//

// main.cpp

// lab6a

// completed

// Created by Jeff on 9/19/16.

// Copyright © 2016 Jeff zhang. All rights reserved.

//

#include <iostream>

#include <string>

#include <sstream>

#include "FeetInches.h"

#include <cstdlib>

using namespace std;

class FeetInches;

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

istream &operator >> (istream&, FeetInches&);

void FeetInches::lowTerm() {

if (inches >= 12) {

feet += (inches / 12);

inches = inches % 12;

}

else if (inches < 0) {

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

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

}

}

int main() {

string input;

int feet,inches,feet1,inches1;

do{

cout << "enter the first fetiches(0,0 to exit) - \n\t feet = ";

cin >> feet;

cout << "\t inches = ";

cin >> inches;

FeetInches fi0(feet,inches);

if(feet == 0 || inches ==0)

return 0;

else

cout << "you have created a "<< fi0.getFeet()<< " \'"

<< fi0.getInches() << "\" FeetInches instance.\n";

cout << "enter a second fetiches - \n\t feet = ";

cin >> feet1;

cout << "\t inches = ";

cin >> inches1;

FeetInches fi1(feet1,inches1);

cout << "you have created second as: "<< fi1.getFeet()<< " \'"

<< fi1.getInches() << "\" FeetInches instance.\n";

/\*

cout << "creating a feetinches(\"3\\\'4\"): ";

FeetInches fil("3'4\"");

cout << "\n the test driver has created: ";

cout <<fil<<endl;

\*/

FeetInches fi2(5,6);

cout << "\n Copy constructed the 3rd FeetInches:( " << fi2 << " )";

cout << "\n --->Overloaded Addition Operator (+): "<<endl;

cout << "\n ( "<<feet <<" " << " feet," << inches<< " inches )+ "

<< " ( "<<feet1 <<" " << " feet," << inches1<< " inches ) = ( " << fi0 + fi1 << ' ' << " )";

cout << "\n --->Overloaded Addition Operator (-): "<<endl;

cout << "\n ( "<<feet <<" " << " feet," << inches<< " inches )- "

<< " ( "<<feet1 <<" " << " feet," << inches1<< " inches ) = ( " << fi0 - fi1 << ' ' << " )";

cout << "\n --> Overloaded Greater Than Operator (>): Is " << fi0

<< " > " << fi1 << " ? "<<"\n" << ((fi0>fi1)? "True":"False") << endl;

cout << "\n --> Overloaded Post ++ than -- Operators "<<endl;

cout << "\n start first ( "<< fi0 << " ) " <<endl;

fi0++;

cout << " Post ++ ( " << fi0 << ' ' <<" )" <<endl;

fi0--;

cout <<" Post -- (" << fi0<< ' ' <<")"<<endl;

cout << "\n --> Overloaded Pre ++ than -- Operators "<<endl;

cout << "\n start second ( "<< fi1 << " ) " <<endl;

++fi1;

cout << " Post ++ ( " << fi1 << ' ' <<" )" <<endl;

--fi1;

cout <<" Post -- (" << fi1<< ' ' <<")"<<endl;

}while(feet !=0 || inches !=0);

return 0;

}

#ifndef LANDTRACT\_H

#define LANDTRACT\_H

#include <iostream>

#include <cmath>

using namespace std;

class FeetInches;

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

istream &operator>> (istream&, FeetInches&);

class FeetInches {

private:

int feet; // To hold a number of feet

int inches; // To hold a number of inches

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

public:

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

{ feet = f; inches = i; lowTerm(); }

FeetInches(string s);

void setFeet(int f) { feet = f; }

void setInches(int i) { inches = i; lowTerm(); }

int getFeet() const { return feet; }

int getInches() const { return inches; }

// Overloaded operator functions

FeetInches operator + (const FeetInches &); // Overloaded +

FeetInches operator - (const FeetInches &); // Overloaded -

FeetInches& operator ++ (); // Prefix ++

FeetInches operator ++ (int); // Postfix ++

FeetInches& operator -- (); // Prefix --

FeetInches operator -- (int); // Postfix --

bool operator > (const FeetInches &); // Overloaded >

bool operator < (const FeetInches &); // Overloaded <

bool operator >= (const FeetInches &); // Overloaded >=

bool operator <= (const FeetInches &); // Overloaded <=

bool operator == (const FeetInches &); // Overloaded ==

bool operator != (const FeetInches &); // Overloaded !=

// Conversion functions

operator double();

operator int();

// Friends

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

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

};

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

FeetInches temp;

temp.inches = inches + right.inches;

temp.feet = feet + right.feet;

temp.lowTerm();

return temp;

}

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

FeetInches temp;

temp.inches = inches - right.inches;

temp.feet = feet - right.feet;

temp.lowTerm();

return temp;

}

FeetInches& FeetInches::operator ++ ()

{ ++inches; lowTerm(); return \*this; }

FeetInches FeetInches::operator ++ (int)

{ FeetInches temp(feet, inches);

inches++; lowTerm(); return temp; }

FeetInches& FeetInches::operator -- ()

{ --inches; lowTerm(); } //return \*this; }

FeetInches FeetInches::operator -- (int)

{ FeetInches temp(feet, inches);

inches--; lowTerm(); return temp; }

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

bool status;

if (feet > right.feet)

status = true;

else if (feet == right.feet && inches > right.inches)

status = true;

else

status = false;

return status;

}

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

bool status;

if (feet < right.feet)

status = true;

else if (feet == right.feet && inches < right.inches)

status = true;

else

status = false;

return status;

}

bool FeetInches::operator >= (const FeetInches &right){

bool status;

if ((\*this > right) || (\*this == right))

status = true;

else status = false;

return status;

}

bool FeetInches::operator <= (const FeetInches &right) {

bool status;

if ((\*this < right) || (\*this == right))

status = true;

else status = false;

return status;

}

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

bool status;

if (feet == right.feet && inches == right.inches)

status = true;

else

status = false;

return status;

}

bool FeetInches::operator != (const FeetInches &right) {

bool status;

if (\*this == right) status = false;

else status = true;

return status;

}

ostream &operator<<(ostream &strm, const FeetInches &obj) {

strm << obj.feet << " feet, " << obj.inches << " inches";

return strm;

}

istream &operator >> (istream &strm, FeetInches &obj) {

cout << "Feet: ";

strm >> obj.feet;

cout << "Inches: ";

strm >> obj.inches;

obj.lowTerm();

return strm;

}

FeetInches::operator double() {

double temp = feet;

temp += (inches / 12.0);

return temp;

}

FeetInches::operator int() { return feet; }

FeetInches::FeetInches(string s) {

stringstream ss(s); // use this form 3'9"

ss >> feet;

char c = ss.peek();

if(ss && c=='\'')

ss.get();

ss >> inches;

};

class LandTract {

private:

FeetInches width, length;

public:

LandTract() {width=1; length=1;};

LandTract(FeetInches w, FeetInches l) {width=w; length=l;};

FeetInches getWidth() { return width; }

FeetInches getLength() { return length; }

void setWidth( FeetInches w ) { width = w; }

void setLength( FeetInches l ) { length = l; }

FeetInches getArea() {

double wid = width;

double len = length;

double area = wid\*len;

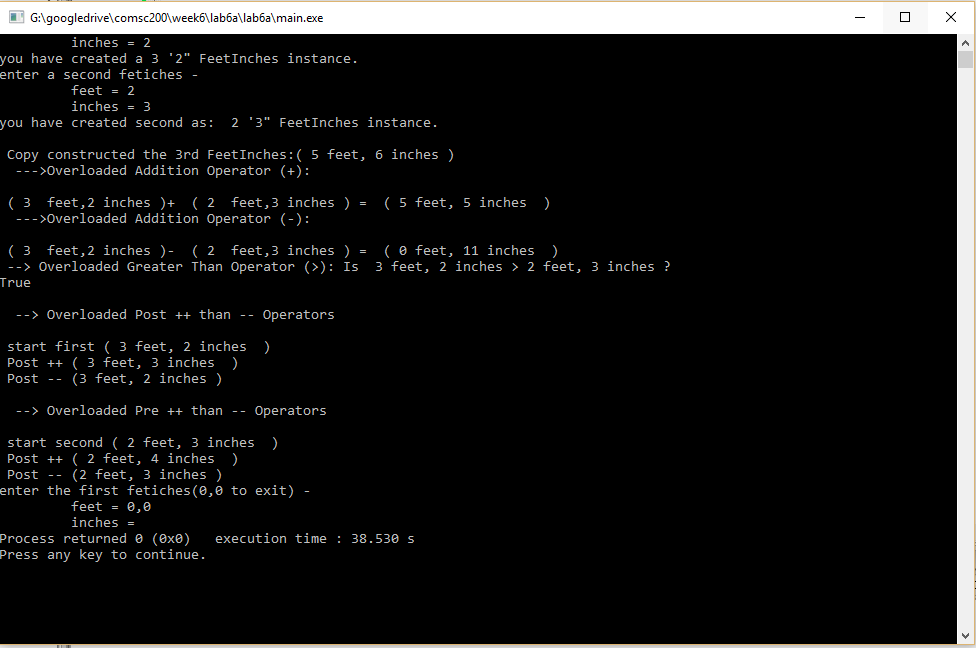
FeetInches temp(area);

return temp;

};

};

#endif



// Chapter 14, Programming Challenge 12: LandTract Class

//comsc 200

//completed

//boli zhang

#include <iostream>

#include <sstream>

#include "LandTract.h"

using namespace std;

void menu();

void getFeetInches(int & feet, int & inches);

int main()

{

bool stay = true; // Boolean to stay in this menu

string choice, word, text; // User input for choices and input

int feet, inches;

FeetInches width, length;

LandTract myProperty;

while(stay) { // Main menu while starts

menu();

cin >> choice; // Take in user choice from menu

cin.ignore();

if(choice.size() == 1) {

char ch = choice[0];

switch(ch) { // menu switch start

case 'n':

case 'N': // new LandTract

{

cout << " Enter Tract Width in Feet/Inches: \n";

getFeetInches(feet, inches);

width.setFeet(feet);

width.setInches(inches);

myProperty.setWidth(width);

cout << " Enter Tract Length in Feet/Inches: \n";

getFeetInches(feet, inches);

length.setFeet(feet);

length.setInches(inches);

myProperty.setLength(length);

cout << endl;

break;

}

case 'd':

case 'D': // Display this LandTract

{

// Display the area.

cout << " the width is " << myProperty.getWidth()

<< "\n the length is " << myProperty.getLength()

<< "\n the area is " << myProperty.getArea()

<< "\n\n";

break;

}

case 'q':

case 'Q': // User can press q or Q to exit

stay = false;

break;

default: // Invalid! Tells user to try again

cout << "\nInvalid command!\nTry again!\n\n";

}

}

}

cout << "\nProgram exit!";

}

// A LandTract test with a fool proof menu

void menu() {

cout << "\_\_\_\_\_\_\_ LandTract Class Test Menu \_\_\_\_\_\_\_\_\_\_\n"

<< " n - Create a new LandTract\n"

<< " d - Display the LandTract attributes\n"

<< " q - Quit\n"

<< " Enter your choice:";

}

void getFeetInches(int & feet, int & inches) {

string input;

cout << " Enter Feet: ";

while(true) {

getline(cin, input);

stringstream ss(input);

ss >> feet;

if (ss.fail())

cout << "Feet is an integer! Try again: ";

else break;

}

cout << " Enter Inches: ";

while(true) {

getline(cin, input);

stringstream ss(input);

ss >> inches;

if (ss.fail())

cout << "Inches is an integer! Try again: ";

else break;

}

}

#ifndef LANDTRACT\_H

#define LANDTRACT\_H

//comsc 200

//completed

// boli zhang

#include <iostream>

#include <cstdlib>>

using namespace std;

class FeetInches;

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

istream &operator>> (istream&, FeetInches&);

class FeetInches {

private:

int feet; // To hold a number of feet

int inches; // To hold a number of inches

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

public:

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

{ feet = f; inches = i; lowTerm(); }

FeetInches(string s);

void setFeet(int f) { feet = f; }

void setInches(int i) { inches = i; lowTerm(); }

int getFeet() const { return feet; }

int getInches() const { return inches; }

// Overloaded operator functions

FeetInches operator + (const FeetInches &); // Overloaded +

FeetInches operator - (const FeetInches &); // Overloaded -

FeetInches& operator ++ (); // Prefix ++

FeetInches operator ++ (int); // Postfix ++

FeetInches& operator -- (); // Prefix --

FeetInches operator -- (int); // Postfix --

bool operator > (const FeetInches &); // Overloaded >

bool operator < (const FeetInches &); // Overloaded <

bool operator >= (const FeetInches &); // Overloaded >=

bool operator <= (const FeetInches &); // Overloaded <=

bool operator == (const FeetInches &); // Overloaded ==

bool operator != (const FeetInches &); // Overloaded !=

// Conversion functions

operator double();

operator int();

// Friends

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

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

};

void FeetInches::lowTerm() {

if (inches >= 12) {

feet += (inches / 12);

inches = inches % 12;

}

else if (inches < 0) {

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

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

}

}

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

FeetInches temp;

temp.inches = inches + right.inches;

temp.feet = feet + right.feet;

temp.lowTerm();

return temp;

}

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

FeetInches temp;

temp.inches = inches - right.inches;

temp.feet = feet - right.feet;

temp.lowTerm();

return temp;

}

FeetInches& FeetInches::operator ++ ()

{ ++inches; lowTerm(); return \*this; }

FeetInches FeetInches::operator ++ (int)

{ FeetInches temp(feet, inches);

inches++; lowTerm(); return temp; }

FeetInches& FeetInches::operator -- ()

{ --inches; lowTerm(); } //return \*this; }

FeetInches FeetInches::operator -- (int)

{ FeetInches temp(feet, inches);

inches--; lowTerm(); return temp; }

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

bool status;

if (feet > right.feet)

status = true;

else if (feet == right.feet && inches > right.inches)

status = true;

else

status = false;

return status;

}

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

bool status;

if (feet < right.feet)

status = true;

else if (feet == right.feet && inches < right.inches)

status = true;

else

status = false;

return status;

}

bool FeetInches::operator >= (const FeetInches &right){

bool status;

if ((\*this > right) || (\*this == right))

status = true;

else status = false;

return status;

}

bool FeetInches::operator <= (const FeetInches &right) {

bool status;

if ((\*this < right) || (\*this == right))

status = true;

else status = false;

return status;

}

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

bool status;

if (feet == right.feet && inches == right.inches)

status = true;

else

status = false;

return status;

}

bool FeetInches::operator != (const FeetInches &right) {

bool status;

if (\*this == right) status = false;

else status = true;

return status;

}

ostream &operator<<(ostream &strm, const FeetInches &obj) {

strm << obj.feet << " feet, " << obj.inches << " inches";

return strm;

}

istream &operator >> (istream &strm, FeetInches &obj) {

cout << "Feet: ";

strm >> obj.feet;

cout << "Inches: ";

strm >> obj.inches;

obj.lowTerm();

return strm;

}

FeetInches::operator double() {

double temp = feet;

temp += (inches / 12.0);

return temp;

}

FeetInches::operator int() { return feet; }

FeetInches::FeetInches(string s) {

stringstream ss(s); // use this form 3'9"

ss >> feet;

char c = ss.peek();

if(ss && c=='\'')

ss.get();

ss >> inches;

};

class LandTract {

private:

FeetInches width, length;

public:

LandTract() {width=1; length=1;};

LandTract(FeetInches w, FeetInches l) {width=w; length=l;};

FeetInches getWidth() { return width; }

FeetInches getLength() { return length; }

void setWidth( FeetInches w ) { width = w; }

void setLength( FeetInches l ) { length = l; }

FeetInches getArea() {

double wid = width;

double len = length;

double area = wid\*len;

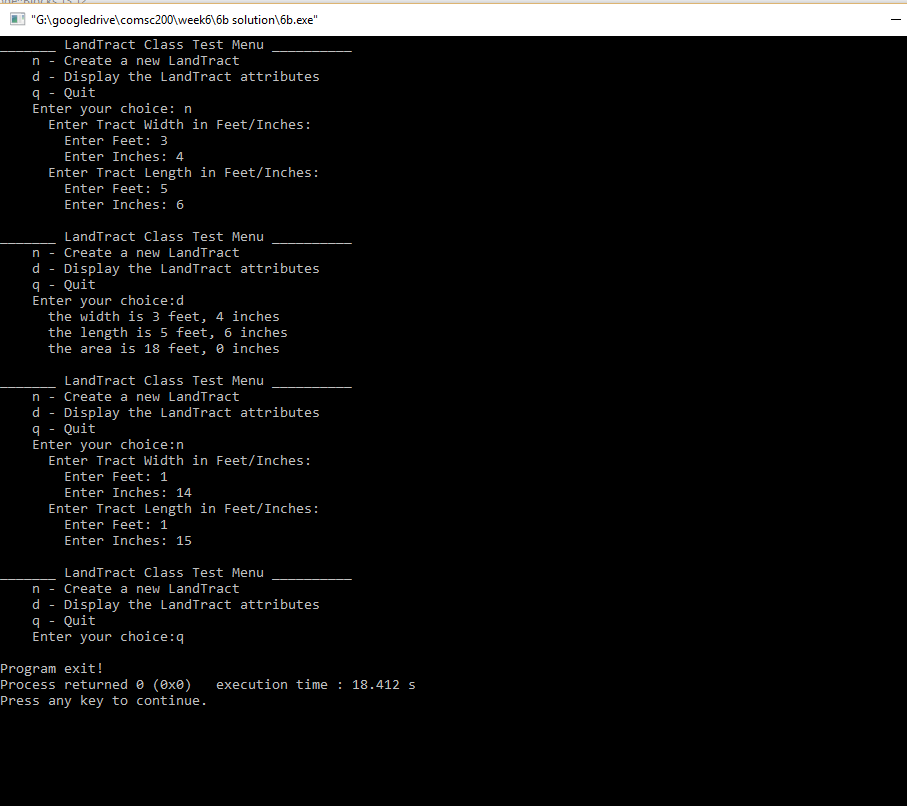
FeetInches temp(area);

return temp;

};

};

#endif



//

// main.cpp

// lab2

//

// Created by Jeff on 9/19/16.

// Copyright © 2016 Jeff zhang. All rights reserved.

//

#include <iostream>

using namespace std;

class NumDays{

private:

double hours;

double days;

public:

NumDays(double h=0){

hours=h;

days=h/8;

}

void setHours(double h){

hours = h;

days = h/8;

}

double getHours()const {

return hours;

}

void setDays(double d){

days = d ;

hours = d\*8;

}

double getDays() const {

return days;

}

NumDays operator+(NumDays&right){

NumDays temp;

temp.setHours(hours + right.getHours());

return temp;

}

NumDays operator++(){//prefix

hours++;

days = hours/8;

return \*this;

}

NumDays operator--(){//prefix

hours--;

days = hours/8;

return \*this;

}

NumDays operator--(int){//prefix

NumDays temp(hours);

hours--;

days = hours/8;

return temp;

}

NumDays operator++(int){//prefix

NumDays temp(hours);

hours++;

days = hours/8;

return temp;

}

};

int main() {

NumDays one(25),two(15);

NumDays three,four;

cout << "one(25): "<< one.getDays()<<endl;

cout << "two(25): "<< two.getDays()<<endl;

three = one + two;

cout<< "\n three = one + two is "<< three.getDays();

++three;

cout << "\n prefix increment ++three is " << three.getDays();

three++;

cout << "\n prefix increment three++ is " << three.getDays();

--three;

cout << "\n prefix increment --three is " << three.getDays();

three--;

cout << "\n prefix increment three-- is " << three.getDays();

}

