```
//
//  main.cpp
//  AbsoluteCpp_ch8_1
//
//  Created by Liwei on 2020/3/20.
//  Copyright © 2020 Liwei. All rights reserved.
//

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

//Class for amounts of money in U.S. currency
class Money
{

public:
    Money( );
    Money(double amount);
    Money(int theDollars, int theCents);
    Money(int theDollars);
    double getAmount( ) const;
    int getDollars( ) const;
    int getCents( ) const;
    void input( ); //Reads the dollar sign as well as the amount number.
    void output( ) const;

private:
    int dollars; //A negative amount is represented as negative
    //dollars and
    int cents; //negative cents. Negative $4.50 is represented
    //as -4 and -50.
    int dollarsPart(double amount) const;
    int centsPart(double amount) const;
    int round(double number) const;
};

const Money operator +(const Money& amount1, const Money& amount2);
const Money operator -(const Money& amount1, const Money& amount2);
bool operator ==(const Money& amount1, const Money& amount2);

const Money operator -(const Money& amount);

int main(int argc, const char * argv[]) {
    Money yourAmount, myAmount(10, 9); cout << "Enter an amount of money:
     "; yourAmount.input( );
    cout << "Your amount is "; yourAmount.output( );
    cout << endl;
    cout << "My amount is "; myAmount.output( );
    cout << endl;

    if (yourAmount == myAmount)
        cout << "We have the same amounts.\n";
    else
```

```
        cout << "One of us is richer.\n";

    Money ourAmount = yourAmount + myAmount;
    yourAmount.output( ); cout << " + "; myAmount.output( );
    cout << " equals "; ourAmount.output(); cout << endl;

    Money diffAmount = yourAmount - myAmount;
    yourAmount.output( ); cout << " - "; myAmount.output( );
    cout << " equals "; diffAmount.output(); cout << endl;

    return 0;
}

const Money operator +(const Money& amount1, const Money& amount2)
{
    int allCents1 = amount1.getCents( ) + amount1.getDollars( )*100; int
     allCents2 = amount2.getCents( ) + amount2.getDollars( )*100; int
     sumAllCents = allCents1 + allCents2;
    int absAllCents = abs(sumAllCents); //Money can be negative.
    int finalDollars = absAllCents / 100;
    int finalCents = absAllCents % 100;
    if (sumAllCents < 0)
    {
        finalDollars = -finalDollars;
        finalCents = -finalCents; }
    return Money(finalDollars, finalCents);
}

const Money operator -(const Money& amount1, const Money& amount2)
{
    int allCents1 = amount1.getCents()+amount1.getDollars()*100;
    int allCents2 = amount2.getCents()+amount2.getDollars()*100;
    int diffAllCents = allCents1 - allCents2;
    int absAllCents = abs(diffAllCents);
    int finalDollars = absAllCents / 100;
    int finalCents = absAllCents % 100;
    if (diffAllCents < 0)
    {
        finalDollars = -finalDollars;
        finalCents = -finalCents; }
    return Money(finalDollars, finalCents);
}

bool operator ==(const Money& amount1, const Money& amount2)
{
    return ((amount1.getDollars( ) == amount2.getDollars( )) &&
     (amount1.getCents( ) == amount2.getCents( )));
}

const Money operator -(const Money& amount) {
    return Money(-amount.getDollars( ), -amount.getCents( ));

}

Money::Money( ): dollars(0), cents(0)
```

```
{/*Body intentionally empty.*/}

Money::Money(double amount)
: dollars(dollarsPart(amount)), cents(centsPart(amount))
{/*Body intentionally empty*/}

Money::Money(int theDollars)
:  dollars(theDollars), cents(0)
{/*Body intentionally empty*/}

//Uses cstdlib:
Money::Money(int theDollars, int theCents)  {
    if ((theDollars < 0 && theCents > 0) ||
        (theDollars > 0 && theCents < 0))
    {
        cout << "Inconsistent money data.\n"; exit(1);
    }
    dollars = theDollars;
    cents = theCents;
}

double Money::getAmount( ) const {
    return (dollars + cents*0.01);  }

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

int Money::getCents( ) const
{
    return cents;
}

//Uses iostream and cstdlib:
void Money::output( ) const
{
    int absDollars = abs(dollars);
    int absCents = abs(cents);
    if (dollars < 0 || cents < 0)
        //accounts for dollars == 0 or cents == 0
        cout << "$-";
    else
        cout << '$';
    cout << absDollars;
    if (absCents >= 10)
        cout << '.' << absCents;
    else
        cout << '.' << '0' << absCents;
}


//Uses iostream and cstdlib:
void Money::input( )
{
    char dollarSign;
    cin >> dollarSign; //hopefully
    if (dollarSign != '$')
    {
        cout << "No dollar sign in Money input.\n";
        exit(1);
    }
    double amountAsDouble;
    cin >> amountAsDouble;
    dollars = dollarsPart(amountAsDouble);
    cents = centsPart(amountAsDouble);
}

int Money::dollarsPart(double amount) const {
    return static_cast<int>(amount);
}

int Money::centsPart(double amount) const{
    double doubleCents = amount * 100;
    int intCents = (round(fabs(doubleCents))) % 100;
    //% can misbehave on negatives
    if (amount < 0)
        intCents = -intCents;
    return intCents;
}

int Money::round(double number) const{
    return static_cast<int>(floor(number + 0.5));
}
```

```cpp
//
//  main.cpp
//  AbsoluteCpp_ch8_2
//
//  Created by Liwei on 2020/3/20.
//  Copyright © 2020 Liwei. All rights reserved.
//

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

//Class for amounts of money in U.S. currency
class Money
{
public:
    Money( );
    Money(double amount);
    Money(int dollars, int cents);
    Money(int dollars);
    double getAmount( ) const;
    int getDollars( ) const;
    int getCents( ) const;
    void input( ); //Reads the dollar sign as well as the amount number.
    void output( ) const;
    const Money operator +(const Money& amount2) const;
    const Money operator -(const Money& amount2) const;
    bool operator ==(const Money& amount2) const;
    const Money operator -( ) const;

private:
    int dollars; //A negative amount is represented as negative //dollars
     and
    int cents; //negative cents. Negative $4.50 is represented as //-4 and
     -50.
    int dollarsPart(double amount) const;
    int centsPart(double amount) const;
    int round(double number) const;
};

int main(int argc, const char * argv[]) {
    // insert code here...
    std::cout << "Hello, World!\n";
    return 0;
}


const Money Money::operator +(const Money& secondOperand) const
{
    int allCents1 = cents+dollars*100;
    int allCents2 = secondOperand.cents+secondOperand.dollars*100;
    int sumAllCents = allCents1 + allCents2;
    int absAllCents = abs(sumAllCents); //Money can be negative.
    int finalDollars = absAllCents / 100;
    int finalCents = absAllCents % 100;
    if (sumAllCents < 0)
    {
        finalDollars = -finalDollars;
        finalCents = -finalCents;
    }
    return Money(finalDollars, finalCents);
}

const Money Money::operator -(const Money& secondOperand) const
{
    int allCents1 = cents+dollars*100;
    int allCents2 =  secondOperand.cents+secondOperand.dollars*100;
    int diffAllCents = allCents1 - allCents2;
    int absAllCents = abs(diffAllCents);
    int finalDollars = absAllCents / 100;
    int finalCents = absAllCents % 100;
    if (diffAllCents < 0)
    {
        finalDollars = -finalDollars;
        finalCents = -finalCents; }
    return Money(finalDollars, finalCents);
}

bool Money::operator ==(const Money& secondOperand) const  {
    return ((dollars == secondOperand.dollars)
            && (cents == secondOperand.cents));
}

const Money Money::operator -( ) const  {
    return Money(-dollars, -cents);
}

Money::Money( ): dollars(0), cents(0)
{/*Body intentionally empty.*/}

Money::Money(double amount)
: dollars(dollarsPart(amount)), cents(centsPart(amount))
{/*Body intentionally empty*/}

Money::Money(int theDollars)
:  dollars(theDollars), cents(0)
{/*Body intentionally empty*/}

//Uses cstdlib:
Money::Money(int theDollars, int theCents)  {
    if ((theDollars < 0 && theCents > 0) ||
        (theDollars > 0 && theCents < 0))
    {
        cout << "Inconsistent money data.\n"; exit(1);
    }
    dollars = theDollars;
    cents = theCents;
}
```

```
// ----------- Money ------------

double Money::getAmount( ) const {
    return (dollars + cents*0.01);  }

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

int Money::getCents( ) const
{
    return cents;
}

//Uses iostream and cstdlib:
void Money::output( ) const
{
    int absDollars = abs(dollars);
    int absCents = abs(cents);
    if (dollars < 0 || cents < 0)
        //accounts for dollars == 0 or cents == 0
        cout << "$-";
    else
        cout << '$';
    cout << absDollars;
    if (absCents >= 10)
        cout << '.' << absCents;
    else
        cout << '.' << '0' << absCents;
}


//Uses iostream and cstdlib:
void Money::input( )
{
    char dollarSign;
    cin >> dollarSign; //hopefully
    if (dollarSign != '$')
    {
        cout << "No dollar sign in Money input.\n";
        exit(1);
    }
    double amountAsDouble;
    cin >> amountAsDouble;
    dollars = dollarsPart(amountAsDouble);
    cents = centsPart(amountAsDouble);
}

int Money::dollarsPart(double amount) const {
    return static_cast<int>(amount);
}

int Money::centsPart(double amount) const{
    double doubleCents = amount * 100;
    int intCents = (round(fabs(doubleCents))) % 100;
    //% can misbehave on negatives
    if (amount < 0)
        intCents = -intCents;
    return intCents;
}

int Money::round(double number) const{
    return static_cast<int>(floor(number + 0.5));
}
```

```cpp
//
//  main.cpp
//  AbsoluteCpp_ch8_3
//
//  Created by Liwei on 2020/3/20.
//  Copyright © 2020 Liwei. All rights reserved.
//
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;

//Class for amounts of money in U.S. currency.
class Money
{
public:
    Money( );
    Money(double amount);
    Money(int dollars, int cents);
    Money(int dollars);
    double getAmount( ) const;
    int getDollars( ) const;
    int getCents( ) const;
    void input( ); //Reads the dollar sign as well as the amount number.
    void output( ) const;
    friend const Money operator +(const Money& amount1, const Money&
     amount2);
    friend const Money operator -(const Money& amount1, const Money&
     amount2);
    friend bool operator ==(const Money& amount1, const Money& amount2);
    friend const Money operator -(const Money& amount);
private:
    int dollars; //A negative amount is represented as negative dollars and
    int cents; //negative cents. Negative $4.50 is represented as -4 and -50

    int dollarsPart(double amount) const;
    int centsPart(double amount) const;
    int round(double number) const;
};


int main( )
{
    Money yourAmount, myAmount(10, 9);
    cout << "Enter an amount of money: ";
    yourAmount.input( );

    cout << "Your amount is ";
    yourAmount.output( );
    cout << endl;
    cout << "My amount is ";
    myAmount.output( );
    cout << endl;

    if (yourAmount == myAmount)
```

```cpp
        cout << "We have the same amounts.\n";
    else
        cout << "One of us is richer.\n";

    Money ourAmount = yourAmount + myAmount;
    yourAmount.output( ); cout << " + "; myAmount.output( );
    cout << " equals "; ourAmount.output( ); cout << endl;

    Money diffAmount = yourAmount - myAmount;
    yourAmount.output( ); cout << " - "; myAmount.output( );
    cout << " equals "; diffAmount.output( ); cout << endl;

    return 0;
}


const Money operator +(const Money& amount1, const Money& amount2)
{
    int allCents1 = amount1.cents + amount1.dollars*100;
    int allCents2 = amount2.cents + amount2.dollars*100;
    int sumAllCents = allCents1 + allCents2;
    int absAllCents = abs(sumAllCents); //Money can be negative.
    int finalDollars = absAllCents/100;
    int finalCents = absAllCents%100;

    if (sumAllCents < 0)
    {
        finalDollars = -finalDollars;
        finalCents = -finalCents;
    }

    return Money(finalDollars, finalCents);
}

//Uses cstdlib:
const Money operator -(const Money& amount1, const Money& amount2)
{
    int allCents1 = amount1.cents + amount1.dollars*100;
    int allCents2 = amount2.cents + amount2.dollars*100;
    int diffAllCents = allCents1 - allCents2;
    int absAllCents = abs(diffAllCents);

    int finalDollars = absAllCents/100;
    int finalCents = absAllCents%100;

    if (diffAllCents < 0)
    {
        finalDollars = -finalDollars;
        finalCents = -finalCents;
  }

    return Money(finalDollars, finalCents);
}

bool operator ==(const Money& amount1, const Money& amount2)
```

```
{
    return ((amount1.dollars == amount2.dollars)
           && (amount1.cents == amount2.cents));
}

const Money operator -(const Money& amount)
{
    return Money(-amount.dollars, -amount.cents);
}


Money::Money( ): dollars(0), cents(0)
{/*Body intentionally empty.*/}

Money::Money(double amount)
              : dollars(dollarsPart(amount)), cents(centsPart(amount))
{/*Body intentionally empty*/}

Money::Money(int theDollars)
              : dollars(theDollars), cents(0)
{/*Body intentionally empty*/}

//Uses cstdlib:
Money::Money(int theDollars, int theCents)
{
    if ((theDollars < 0 && theCents > 0) || (theDollars > 0 && theCents <
     0))
    {
        cout << "Inconsistent money data.\n";
        exit(1);
    }
    dollars = theDollars;
    cents = theCents;
}

double Money::getAmount( ) const
{
    return (dollars + cents*0.01);
}

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

int Money::getCents( ) const
{
    return cents;
}

//Uses iostream and cstdlib:
void Money::output( ) const
{
    int absDollars = abs(dollars);
    int absCents = abs(cents);
    if (dollars < 0 || cents < 0)//accounts for dollars == 0 or cents == 0
        cout << "$-";
    else
        cout << '$';
    cout << absDollars;

    if (absCents >= 10)
        cout << '.' << absCents;
    else
        cout << '.' << '0' << absCents;
}

//Uses iostream and cstdlib:
void Money::input( )
{
    char dollarSign;
    cin >> dollarSign; //hopefully
    if (dollarSign != '$')
    {
        cout << "No dollar sign in Money input.\n";
        exit(1);
    }

    double amountAsDouble;
    cin >> amountAsDouble;
    dollars = dollarsPart(amountAsDouble);
    cents = centsPart(amountAsDouble);
}

int Money::dollarsPart(double amount) const
{
    return static_cast<int>(amount);
}

int Money::centsPart(double amount) const
{
    double doubleCents = amount*100;
    int intCents = (round(fabs(doubleCents)))%100;//% can misbehave on
     negatives
    if (amount < 0)
        intCents = -intCents;
    return intCents;
}

int Money::round(double number) const
{
    return static_cast<int>(floor(number + 0.5));
}
```

```cpp
/*
 * AbsoluteCpp_ch8_4
 *
 *  Created on: 2019Äê3ÔÂ9ÈÕ
 *      Author: Skuller
 */

#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

double& sampleFunction(double& variable);
double& sampleFunction1(double variable);

int main() {
    double a = 99;
    cout << "a is 99, and call sampleFunction result is " <<
     sampleFunction(a)
            << endl;

    sampleFunction(a) = 50;
    cout << "a is " << a << endl;

    a = 99;
    cout << "a is 99, and call sampleFunction result is " <<
     sampleFunction(a)
            << endl;
    double& d = sampleFunction(a);
    a = 45;
    cout << "d is " << d << endl;

    double b = 100;
    double c = sampleFunction1(b);
    cout << "b is 100, and call sampleFunction1 result is "
            << sampleFunction1(b) << endl;
    cout << "c is " << c << endl;
    b = 200;
    cout << "c is " << c << endl;
    return 0;
}

// Note this function return a reference of argument
double& sampleFunction(double& variable) {
    return variable;
}

double& sampleFunction1(double variable) {
    double& result = variable;
    return result;
}
```

```cpp
//
//  main.cpp
//  AbsoluteCpp_ch8_5
//

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

//Class for amounts of money in U.S. currency.
class Money
{
public:
    Money( );
    Money(double amount);
    Money(int theDollars, int theCents);
    Money(int theDollars);
    double getAmount( ) const;
    int getDollars( ) const;
    int getCents( ) const;
    friend const Money operator +(const Money& amount1, const Money&
     amount2);
    friend const Money operator -(const Money& amount1, const Money&
     amount2);
    friend bool operator ==(const Money& amount1, const Money& amount2);
    friend const Money operator -(const Money& amount);
    friend ostream& operator <<(ostream& outputStream, const Money& amount);
    friend istream& operator >>(istream& inputStream, Money& amount);
private:
    int dollars; //A negative amount is represented as negative dollars and
    int cents; //negative cents. Negative $4.50 is represented as -4 and -50

    int dollarsPart(double amount) const;
    int centsPart(double amount) const;
    int round(double number) const;
};

int main( )
{
    Money yourAmount, myAmount(10, 9);
    cout << "Enter an amount of money: ";
    cin >> yourAmount;
    cout << "Your amount is " << yourAmount << endl;
    cout << "My amount is " << myAmount << endl;

    if (yourAmount == myAmount)
        cout << "We have the same amounts.\n";
    else
        cout << "One of us is richer.\n";

    Money ourAmount = yourAmount + myAmount;
    cout << yourAmount << " + " << myAmount
         << " equals " << ourAmount << endl;
    Money diffAmount = yourAmount - myAmount;
    cout << yourAmount << " - " << myAmount
         << " equals " << diffAmount << endl;

    return 0;
}

ostream& operator <<(ostream& outputStream, const Money& amount)
{
    int absDollars = abs(amount.dollars);
    int absCents = abs(amount.cents);
    if (amount.dollars < 0 || amount.cents < 0)
        //accounts for dollars == 0 or cents == 0
        outputStream << "$-";
    else
        outputStream << '$';
    outputStream << absDollars;

    if (absCents >= 10)
        outputStream << '.' << absCents;
    else
        outputStream << '.' << '0' << absCents;

    return outputStream;
}

//Uses iostream and cstdlib:
istream& operator >>(istream& inputStream, Money& amount)
{
    char dollarSign;
    inputStream >> dollarSign; //hopefully
    if (dollarSign != '$')
    {
        cout << "No dollar sign in Money input.\n";
        exit(1);
    }


    double amountAsDouble;
    inputStream >> amountAsDouble;
    amount.dollars = amount.dollarsPart(amountAsDouble);
    amount.cents = amount.centsPart(amountAsDouble);


    return inputStream;
}

const Money operator +(const Money& amount1, const Money& amount2)
{
    int allCents1 = amount1.cents + amount1.dollars*100;
    int allCents2 = amount2.cents + amount2.dollars*100;
    int sumAllCents = allCents1 + allCents2;
    int absAllCents = abs(sumAllCents); //Money can be negative.
    int finalDollars = absAllCents/100;
    int finalCents = absAllCents%100;
    if (sumAllCents < 0)
    {
        finalDollars = -finalDollars;
        finalCents = -finalCents;
    }

    return Money(finalDollars, finalCents);
}

//Uses cstdlib:
const Money operator -(const Money& amount1, const Money& amount2)
{
    int allCents1 = amount1.cents + amount1.dollars*100;
    int allCents2 = amount2.cents + amount2.dollars*100;
    int diffAllCents = allCents1 - allCents2;
    int absAllCents = abs(diffAllCents);

    int finalDollars = absAllCents/100;
    int finalCents = absAllCents%100;

    if (diffAllCents < 0)
    {
        finalDollars = -finalDollars;
        finalCents = -finalCents;
  }

    return Money(finalDollars, finalCents);
}

bool operator ==(const Money& amount1, const Money& amount2)
{
    return ((amount1.dollars == amount2.dollars)
           && (amount1.cents == amount2.cents));
}

const Money operator -(const Money& amount)
{
    return Money(-amount.dollars, -amount.cents);
}


Money::Money( ): dollars(0), cents(0)
{/*Body intentionally empty.*/}

Money::Money(double amount)
              : dollars(dollarsPart(amount)), cents(centsPart(amount))
{/*Body intentionally empty*/}

Money::Money(int theDollars)
              : dollars(theDollars), cents(0)
{/*Body intentionally empty*/}

//Uses cstdlib:
Money::Money(int theDollars, int theCents)
```

```
{
    if ((theDollars < 0 && theCents > 0) || (theDollars > 0 && theCents <
     0))
    {
        cout << "Inconsistent money data.\n";
        exit(1);
    }
    dollars = theDollars;
    cents = theCents;
}

double Money::getAmount( ) const
{
    return (dollars + cents*0.01);
}

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

int Money::getCents( ) const
{
    return cents;
}

int Money::dollarsPart(double amount) const
{
    return static_cast<int>(amount);
}

int Money::centsPart(double amount) const
{
    double doubleCents = amount*100;
    int intCents = (round(fabs(doubleCents)))%100;//% can misbehave on
     negatives
    if (amount < 0)
        intCents = -intCents;
    return intCents;
}

int Money::round(double number) const
{
    return static_cast<int>(floor(number + 0.5));
}
```

```
//
//  main.cpp
//  AbsoluteCpp_ch8_6
//

#include <iostream>
#include <cstdlib>
using namespace std;

class IntPair
{
public:
    IntPair(int firstValue, int secondValue);
    IntPair operator++( ); //Prefix version
    IntPair operator++(int); //Postfix version
    void setFirst(int newValue);
    void setSecond(int newValue);
    int getFirst( ) const;
    int getSecond( ) const;
private:
    int first;
    int second;
};

int main( )
{
    IntPair a(1,2);
    cout << "Postfix a++: Start value of object a: ";
    cout << a.getFirst( ) << " " << a.getSecond( ) << endl;
    IntPair b = a++;
    cout << "Value returned: ";
    cout << b.getFirst( ) << " " << b.getSecond( ) << endl;
    cout << "Changed object: ";
    cout << a.getFirst( ) << " " << a.getSecond( ) << endl;

    a = IntPair(1, 2);
    cout << "Prefix ++a: Start value of object a: ";
    cout << a.getFirst( ) << " " << a.getSecond( ) << endl;
    IntPair c = ++a;
    cout << "Value returned: ";
    cout << c.getFirst( ) << " " << c.getSecond( ) << endl;
    cout << "Changed object: ";
    cout << a.getFirst( ) << " " << a.getSecond( ) << endl;
    return 0;
}

IntPair::IntPair(int firstValue, int secondValue)
                     : first(firstValue), second(secondValue)
{/*Body intentionally empty*/}

IntPair IntPair::operator++(int ignoreMe) //postfix version
{
    int temp1 = first;
    int temp2 = second;
    first++;
    second++;
    return IntPair(temp1, temp2);
}

IntPair IntPair::operator++( ) //prefix version
{
    first++;
    second++;
    return IntPair(first, second);
}

void IntPair::setFirst(int newValue)
{
    first = newValue;
}

void IntPair::setSecond(int newValue)
{
    second = newValue;
}

int IntPair::getFirst( ) const
{
    return first;
}

int IntPair::getSecond( ) const
{
    return second;
}
```

```
//
//  main.cpp
//  AbsoluteCpp_ch8_7
//

#include <iostream>
#include <cstdlib>
using namespace std;

class CharPair
{
public:
    CharPair( ){/*Body intentionally empty*/}
    CharPair(char firstValue, char secondValue)
                   : first(firstValue), second(secondValue)
    {/*Body intentionally empty*/}

    char& operator[](int index);
private:
    char first;
    char second;
};

int main( )
{
    CharPair a;
    a[1] = 'A';
    a[2] = 'B';
    cout << "a[1] and a[2] are:\n";
    cout << a[1] << a[2] << endl;

    cout << "Enter two letters (no spaces):\n";
    cin >> a[1] >> a[2];
    cout << "You entered:\n";
    cout << a[1] << a[2] << endl;

    return 0;
}

//Uses iostream and cstdlib:
char& CharPair::operator[](int index)
{
    if (index == 1)
        return first;
    else if (index == 2)
        return second;
    else
    {
        cout << "Illegal index value.\n";
        exit(1);
    }
}
```