1. Rational

// Rational.h

#ifndef RATIONAL\_H\_

#define RATIONAL\_H\_

#include <stdio.h>

class Rational {

private:

int numerator;

int denominator;

public:

//Constructor & Destructor

Rational( int numerator = 0, int denominator = 1);

~Rational();

//Getters and Setters

int getNum() const;

int getDen() const;

void setNum(int a);

void setDen(int b);

//Math

void Add(Rational a1);

void Subtract(Rational s1);

void Multiply(Rational m1);

void Divide(Rational d1);

Rational GCD(Rational g1);

double CalculateFloat();

//Display

void DisplayFract() const;

void DisplayFloat();

//Copy Constructor

Rational(const Rational &aRational);

//Overloaded Operators

Rational operator + (const Rational &tempRational);// addition operator

Rational operator - (const Rational &tempRational);// subtraction operator

Rational operator \* (const Rational &tempRational);// multiplication operator

Rational operator / (const Rational &tempRational);// division operator

Rational &operator=(const Rational &);//assign operator

bool operator >(Rational &tempRational);// is greater than operator

bool operator <(Rational &tempRational);//is less than operator

bool operator ==(const Rational &);//is equal to operator

friend ostream &operator<<(ostream &, const Rational &tempRational);//cout operator

friend istream &operator>>(std::istream &, Rational &);//cin operator

};

#endif /\* Rational\_h \*/

// Rational.cpp

#include <iostream>

using namespace std;

#include "Rational.h"

//Overloaded Operators

double Rational::CalculateFloat() {

double q1 = (double)this->getNum();

double q2 = (double)this->getDen();

return q1 / q2;

}

Rational Rational::operator + (const Rational &tempRational) {

int cd = this->getDen() \* tempRational.getDen();

int Num1 = this->getNum() \* tempRational.getDen();

int Num2 = tempRational.getNum() \* this->getDen();

Rational R1(Num1 + Num2, cd);

R1.GCD(R1);

return R1;

}

Rational Rational::operator - (const Rational &tempRational) {

int cd = this->getDen() \* tempRational.getDen();

int Num1 = this->getNum() \* tempRational.getDen();

int Num2 = tempRational.getNum() \* this->getDen();

Rational R1(Num1 - Num2, cd);

R1.GCD(R1);

return R1;

}

Rational Rational::operator \* (const Rational &tempRational) {

int cdr = this->getDen() \* tempRational.getDen();

int Num5 = this->getNum() \* tempRational.getNum();

Rational R1(Num5, cdr);

R1.GCD(R1);

return R1;

}

Rational Rational::operator / (const Rational &tempRational) {

int cdd = this->getDen() \* tempRational.getNum();

int Num6 = this-> getNum() \* tempRational.getNum();

Rational R1(Num6, cdd);

R1.GCD(R1);

return R1;

}

Rational & Rational::operator=(const Rational &tempRational) {

this->numerator = tempRational.getNum();

this->denominator = tempRational.getDen();

return \*this;

}

//"Is more than" operator

bool Rational::operator>(Rational &tempRational) {

bool flag = false;

if (this -> CalculateFloat() > tempRational.CalculateFloat()) {

flag = true;

}

return flag;

}

//"Is less than" operator

bool Rational::operator<(Rational &tempRational) {

bool flag = false;

if (this -> CalculateFloat() < tempRational.CalculateFloat()) {

flag = true;

}

return flag;

}

//"Is equal to" operator

bool Rational::operator==(const Rational &tempRational) {

bool flag = false;

if(tempRational.getNum() == this-> getNum() & tempRational.getDen() == this-> getDen()) {

flag = true;

}

return flag;

}

ostream &operator<<(ostream &output, const Rational &tempRational) {

output << "(" << tempRational.getNum() << "/" << tempRational.getDen() << ") ";

return output;

}///end operator

istream &operator>>(istream &input, Rational &tempRational) {

cout << "Entra el valor de la base: " << endl;

input >> tempRational.numerator;

cout << "Entra el valor de la altura: " << endl;

input >> tempRational.denominator;

return input;

}

//Constructor, Destructor, Getters and Setters

Rational::Rational( int numerator, int denominator): numerator(numerator), denominator(denominator)

{}

Rational::~Rational() {

//cout << "Rational object destroyed.\n";

}

//Getters and Setters

int Rational::getNum() const {

return numerator;

}

int Rational::getDen() const {

return denominator;

}

void Rational::setNum(int a) {

numerator = a;

}

void Rational::setDen(int b) {

denominator = b;

}

//Math

Rational Rational::GCD(Rational g1) {

int temp = 1, a = g1.getNum() , b = g1.getDen();

if(a < 0) {

a \*= 1;

}

else if(b < 0) {

b \*= -1;

}

temp = a % b;

while(temp < 0 || temp > 0)

{

a = b;

b = temp;

temp = (a%b);

}

return Rational (g1.getNum()/b, g1.getDen()/b);

}

void Rational::Add(Rational a1) {

int cd = getDen() \* a1.getDen();

int Num1 = getNum() \* a1.getDen();

int Num2 = a1.getNum() \* getDen();

cout << "SUM: (" << getNum() << " / " << getDen() << ") + (" << a1.getNum() << " / " << a1.getDen() << ") = (" << Num1 + Num2 << " / " << cd << ")"<< endl;

}

void Rational::Subtract(Rational s1) {

int cds = getDen()\* s1.getDen();

int Num3 = getNum() \* s1.getDen();

int Num4 = s1.getNum() \* getDen();

cout << "SUBTRACTION: (" << getNum() << " / " << getDen() << ") - (" << s1.getNum() << " / " << s1.getDen() << ") = (" << Num3 - Num4 << " / " << cds << ")"<< endl;

}

void Rational::Multiply(Rational m1) {

int cdr = getDen()\* m1.getDen();

int Num5 = getNum() \* m1.getNum();

cout << "MULTIPLICATION: (" << getNum() << " / " << getDen() << ") \* (" << m1.getNum() << " / " << m1.getDen() << ") = (" << Num5 << " / " << cdr << ")"<< endl;

}

void Rational::Divide(Rational d1) {

int cdd = getDen()\* d1.getNum();

int Num6 = getNum() \* d1.getDen();

cout << "DIVISION: (" << getNum() << " / " << getDen() << ") / (" << d1.getNum() << " / " << d1.getDen() << ") = (" << Num6 << " / " << cdd << ")"<< endl;

}

//Display

void Rational::DisplayFract() const {

cout <<"DISPLAY FRACTION: " << this->getNum() << " / " << this->getDen() << endl;

}

void Rational::DisplayFloat() {

cout <<"DISPLAY DECIMAL OF FRACTION: " << this->getNum() << " / " << this->getDen() <<" = " << this->CalculateFloat() << endl;

}

//Copy Constructor

Rational::Rational( const Rational &aRational) {

setNum( aRational.getNum() );

setDen( aRational.getDen() );

//cout << "Copied fraction using the copy constructor." << endl;

}

// main.cpp

#include <iostream>

using namespace std ;

#include "Rational.h"

int main() {

Rational F1, F2, F4;

F1.setNum(1);

F1.setDen(2);

F2.setNum(1);

F2.setDen(4);

//Operaciones usando funciones.

cout << "WITH FUNCTIONS:" << endl;

//Math

F1.Add(F2);

F1.Subtract(F2);

F1.Multiply(F2);

F1.Divide(F2);

//Display

F1.DisplayFract();

F1.DisplayFloat();

//Copy Constructor

Rational F3(F2);

cout << "\n--COPIED FRACTION--" << endl;

F3.DisplayFract();

cout << endl;

//Uso de operadores.

cout << "WITH OPERATORS:" << endl;

cout << "SUM: \n" << F1 << "+ " << F2;

F3 = F1 + F2;

cout <<"= " << F3 << endl;

cout << "SUBCTRACTION: \n" << F1 << "- " << F2;

F3 = F1 - F2;

cout <<"= " << F3 << endl;

cout << "MUTIPLICATION: \n" << F1 << "\* " << F2;

F3 = F1 \* F2;

cout <<"= " << F3 << endl;

cout << "DIVISION:\n" << F1 << "/ " << F2;

F3 = F1 / F2;

cout <<"= " << F3 << endl;

return 0;

}

1. Numdays

// Numdays.h

#ifndef NUMDAYS\_H\_

#define NUMDAYS\_H\_

#include <iostream>

using namespace std;

class Numdays {

private:

float hours;

float numdays;

public:

//Constructor and Destructor

Numdays(float hours = 8, float numdays = 1);

~Numdays();

//Accesors and Mutators

void setHours(float h);

void setNumdays(float n);

float getHours() const;

float getNumdays() const;

//Copy Constructor

Numdays(const Numdays &N);

//Logical Operators

Numdays &operator = (const Numdays &);

bool operator > (const Numdays &tempNum);

bool operator < (const Numdays &tempNum);

//Assign Operator

bool operator ==(const Numdays &);

//Math Operators

Numdays operator + (const Numdays &tempNum);// addition operator

Numdays operator - (const Numdays &tempNum);// subtraction operator

Numdays operator \* (const Numdays &tempNum);// multiplication operator

Numdays operator / (const Numdays &tempNum);// division operator

// Unary ++ and -- Operators

//Prefix

Numdays operator ++ ();

Numdays operator -- ();

//Postfix

Numdays operator ++ (int);

Numdays operator -- (int);

//Stream Operators

friend ostream &operator << (ostream &, const Numdays &tempNum);

friend istream &operator >> (std::istream &, Numdays &);

};

#endif /\* Numdays\_h \*/

// Numdays.cpp

// Numdays Class

#include "Numdays.h"

#include <cmath>

#include <iostream>

//Constructor, Destructor, Accessors and Mutators

//Constructor and Destructor

Numdays::Numdays( float hours, float numdays): hours(hours), numdays(numdays)

{}

Numdays::~Numdays() {

//cout <<"Object has been destroyed." << endl;

}

//Accesors and Mutators

void Numdays::setHours(float h) {

this->hours = h;

this->numdays = this->hours/8;

}

void Numdays::setNumdays(float d) {

this->numdays = d;

this->hours = this->numdays \* 8;

}

float Numdays::getHours() const{

return this->hours;

}

float Numdays::getNumdays() const {

return this->numdays;

}

//Copy Constructor

Numdays::Numdays (const Numdays &n) {

setNumdays(n.getNumdays());

}

//Overloaded Operators

//Logical Operators

bool Numdays::operator==(const Numdays &tempNum) {

bool flag = false;

if(tempNum.getNumdays() == this->getNumdays()) {

flag = true;

}

return flag;

}

bool Numdays::operator>(const Numdays &tempNum) {

bool flag = false;

if(tempNum.getNumdays() > this->getNumdays()) {

flag = true;

}

return flag;

}

bool Numdays::operator<(const Numdays &tempNum) {

bool flag = false;

if(tempNum.getNumdays() < this->getNumdays()) {

flag = true;

}

return flag;

}

//Assign Operator

Numdays & Numdays::operator=(const Numdays &tempNum) {

this->numdays = tempNum.getNumdays();

this->hours = tempNum.getHours();

return \*this;

}

//Math Operators

Numdays Numdays::operator + (const Numdays &tempNum) {

float temp\_hours = this->getHours() + tempNum.getHours();

float temp\_days = this->getNumdays() + tempNum.getNumdays();

return Numdays (temp\_hours, temp\_days);

}

Numdays Numdays::operator - (const Numdays &tempNum) {

float temp\_hours = abs(this->getHours() - tempNum.getHours());

float temp\_days = abs(this->getNumdays() - tempNum.getNumdays());

return Numdays (temp\_hours, temp\_days);

}

Numdays Numdays::operator \* (const Numdays &tempNum) {

float temp\_hours = this->getHours() \* tempNum.getHours();

float temp\_days = this->getNumdays() \* tempNum.getNumdays();

return Numdays (temp\_hours, temp\_days);

}

Numdays Numdays::operator / (const Numdays &tempNum) {

float temp\_hours = this->getHours() / tempNum.getHours();

float temp\_days = this->getNumdays() / tempNum.getNumdays();

return Numdays (temp\_hours, temp\_days);

}

// Unary ++ and -- Operators

//Prefix

Numdays Numdays::operator ++ () {

++hours;

this->setHours(hours);

return \*this;

}

Numdays Numdays::operator -- () {

--hours;

this->setHours(hours);

return \*this;

}

//Postfix

Numdays Numdays::operator++(int) {

Numdays temp(hours, numdays) ;

hours++;

this->setHours(hours);

return temp;

}

Numdays Numdays::operator--(int) {

Numdays temp(hours, numdays) ;

hours--;

this->setHours(hours);

return temp;

}

//Stream Operators

ostream &operator<<(ostream &output, const Numdays &tempNum) {

output << "Number of hours: " << tempNum.getHours() << endl;

output << "Number of days: " << tempNum.getNumdays() << endl;

return output;

}

istream &operator>>(istream &input, Numdays &tempNum) {

cout << "Enter the number of hours worked this shift: " << endl;

input >> tempNum.hours;

//tempNum.numdays = this->getNumdays();

input >> tempNum.numdays;

return input;

}

// main.cpp

#include <iostream>

#include "Numdays.h"

using namespace std;

int main() {

//Code demonstrating the Numdays Class.

Numdays N, M, L;

cout << "Setting 35 hours to the object N." << endl;

N.setHours(35);

cout << "Hours: " << N.getHours() << endl;

cout << "Days: " << N.getNumdays() << endl;

cout << "Adding objects N and M. The latter has default values hours = 8, numdays = 1." << endl;

L = N + M;

cout << L << endl;

cout << "Subtracting objects N and M." << endl;

L = N - M;

cout << "Hours: " << L.getHours() << endl;

cout << "Days: " << L.getNumdays() << endl;

cout << "Prefix operators" << endl;

cout << "--L: " << endl;

--L;

cout << L << endl;

cout << "++L: " << endl;

++L;

cout << L << endl;

cout << "Postfix operators" << endl;

cout << "L--: " << endl;

L--;

cout << L << endl;

cout << "L++: " << endl;

L++;

cout << L << endl;

return 0;

}

1. Carpet Calculator

// FeetInches.h

#ifndef FEETINCHES\_H

#define FEETINCHES\_H

#include <iostream>

using namespace std;

// The FeetInches class holds distances or measurements // expressed in feet and inches.

class FeetInches {

private:

// To hold a number of feet // To hold a number of inches

int feet;

int inches;

void simplify(); // Defined in FeetInches.cpp

public:

// Constructor

FeetInches(int f = 0, int i = 0) {

feet = f;

inches = i;

simplify();

}

// Mutator functions

void setFeet(int f) {

feet = f;

}

void setInches(int i) {

inches = i;

simplify();

}

// Accessor functions

int getFeet() const {

return feet;

}

int getInches() const {

return inches;

}

float FeetInchesToFloat ();

// Overloaded operator functions

FeetInches operator + (const FeetInches &);

FeetInches operator - (const FeetInches &);

FeetInches operator \* (const FeetInches &);

bool operator > (const FeetInches &);

bool operator < (const FeetInches &);

FeetInches &operator = (const FeetInches &);

bool operator == (const FeetInches &);

// Friends

friend ostream &operator << (ostream &, const FeetInches &temp);

friend istream &operator >> (std::istream &, FeetInches &);

};

#endif /\* FeetInches\_h \*/

// Implementation file for the FeetInches class

#include <cstdlib> // Needed for abs()

#include "FeetInches.h"

void FeetInches::simplify() {

if (inches >= 12) {

feet += (inches / 12);

inches = inches % 12;

}

else if (inches < 0) {

feet -= ((abs(inches) / 12) + 1);

inches = 12 - (abs(inches) % 12);

}

}

float FeetInches::FeetInchesToFloat () {

float feet\_float = (float)this->getFeet();

float float\_inches = (float)this->getInches()/12;

float float\_feet = feet\_float + float\_inches;

return float\_feet;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Overloaded binary + and - operators. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FeetInches FeetInches::operator + (const FeetInches &right) {

FeetInches temp;

temp.inches = inches + right.inches;

temp.feet = feet + right.feet;

temp.simplify();

return temp;

}

FeetInches FeetInches::operator - (const FeetInches &right) {

FeetInches temp;

temp.inches = inches - right.inches;

temp.feet = feet - right.feet;

temp.simplify();

return temp;

}

FeetInches FeetInches::operator \* ( const FeetInches &temp) {

int total\_inches = (feet \* 12) + inches;

int temp\_total\_inches = (temp.feet \* 12) + temp.inches;

int mult = total\_inches \* temp\_total\_inches;

FeetInches temp1;

temp1.inches = mult;

temp1.feet = 0;

temp1.simplify();

return temp1;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Overloaded >, < and == operators. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool FeetInches::operator>(const FeetInches &tempFeetInches) {

bool flag = false;

if( this-> feet > tempFeetInches.feet) {

flag = true;

} else if (feet == tempFeetInches.feet) {

if(inches > tempFeetInches.inches) {

flag = true;

}

}

return flag;

}

bool FeetInches::operator<(const FeetInches &tempFeetInches) {

bool flag = false;

if( this-> feet < tempFeetInches.feet) {

flag = true;

} else if (feet == tempFeetInches.feet) {

if(inches < tempFeetInches.inches) {

flag = true;

}

}

return flag;

}

FeetInches & FeetInches::operator=(const FeetInches &tempFeetInches) {

this->feet = tempFeetInches.getFeet();

this->inches = tempFeetInches.getInches();

simplify();

return \*this;

}

bool FeetInches::operator==(const FeetInches &tempFeetInches) {

bool flag = false;

if( this-> feet == tempFeetInches.feet) {

flag = true;

}

return flag;

}

//Stream Operators

ostream &operator << (ostream &output, const FeetInches &temp) {

output << temp.getFeet() << " ft. ";

output << temp.getInches() << " inches" << endl;

return output;

}

istream &operator >> (istream &input, FeetInches &temp) {

cout << "Enter the number of feet: " << endl;

input >> temp.feet;

cout << "Enter the number of inches: " << endl;

input >> temp.inches;

temp.simplify();

return input;

}

// RoomDimension.hpp

// Carpet Calculator

#ifndef ROOMDIMENSION\_H

#define ROOMDIMENSION\_H

#include "FeetInches.h"

class RoomDimension {

private:

FeetInches width;

FeetInches length;

FeetInches area;

public:

RoomDimension(FeetInches width = 1, FeetInches length = 1, FeetInches area = 1);

RoomDimension();

~RoomDimension();

FeetInches getWidth() const;

FeetInches getLength() const;

FeetInches getArea() const;

void setWidth(int w);

void setLength(int l);

void setArea();

// Overloaded operator functions

RoomDimension operator + (const RoomDimension &);

RoomDimension operator - (const RoomDimension &);

bool operator > ( RoomDimension &);

bool operator < ( RoomDimension &);

RoomDimension &operator = (const RoomDimension &);

bool operator == ( RoomDimension &);

// Friends

friend ostream &operator << (ostream &, const RoomDimension &temp);

friend istream &operator >> (std::istream &, RoomDimension &);

};

#endif /\* RoomDimension\_h \*/

//

// RoomDimension.cpp

// Carpet Calculator

//

// Created by Rafael Pagan on 4/27/17.

// Copyright © 2017 Rafael Pagan. All rights reserved.

//

#include "FeetInches.h"

#include "RoomDimension.h"

RoomDimension::RoomDimension(FeetInches width, FeetInches length, FeetInches area): width(width), length(length), area(area) {}

RoomDimension::RoomDimension() {

width = 1;

length = 1;

area = 1;

}

RoomDimension::~RoomDimension() {}

FeetInches RoomDimension::getWidth() const {

return this->width;

}

FeetInches RoomDimension::getLength() const {

return this->length;

}

FeetInches RoomDimension::getArea() const {

return this->area;

}

void RoomDimension::setWidth(int w) {

this->width = w;

}

void RoomDimension::setLength(int l) {

length = l;

}

void RoomDimension::setArea() {

area = width \* length;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Overloaded binary + and - operators. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RoomDimension RoomDimension::operator + (const RoomDimension &temp) {

FeetInches temp\_width = width + temp.width;

FeetInches temp\_length = length + temp.length;

FeetInches temp\_area = temp\_width \* temp\_length;

return RoomDimension (temp\_width, temp\_length, temp\_area);

}

RoomDimension RoomDimension::operator - (const RoomDimension &temp) {

FeetInches temp\_width = width - temp.width;

FeetInches temp\_length = length - temp.length;

FeetInches temp\_area = temp\_width \* temp\_length;

return RoomDimension (temp\_width, temp\_length, temp\_area);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Overloaded >, < and == operators. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool RoomDimension::operator > ( RoomDimension &temp) {

this->setArea();

temp.setArea();

bool flag = false;

if( this-> area > temp.area) {

flag = true;

}

return flag;

}

bool RoomDimension::operator < ( RoomDimension &temp) {

this->setArea();

temp.setArea();

bool flag = false;

if( this-> area < temp.area) {

flag = true;

}

return flag;

}

RoomDimension & RoomDimension::operator = (const RoomDimension &temp) {

this->width = temp.getWidth();

this->length = temp.getLength();

return \*this;

}

bool RoomDimension::operator == ( RoomDimension &temp) {

this->setArea();

temp.setArea();

bool flag = false;

if( this-> area == temp.area) {

flag = true;

}

return flag;

}

//Stream Operators

ostream &operator << (ostream &output, const RoomDimension &temp) {

output << "Width: " << temp.getWidth() << endl;

output << "Length: " << temp.getLength() << endl;

return output;

}

istream &operator >> (istream &input, RoomDimension &temp) {

cout << "Width: " << endl;

input >> temp.width;

cout << "Length: " << endl;

input >> temp.length;

temp.setArea();

return input;

}

//

// RoomCarpet.h

// Carpet Calculator

//

// Created by Rafael Pagan on 4/27/17.

// Copyright © 2017 Rafael Pagan. All rights reserved.

// Next, you should create a RoomCarpet class that has a RoomDimension object as an attribute. It should also have an attribute for the cost of the carpet per square foot. The RoomCarpet class should have a member function that returns the total cost of the carpet.

#ifndef ROOMCARPET\_H

#define ROOMCARPET\_H

#include "RoomDimension.h"

class RoomCarpet {

private:

RoomDimension room;

float cost;

public:

//RoomCarpet();

RoomCarpet(RoomDimension room, float cost = 1);

~RoomCarpet();

RoomDimension getRoom() const;

float getCost() const;

void setCost();

void setRoom(RoomDimension R);

// Friends

friend ostream &operator << (ostream &, const RoomCarpet &temp);

friend istream &operator >> (std::istream &, RoomCarpet &);

};

#endif /\* RoomCarpet\_h \*/

//

// RoomCarpet.cpp

// Carpet Calculator

//

// Created by Rafael Pagan on 4/27/17.

// Copyright © 2017 Rafael Pagan. All rights reserved.

//

#include "RoomCarpet.h"

#include "RoomDimension.h"

RoomCarpet::RoomCarpet(RoomDimension room, float cost): room(room), cost(cost) {}

RoomCarpet::~RoomCarpet() {}

RoomDimension RoomCarpet::getRoom() const {

return this->room;

}

float RoomCarpet::getCost() const {

return this->cost;

}

void RoomCarpet::setCost() {

FeetInches room\_width = room.getWidth();

FeetInches room\_length = room.getLength();

float w = room\_width.FeetInchesToFloat();

float l = room\_length.FeetInchesToFloat();

this->cost = w \* l \* this->cost;

}

void RoomCarpet::setRoom(RoomDimension R) {

cin >> R;

}

//Stream Operators

ostream &operator << (ostream &output, const RoomCarpet &temp) {

output << "Room dimensions: \n" << temp.getRoom() << endl;

output << "Carpet Cost: \n$" << temp.getCost() << endl;

return output;

}

istream &operator >> (istream &input, RoomCarpet &temp) {

cout << "Please enter the room dimensions. " << endl;

input >> temp.room;

cout << "Please enter the cost per square foot. " << endl;

input >> temp.cost;

temp.setCost();

return input;

}

// main.cpp

#include <iostream>

#include "RoomCarpet.h"

int main() {

//RoomCarpet Carpet;

RoomDimension R(1,1,1);

RoomCarpet C(R,1);

cout << "--CARPET CALCULATOR--" << endl;

//Get information from the user.

cin >> C;

//Display carpet cost and room values.

cout << C;

return 0;

}