1. What is the difference between a (binary) operator and a function?

There are two differences between the use of an operator and an ordinary function call. The syntax is different; an operator is often “called” by placing it between or sometimes after the arguments. The second difference is that the compiler determines which “function” to call. For example, operator + with floating point args, the compiler “calls” the functions to perform floating point addition. If you use operator + with a floating point and an integer, the compiler “calls” a special function to cast the int into a float, then “calls” the floating point addition code.

[Operator Overloading. URL: <http://www.drbio.cornell.edu/pl47/programming/TICPP-2nd-ed-Vol-one-html/Chapter12.html>, on 19/11/2016, at 21:25]

1. Is it possible using operator overloading to change the effect of + on integers? Why or why not?

Yes, it is already mentioned in the above question. Because when use operator + with a floating point and an integer, the compiler “calls” a special function to cast the int into a float, then “calls” the floating point addition code.

[Operator Overloading. URL: <http://www.drbio.cornell.edu/pl47/programming/TICPP-2nd-ed-Vol-one-html/Chapter12.html>, on 19/11/2016, at 21:25]

1. Why can’t we overload << or >> as member operators?

Because the first (left hand side) is NOT an object of a class, we must have a stand-alone function.

[ADS1, Moodle page, 2009-lectures-wk9-Overloaded\_Operators.pdf, page 10]

1. Below is the definition for a class called Percent. Objects of type Percent represent percentages such as 10% or 99%. Give the definitions of the overloaded operators >> and << so that they can be used for input and output with objects of the class Percent. Assume that input always consists of an integer followed by the character ’%’, such as 25%. All percentages are whole numbers and are stored in an int member variable named value. You do not yet need to define the other overloaded operators and do not yet need to define the constructor. Firstly, you only have to define the overloaded operators >> and <<.

#include <iostream>

using namespace std;

class Percent

{

public:

friend bool operator== (const Percent& first, const Percent& second);

friend bool operator< (const Percent& first, const Percent& second);

Percent( );

Percent(int valueIn);

friend istream& operator>> (istream& inputStream, Percent& aPercent);

friend ostream& operator<< (ostream& outputStream, const Percent& aPercent);

//There will be other members and friends.

private:

int value;

};

1. Add to the class overloaded operators to add, subtract and multiply 2 percentages. If you have to re-write the code so that wherever possible the overloaded operators are made member functions of the class.

//There will be other members and friends.

/\* usage: used/modified

\* availability: Absolute\_C++\_(5th\_Edition).pdf

\* chapter 8.1: Basic Operator Overloading

\* page: 324-325

\*/

const Percent operator+ (const Percent& right) const;

const Percent operator- (const Percent& right) const;

const Percent operator\* (const Percent& right) const;

1. Implement all of the overloaded operators in the class (and any stand-alone ones still declared as friend functions of the class)

***N.B.*** *Think carefully about the multiplication: 50% \* 50% should mean 50% OF 50%, and therefore provide the answer 25% not 2500% (hint: 50 \* 50 / 100 = 25)*

Percent::Percent()

: value(0)

{}

Percent::Percent(int valueIn)

: value(valueIn)

{}

/\* usage: used/modified

\* availability: Absolute\_C++\_(5th\_Edition).pdf

\* chapter 8.1: Basic Operator Overloading

\* page: 326

\*/

bool operator== (const Percent& first, const Percent& second)

{

return (first.value == second.value);

}

/\* usage: used/modified

\* availability:

\* <http://www.learncpp.com/cpp-tutorial/96-overloading-the-comparison-operators/>

\* on 16/11/2016, at 19:45

\*/

bool operator< (const Percent& first, const Percent& second)

{

return (first.value < second.value);

}

istream& operator>> (istream& inputStream, Percent& aPercent)

{

cout << "\nEnter value: ";

inputStream >> aPercent.value;

return inputStream;

}

/\* usage: used/modified

\* availability: Absolute\_C++\_(5th\_Edition).pdf

\* chapter 8.1: Basic Operator Overloading

\* page: 327, 348

\*/

ostream& operator<< (ostream& outputStream, const Percent& aPercent)

{

outputStream << aPercent.value << "%" << endl;

return outputStream;

}

const Percent Percent::operator+ (const Percent& right) const

{

return Percent(value + right.value);

}

const Percent Percent::operator- (const Percent& right) const

{

return Percent(value - right.value);

}

const Percent Percent::operator\* (const Percent& right) const

{

return Percent(value \* right.value / 100);

}

1. Write a program which fully tests your class.

#include "Percent.h"

#include <iostream>

using namespace std;

int main()

{

Percent p1(10), p2(20);

operator<< (cout, p1);

operator<< (cout, p2);

Percent p3;

p3 = p1.operator+(p2); // 10 + 20 = 30

operator<<(cout, p3);

Percent p4;

p4 = p1.operator-(p2); // 10 - 20 = -10

operator<< (cout, p4);

Percent p5;

P5 = 50;

p5 = p5.operator\*(p5); // 50 \* 50 / 100 = 25

operator<< (cout, p5);

Percent p6;

operator >> (cin, p6); // whatever user input is the value

operator<< (cout, p6);

system("pause");

return 0;

}

1. Write a Money class, with data members euro and cent. Add overloaded operators (as member functions where this is possible) to do the following:

* Subtract 2 Money amounts.
* Multiply a Money amount by an integer to facilitate code like this:

myMoney = yourMoney \* 2;

* Divide a Money amount by a decimal
* Output a Money amount using the insertion operator
* Compare 2 Money amounts (with <, > and ==)

/\*

• Subtract 2 Money amounts

==> left hand side cannot be mofified, therefore must be friend

• Multiply a Money amount by an integer to facilitate code like this:

==> myMoney = yourMoney \* 2;

==> left hand side need to be modifed, therefore must be friend

• Divide a Money amount by a decimal

==> left hand side cannot be mofified, therefore must be friend

• Output a Money amount using the insertion operator

==> pass in more than one arg, therefore must be friend

• Compare 2 Money amounts (with <, > and ==)

\*/

#pragma once

#include <ostream>

using namespace std;

// Money.h

class Money

{

public:

friend ostream& operator<< (ostream& outputStream, const Money& money);

//friend const Money operator+ (Money& left, const Money& right); // left can be modified

friend const Money operator+ (const Money& left, const Money& right);

friend const Money operator- (const Money& left, const Money& right);

friend const Money operator\* (const Money& money, int multiplyBy);

friend const Money operator/ (const Money& money, double dividedBy);

// without friend, other friend function cannot get access

friend int getTotalCent(const Money& amount); // to handle when pass in -5.70 for example

Money();

Money(int euro);

Money(int euro, int cent);

Money(double amount);

bool operator< (const Money& right);

bool operator> (const Money& right);

bool operator== (const Money& right);

private:

int euro;

int cent;

};

#include "Money.h"

#include <math.h>

// Money.cpp

Money::Money()

: euro(0), cent(0)

{}

Money::Money(int euro)

: euro(euro), cent(0)

{}

Money::Money(int euro, int cent)

: euro(euro), cent(cent)

{}

/\* euro = static\_cast<int> (amount);

\* cent = static\_cast<int> (amount \* 100) % 100; ==> 4.60 will print 4.59

\*

\* double doubleCents = 4.60 \* 100;

\* int intCents = static\_cast<int> (round(abs(doubleCents))); // tested will give 460

\* cout << intCents << endl;

\*/

Money::Money(double amount)

: euro(static\_cast<int>(amount)), cent(static\_cast<int>(round(abs(amount \* 100))) % 100)

{}

bool Money::operator< (const Money& right)

{

return (euro < right.euro) || (euro == right.euro && cent < right.cent);

}

bool Money::operator> (const Money& right)

{

return (euro > right.euro) || (euro == right.euro && cent > right.cent);

}

bool Money::operator== (const Money& right)

{

return (euro == right.euro && cent == right.cent);

}

ostream& operator<< (ostream& outputStream, const Money& money)

{

double amount = getTotalCent(money) / 100.0;

outputStream << amount << endl;

return outputStream;

}

const Money operator+ (const Money& left, const Money& right)

{

int totalCent1 = getTotalCent(left);

int totalCent2 = getTotalCent(right);

double amount = (totalCent1 + totalCent2) / 100.0;

return Money(amount);

}

const Money operator- (const Money& left, const Money& right)

{

int totalCent1 = getTotalCent(left);

int totalCent2 = getTotalCent(right);

double amount = (totalCent1 - totalCent2) / 100.0;

return Money(amount);

}

// Multiply a Money amount by an integer to facilitate code like this:

// myMoney = yourMoney \* 2;

const Money operator\* (const Money& amount, int multiplyBy)

{

double myAmount = getTotalCent(amount) \* multiplyBy / 100.0;

return Money(myAmount);

}

// Divide a Money amount by a decimal

const Money operator/ (const Money& amount, double dividedBy)

{

double myAmount = getTotalCent(amount) / dividedBy / 100.0;

return Money(myAmount);

}

int getTotalCent(const Money& amount)

{

if (amount.euro < 0)

{

return (amount.euro \* 100 - amount.cent);

}

else

{

return (amount.euro \* 100 + amount.cent);

}

}

#include "Money.h"

#include <iostream>

using namespace std;

// main

int main()

{

Money m1; // default constructor is call, euro and cent initialized to Zero

Money m2(2); // pass in euro only

Money m3(3, 30); // pass in euro and cent

Money m4(4.40); // pass in amount

operator<< (cout, m1);

operator<< (cout, m2);

operator<< (cout, m3);

operator<< (cout, m4);

cout << endl; // print blank line

Money tony(5.50), paul(5.50);

if (tony.operator==(paul))

{

cout << "Tony and Paul both have the same amount of money!" << endl;

}

Money mary(2.2), sara(2.3);

if (mary.operator<(sara))

{

cout << "Mary has less money than Sara!" << endl;

}

if (sara.operator>(mary))

{

cout << "Sara has more money than Rose!" << endl;

}

if (!sara.operator==(mary))

{

cout << "Sara and Mary do not have the same amount of money!" << endl;

}

cout << endl; // print blank line

Money m5, m6(4.60), m7(-5.70);

m5 = operator+ (m6, m7); // 4.60 - 5.70 = -1.10

operator<< (cout, m5);

m5 = operator- (m6, m7); // 4.60 - (-5.70) = 10.30

operator<< (cout, m5);

m5 = operator\*(m6, 3); // 4.60 \* 3 = 13.80

operator<< (cout, m5);

m5 = operator/(m7, 3.3); // -5.70 / 3.3 = -1.727272... ==> -1.73

operator<< (cout, m5);

system("pause");

return 0;

}