << endl;
    return 0;
}
```
Overloaded "+"

```cpp
Money::Money(double amount)
    : dollars(int(amount))
    , cents((amount-int(amount))*100)
{
}

Money::Money(int theDollars, int theCents)
    : dollars(theDollars)
    , cents(theCents)
{
}

Money::Money(int theDollars)
    : dollars(theDollars)
    , cents(0)
{
}

double Money::getAmount() const
{
    return dollars*100+cents;
}

int Money::getDollars() const
{
    return dollars;
}

int Money::getCents() const
{
    return cents;
}
```
Overloaded "=="

```cpp
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;

class Money {
public:
    Money(double amount);
    Money(int theDollars, int theCents);
    Money(int theDollars);
    double getAmount() const;
    int getDollars() const;
    int getCents() const;
private:
    int dollars;
    int cents;
};

const Money operator +(const Money& amount1, const Money& amount2) { ... }
bool operator == (const Money& amount1, const Money& amount2) { ... }

int main() {
    Money yourAmount(10, 40), myAmount(10, 40);
    if (yourAmount == myAmount)
        cout << "We have the same amounts." << endl;
    else
        cout << "One of us is richer." << endl;
}
```
Exercise

• Make operator "/" print like as below.

```cpp
#include <iostream>
#include <cmath>
using namespace std;

class Money {
public:
    Money(double amount);
    Money(int theDollars, int theCents);
    Money(int theDollars);
    double getAmount() const;
    int getDollars() const;
    int getCents() const;
private:
    int dollars;
    int cents;
};

const Money operator +(const Money& amount1, const Money& amount2) {
    ...
}

bool operator ==(const Money& amount1, const Money& amount2) {
    ...
}

double operator /(const Money& amount1, const Money& amount2) {
    ...
}

int main() {
    double BetterThan;
    Money yourAmount(10, 40), myAmount(1, 30);
    cout << "How much more is your money than me? \& \>
```
Friend Functions

• Not a member function
• But, can **direct access to private members**
  – Efficient for operator overloading
    (avoid calling accessor(getValue()) & mutator(setValue()))
• Use keyword `friend` in front of function declaration
• Notice!
  – **Friend functions are not member functions!**

Ex)

class A {
    friend f_name (...)  
    ...
}
```cpp
#include <iostream>

using namespace std;

class privater {
public:
    privater() {
        money = 100;
    }

private:
    int money;
    friend void privateFunction(privater &pri);
};

void privateFunction(privater &pri) {
    cout << "I have " << pri.money << " dollars." << endl;
}

int main() {
    privater p;
    privateFunction(p);
    return 1;
}
```

Friend Functions Example 1
Friend Classes

• Entire classes can be friends
  – Similar to friend functions.

• Example:
  
  ```
  class A {
    friend class B;
    ...
  }
  ```
  – All class B member functions are friends of A
  – B’s member functions can access A’s private members.
Friend Classes Example

```cpp
#include <iostream>
using namespace std;

class B;

class A {
    friend class B;
    public:
    A() {
        a_data = 10;
    }
    private:
    int a_data;
};

class B {
    public:
    B() {
        b_data = 20;
    }
    private:
    int b_data;
};

void showAData(A a) {
    std::cout << "A data : " << a.a_data << endl;
}

void showBData(B b) {
    std::cout << "B data : " << b.b_data << endl;
}

int main() {
    A a;
    B b;
    b.showAData(a);
    return 1;
}
```

B class exists.

```cpp
#include <iostream>
using namespace std;

class B;

class A {
    friend class B;
    public:
    A() {
        a_data = 10;
    }
    private:
    int a_data;
};

class B {
    public:
    B() {
        b_data = 20;
    }
    private:
    int b_data;
};

void showAData(A a) {
    std::cout << "A data : " << a.a_data << endl;
}

void showBData(B b) {
    std::cout << "B data : " << b.b_data << endl;
}

int main() {
    A a;
    B b;
    b.showAData(a);
    a.showData(b);
    return 1;
}
```

Error!

Why?