```
//
//  main.cpp
//  AbsoluteCpp_ch7_1
//

#include <iostream>
#include <cstdlib> //for exit
using namespace std;

class DayOfYear
{
public:
    DayOfYear(int monthValue, int dayValue);
    //Initializes the month and day to arguments.

    DayOfYear(int monthValue);
    //Initializes the date to the first of the given month.

    DayOfYear( );
    //Initializes the date to January 1.

    void input( );
    void output( );
    int getMonthNumber( );
    //Returns 1 for January, 2 for February, etc.

    int getDay( );
private:
    int month;
    int day;
    void testDate( );
};

int main( )
{
    DayOfYear date1(2, 21), date2(5), date3;
    cout << "Initialized dates:\n";
    date1.output( ); cout << endl;
    date2.output( ); cout << endl;
    date3.output( ); cout << endl;

    date1 = DayOfYear(10, 31);
    cout << "date1 reset to the following:\n";
    date1.output( ); cout << endl;
    return 0;
}


DayOfYear::DayOfYear(int monthValue, int dayValue)
                          : month(monthValue), day(dayValue)
{
    testDate( );
}

DayOfYear::DayOfYear(int monthValue) : month(monthValue), day(1)
```

```
{
    testDate( );
}

DayOfYear::DayOfYear( ) : month(1), day(1)
{/*Body intentionally empty.*/}

//uses iostream and cstdlib:
void DayOfYear::testDate( )
{
    if ((month < 1) || (month > 12))
    {
        cout << "Illegal month value!\n";
        exit(1);
    }
    if ((day < 1) || (day > 31))
    {
        cout << "Illegal day value!\n";
        exit(1);
    }
}

int DayOfYear::getMonthNumber( )
{
    return month;
}

int DayOfYear::getDay( )
{
    return day;
}

//Uses iostream and cstdlib:
void DayOfYear::input( )
{
    cout << "Enter the month as a number: ";
    cin >> month;
    cout << "Enter the day of the month: ";
    cin >> day;
    if ((month < 1) || (month > 12) || (day < 1) || (day > 31))
    {
        cout << "Illegal date! Program aborted.\n";
        exit(1);
    }
}

void DayOfYear::output( )
{
    switch (month)
    {
        case 1:
            cout << "January "; break;
        case 2:
            cout << "February "; break;
        case 3:
```

```
            cout << "March "; break;
        case 4:
            cout << "April "; break;
        case 5:
            cout << "May "; break;
        case 6:
            cout << "June "; break;
        case 7:
            cout << "July "; break;
        case 8:
            cout << "August "; break;
        case 9:
            cout << "September "; break;
        case 10:
            cout << "October "; break;
        case 11:
            cout << "November "; break;
        case 12:
            cout << "December "; break;
        default:
            cout << "Error in DayOfYear::output. Contact software vendor.";
    }

    cout << day;
}
```

```
//
//  main.cpp
//  AbsoluteCpp_ch7_2
//


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

//Data consists of two items, an amount of money for the account balance
//and a percent for the interest rate.
class BankAccount
{
public:
    BankAccount(double balance, double rate);
    //Initializes balance and rate according to arguments.

    BankAccount(int dollars, int cents, double rate);
    //Initializes the account balance to $dollars.cents. For a negative
     balance both
    //dollars and cents must be negative. Initializes the interest rate to
     rate percent.

    BankAccount(int dollars, double rate);
    //Initializes the account balance to $dollars.00 and
    //initializes the interest rate to rate percent.

    BankAccount( );
    //Initializes the account balance to $0.00 and the interest rate to
     0.0%.

    void update( );
    //Postcondition: One year of simple interest has been added to the
     account.
    void input( );
    void output( );
    double getBalance( );
    int getDollars( );
    int getCents( );
    double getRate( );//Returns interest rate as a percent.

    void setBalance(double balance);
    void setBalance(int dollars, int cents);
    //checks that arguments are both nonnegative or both nonpositive

    void setRate(double newRate);
    //If newRate is nonnegative, it becomes the new rate. Otherwise abort
     program.

 private:
    //A negative amount is represented as negative dollars and negative
     cents.
```

```
    //For example, negative $4.50 sets accountDollars to -4 and
     accountCents to -50.
    int accountDollars; //of balance
    int accountCents; //of balance
    double rate;//as a percent

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

    double fraction(double percent);
    //Converts a percent to a fraction. For example, fraction(50.3) returns
     0.503.
};

int main( )
{
    BankAccount account1(1345.52, 2.3), account2;
    cout << "account1 initialized as follows:\n";
    account1.output( );
    cout << "account2 initialized as follows:\n";
    account2.output( );

    account1 = BankAccount(999, 99, 5.5);
    cout << "account1 reset to the following:\n";
    account1.output( );

    cout << "Enter new data for account 2:\n";
    account2.input( );
    cout << "account2 reset to the following:\n";
    account2.output( );

    account2.update( );
    cout << "In one year account2 will grow to:\n";
    account2.output( );

    return 0;
}

BankAccount::BankAccount(double balance, double rate)
 : accountDollars(dollarsPart(balance)), accountCents(centsPart(balance))
{
    setRate(rate);
}

BankAccount::BankAccount(int dollars, int cents, double rate)
{
    setBalance(dollars, cents);
    setRate(rate);
}

BankAccount::BankAccount(int dollars, double rate)
                              : accountDollars(dollars), accountCents(0)
{
    setRate(rate);
```

```
}

BankAccount::BankAccount( ): accountDollars(0), accountCents(0), rate(0.0)
{/*Body intentionally empty.*/}

void BankAccount::update( )
{
    double balance = accountDollars + accountCents*0.01;
    balance = balance + fraction(rate)*balance;
    accountDollars = dollarsPart(balance);
    accountCents = centsPart(balance);
}

//Uses iostream:
void BankAccount::input( )
{
    double balanceAsDouble;
    cout << "Enter account balance $";
    cin >> balanceAsDouble;
    accountDollars = dollarsPart(balanceAsDouble);
    accountCents = centsPart(balanceAsDouble);
    cout << "Enter interest rate (NO percent sign): ";
    cin >> rate;
    setRate(rate);
}

//Uses iostream and cstdlib:
void BankAccount::output( )
{
    int absDollars = abs(accountDollars);
    int absCents = abs(accountCents);
    cout << "Account balance: $";
    if (accountDollars < 0)
        cout << "-";
    cout << absDollars;
    if (absCents >= 10)
        cout << "." << absCents << endl;
    else
        cout << "." << '0' << absCents << endl;

    cout << "Rate: " << rate << "%\n";
}

double BankAccount::getBalance( )
{
    return (accountDollars + accountCents*0.01);
}

int BankAccount::getDollars( )
{
    return accountDollars;
}

int BankAccount::getCents( )
{
```

```
    return accountCents;
}

double BankAccount::getRate( )
{
    return rate;
}

void BankAccount::setBalance(double balance)
{
    accountDollars = dollarsPart(balance);
    accountCents = centsPart(balance);
}

//Uses cstdlib:
void BankAccount::setBalance(int dollars, int cents)
{
    if ((dollars < 0 && cents > 0) || (dollars > 0 && cents < 0))
    {
        cout << "Inconsistent account data.\n";
        exit(1);
    }
    accountDollars = dollars;
    accountCents = cents;
}

//Uses cstdlib:
void BankAccount::setRate(double newRate)
{
    if (newRate >= 0.0)
        rate = newRate;
    else
    {
        cout << "Cannot have a negative interest rate.\n";
        exit(1);
    }
}

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

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

//Uses cmath:
```

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

double BankAccount::fraction(double percent)
{
    return (percent/100.0);
}
```

```
//
//  main.cpp
//  AbsoluteCpp_ch7_3
//
//

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

class DayOfYear {
public:
    DayOfYear(int monthValue, int dayValue);
    DayOfYear(int monthValue);

    DayOfYear( );
    void input( );
    void output( );
    int getMonthNumber( );
    int getDay( );
private:
    int month;
    int day;
    void testDate( );
};

class Holiday
{
public:
    Holiday( );//Initializes to January 1 with no parking enforcement
    Holiday(int month, int day, bool theEnforcement);
    void output( );
private:
    DayOfYear date;
    bool parkingEnforcement;//true if enforced
};



int main(int argc, const char * argv[]) {
    Holiday h(2, 14, true);
    cout << "Testing the class Holiday.\n";
    h.output( );
    return 0;
}

Holiday::Holiday( ) : date(1, 1), parkingEnforcement(false)
 {/*Intentionally empty*/}

Holiday::Holiday(int month, int day, bool theEnforcement)
:date(month, day), parkingEnforcement(theEnforcement)
{/*Intentionally empty*/}

void Holiday::output( )  {
    date.output( );
```

```
    cout << endl;
    if (parkingEnforcement)
        cout << "Parking laws will be enforced.\n";
    else
        cout << "Parking laws will not be enforced.\n";
}

DayOfYear::DayOfYear(int monthValue, int dayValue)
: month(monthValue), day(dayValue)
{
    testDate( );
}

//uses iostream and cstdlib:
void DayOfYear::testDate( )
{
    if ((month < 1) || (month > 12))  {
        cout << "Illegal month value!\n";
        exit(1);
    }
    if ((day < 1) || (day > 31))
    {
        cout << "Illegal day value!\n";
        exit(1);
    }
}

//Uses iostream:
void DayOfYear::output( )  {
    switch (month)
    {
        case 1:
            cout << "January "; break;
        case 2:
            cout << "February "; break;
        case 3:
            cout << "March "; break;
        case 4:
            cout << "April "; break;
        case 5:
            cout << "May "; break;
        case 6:
            cout << "June "; break;
        case 7:
            cout << "July "; break;
        case 8:
            cout << "August "; break;
        case 9:
            cout << "September "; break;
        case 10:
            cout << "October "; break;
        case 11:
            cout << "November "; break;
        case 12:
            cout << "December "; break;
```

```
        default:
            cout << "Error in DayOfYear::output.";
    }

    cout << day;
}
```

```
//
//  main.cpp
//  AbsoluteCpp_ch7_4
//
//

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

//Data consists of two items: an amount of money for the account balance
//and a percentage for the interest rate.
class BankAccount
{
public:
    BankAccount(double balance, double rate);
    //Initializes balance and rate according to arguments.
    BankAccount(int dollars, int cents, double rate);
    //Initializes the account balance to $dollars.cents. For a negative
    //balance both dollars and cents must be negative. Initializes the
    //interest rate to rate percent.
    BankAccount(int dollars, double rate);
    //Initializes the account balance to $dollars.00 and
    //initializes the interest rate to rate percent.

    BankAccount( );
    //Initializes the account balance to $0.00 and the interest rate
    //to 0.0%.
    void update( );
    //Postcondition: One year of simple interest has been added to the
    //account.
    void input( );
    void output( ) const;
    double getBalance( ) const;
    int getDollars( ) const;
    int getCents( ) const;
    double getRate( ) const;

    void setBalance(double balance);
    void setBalance(int dollars, int cents);
    //Checks that arguments are both nonnegative or both nonpositive.
    void setRate(double newRate);
    //If newRate is nonnegative, it becomes the new rate. Otherwise,
    //abort program.

private:
    //A negative amount is represented as negative dollars and negative
     cents.
    //For example, negative $4.50 sets accountDollars to -4 and accountCents
    //to -50.
    int accountDollars; //of balance
    int accountCents; //of balance
    double rate;//as a percent
    int dollarsPart(double amount) const;
```

```
    int centsPart(double amount) const;
    int round(double number) const;

    double fraction(double percent) const;
    //Converts a percentage to a fraction. For example, fraction(50.3)
    //returns 0.503.
};

//Returns true if the balance in account1 is greater than that
//in account2. Otherwise returns false.
bool isLarger(const BankAccount& account1, const BankAccount& account2);

void welcome(const BankAccount& yourAccount);

int main() {
    BankAccount account1(6543.21, 4.5), account2; welcome(account1);
    cout << "Enter data for account 2:\n"; account2.input( );
    if (isLarger(account1, account2))
        cout << "account1 is larger.\n";
    else
        cout << "account2 is at least as large as account1.\n";

    return 0;
}

bool isLarger(const BankAccount& account1, const BankAccount& account2)  {
 return(account1.getBalance( ) > account2.getBalance( ));
 }
 void welcome(const BankAccount& yourAccount)
{
    cout << "Welcome to our bank.\n"
    << "The status of your account is: \n";
    yourAccount.output();
}

BankAccount::BankAccount(double balance, double rate)
: accountDollars(dollarsPart(balance)),
accountCents(centsPart(balance))
{
    setRate(rate);
}

BankAccount::BankAccount(int dollars, int cents, double rate)
{
    setBalance(dollars, cents);
    setRate(rate);
}

BankAccount::BankAccount(int dollars, double rate)
: accountDollars(dollars), accountCents(0)
{
    setRate(rate);
}

BankAccount::BankAccount( ): accountDollars(0), accountCents(0), rate(0.0)
```

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


//Uses iostream:
void BankAccount::input( )  {
    double balanceAsDouble;
    cout << "Enter account balance $";
    cin >> balanceAsDouble;
    accountDollars = dollarsPart(balanceAsDouble);
    accountCents = centsPart(balanceAsDouble);
    cout << "Enter interest rate (NO percent sign): ";
    cin >> rate;
    setRate(rate);
}

//Uses iostream and cstdlib:
void BankAccount::output( ) const {
    int absDollars = abs(accountDollars);
    int absCents = abs(accountCents);
    cout << "Account balance: $";
    if (accountDollars > 0)
        cout << "-";
    cout << absDollars;
    if (absCents >= 10)
        cout << "." << absCents << endl;
    else
        cout << "." << '0' << absCents << endl;

    cout << "Rate: " << rate << "%\n";
}

double BankAccount::getBalance( ) const
{
    return (accountDollars + accountCents * 0.01);
}

int BankAccount::getDollars() const {
    return accountDollars;
}

int BankAccount::getCents() const {
    return accountCents;
}

double BankAccount::getRate() const {
    return rate;
}

void BankAccount::setBalance(double balance)
{
    accountDollars = dollarsPart(balance);
    accountCents = centsPart(balance);
}

//Uses cstdlib:
```

```
void BankAccount::setBalance(int dollars, int cents)  {

    if((dollars < 0 && cents > 0) || (dollars > 0 && cents < 0))
    {
        cout << "Inconsistent account data.\n";
        exit(1);
    }
    accountDollars = dollars;
    accountCents = cents;
}

//Uses cstdlib:
void BankAccount::setRate(double newRate)  {
    if(newRate >= 0.0)
        rate = newRate;
    else{
        cout<< "Cannot have a negative interest rate.\n";
        exit(1);
    }
}

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

//Uses cmath:
int BankAccount::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;
}

//Uses cmath:
int BankAccount::round(double number) const {
    return static_cast<int>(floor(number + 0.5));
}

double BankAccount::fraction(double percent) const {
    return (percent/100.0);
}
```

```
//
//  main.cpp
//  AbsoluteCpp_ch7_5
//
//

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

class BankAccount {
public:
    BankAccount(double balance, double rate);
    BankAccount(int dollars, int cents, double rate);
    BankAccount(int dollars, double rate);
    BankAccount( );
    void update( );
    void input( );
    void output( ) const;

    // inline
    double getBalance( ) const { return (accountDollars +
     accountCents*0.01);}
    int getDollars( ) const { return accountDollars; }
    int getCents( ) const { return accountCents; }
    double getRate( ) const { return rate; }

    void setBalance(double balance);
    void setBalance(int dollars, int cents);
    void setRate(double newRate);

private:
    int accountDollars; //of balance int accountCents;
    int accountCents; //of balance double rate;
    double rate; //as a percentage

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

    int centsPart(double amount) const;

    // inline
    int round(double number) const
    { return static_cast<int>(floor(number + 0.5)); }

    // inline
    double fraction(double percent) const { return (percent / 100.0); }
};


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

```
//
//  main.cpp
//  AbsoluteCpp_ch7_5
//
//

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

class BankAccount
{
 public:
    BankAccount(double balance, double rate);
    BankAccount(int dollars, int cents, double rate);
    BankAccount(int dollars, double rate);
    BankAccount( );
    void update( );
    void input( );
    void output( ) const;

    double getBalance( ) const { return (accountDollars +
     accountCents*0.01);}

    int getDollars( ) const { return accountDollars; }

    int getCents( ) const { return accountCents; }

    double getRate( ) const { return rate; }

    void setBalance(double balance);
    void setBalance(int dollars, int cents);
    void setRate(double newRate);
 private:
    int accountDollars; //of balance
    int accountCents; //of balance
    double rate;//as a percent

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

    int centsPart(double amount) const;

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

    double fraction(double percent) const { return (percent/100.0); }
};

//Returns true if the balance in account1 is greater than that
//in account2. Otherwise returns false.
bool isLarger(const BankAccount& account1, const BankAccount& account2);
```

```
void welcome(const BankAccount& yourAccount);

int main( )
{
    BankAccount account1(6543.21, 4.5), account2;
    welcome(account1);
    cout << "Enter data for account 2:\n";
    account2.input( );
    if (isLarger(account1, account2))
        cout << "account1 is larger.\n";
    else
        cout << "account2 is at least as large as account1.\n";

        return 0;
}

bool isLarger(const BankAccount& account1, const BankAccount& account2)
{
    return(account1.getBalance( ) > account2.getBalance( ));
}

void welcome(const BankAccount& yourAccount)
{
    cout << "Welcome to our bank.\n"
         << "The status of your account is:\n";
    yourAccount.output( );
}

//Uses iostream and cstdlib:
void BankAccount::output( ) const
{
    int absDollars = abs(accountDollars);
    int absCents = abs(accountCents);
    cout << "Account balance: $";
    if (accountDollars < 0)
        cout << "-";
    cout << absDollars;
    if (absCents >= 10)
        cout << "." << absCents << endl;
    else
        cout << "." << '0' << absCents << endl;

    cout << "Rate: " << rate << "%\n";
}



BankAccount::BankAccount(double balance, double rate)
 : accountDollars(dollarsPart(balance)), accountCents(centsPart(balance))
{
    setRate(rate);
}

BankAccount::BankAccount(int dollars, int cents, double rate)
{
```

```
    setBalance(dollars, cents);
    setRate(rate);
}

BankAccount::BankAccount(int dollars, double rate)
                              : accountDollars(dollars), accountCents(0)
{
    setRate(rate);
}

BankAccount::BankAccount( ): accountDollars(0), accountCents(0), rate(0.0)
{/*Body intentionally empty.*/}

void BankAccount::update( )
{
    double balance = accountDollars + accountCents*0.01;
    balance = balance + fraction(rate)*balance;
    accountDollars = dollarsPart(balance);
    accountCents = centsPart(balance);
}

//Uses iostream:
void BankAccount::input( )
{
    double balanceAsDouble;
    cout << "Enter account balance $";
    cin >> balanceAsDouble;
    accountDollars = dollarsPart(balanceAsDouble);
    accountCents = centsPart(balanceAsDouble);
    cout << "Enter interest rate (NO percent sign): ";
    cin >> rate;
    setRate(rate);
}

void BankAccount::setBalance(double balance)
{
    accountDollars = dollarsPart(balance);
    accountCents = centsPart(balance);
}

//Uses cstdlib:
void BankAccount::setBalance(int dollars, int cents)
{
    if ((dollars < 0 && cents > 0) || (dollars > 0 && cents < 0))
    {
        cout << "Inconsistent account data.\n";
        exit(1);
    }
    accountDollars = dollars;
    accountCents = cents;
}

//Uses cstdlib:
void BankAccount::setRate(double newRate)
{
```

```
    if (newRate >= 0.0)
        rate = newRate;
    else
    {
        cout << "Cannot have a negative interest rate.\n";
        exit(1);
    }
}

//Uses cmath:
int BankAccount::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;
}
```

```cpp
//
//  main.cpp
//  AbsoluteCpp_ch7_7
//


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

int main( )
{
    vector<int> v;
    cout << "Enter a list of positive numbers.\n"
         << "Place a negative number at the end.\n";

    int next;
    cin >> next;
    while (next > 0)
    {
        v.push_back(next);
        cout << next << " added. ";
        cout << "v.size( ) = " << v.size( ) << endl;
        cin >> next;
    }

    cout << "You entered:\n";
    for (unsigned int i = 0; i < v.size( ); i++)
        cout << v[i] << " ";
    cout << endl;

    return 0;
}
```