// comsc 200

// boli zhang

// completed

// frac.cpp

// Created by Jeff on 9/7/16.

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

//

#include <iostream>

#include <string>

#include "Frac.h"

using namespace std;

int main() {

//mathod 1

Frac x(3,4);

Frac y(2,3);

cout << " created x as "; //argument constructors.

x.show();

cout << "\n created y as ";

y.show();

cout <<endl;

cout << "\nShow created x and y using returnObj() method: "

<< "\nExecuted Frac x(2, 4); Created x: " << x.returnObj()

<< "\nExecuted Frac x(2, 3); Created y: " << y.returnObj();

Frac z = x; //assignment (=) operator.

cout << "\n Overload assignment operator = \n Frac z=x; created z as " << z << endl; //assignment (=) operator.

/\* //mathod 2

cout<< "created x :" << x.toString()

<< "\n created y :" << y.toString() << endl;

\*/

//mathod 3 operator

// cout<< " x assigned as y :" << x;

// << "\n created y :" << y << endl;

Frac zz(x);

cout << "\n copy constructor: \n Frac zz(x); created z: "<< zz << endl; //copy constructor: Frac x(y) that accept a Frac object as an argument.

cout << "\n Overloaded assignment operator + \n " << x <<" + "<< y <<" = "<< x+y << endl;

// cout << "\n\nx += y compound assignments return self:\n"

// << x << " += " << y;

// cout<< "; x is " << (x+= y) ;

cout << "\n Overloaded assignment operator - \n " << x <<" - "<< y <<" = "<< x-y << endl;

cout << "\n Overloaded assignment operator \* \n " << x <<" \* "<< y <<" = "<< x\*y << endl;

cout << "\n Overloaded assignment operator / \n " << x <<" / "<< y <<" = "<< x/y << endl;

cout << "\n Overloaded assignment operator > \n " << y << " the predication " << x << " > "<<y <<" is " << ((x>y)? "True":"False") << endl;

cout << "\n Overloaded assignment operator < \n " << y << " the predication " << x << " < "<<y <<" is " << ((x<y)? "True":"False") << endl;

cout << "\n\nx += y compound assignments return self:\n"

<< x << " += " << y

<< "; x is " << (x += y) ;

// <<<<----- 2="" async="" problems="" ------="">>>>>

cout << "\n\nx + y; operator+ returns an l-value obj:\n"

<< x << " + " << y << " = " << x + y;

cout << "\n\nTesting Overloaded Postfix Operators:";

cout << "\ny: " << y ;

y++;

cout << "\ny++: " << y ;

y--;

cout << "\ny--: " << y ;

cout << "\n\nTesting Overloaded Prefix Operators:";

cout << "\ny: " << y ;

++y;

cout << "\n++y: " << y ;

--y;

cout << "\n--y: " << y ;

cout << "\n\nMore Async issues:";

cout << "\nTesting Overloaded Postfix Operators:";

cout << "\ny: " << y << " y++: " << y++;

cout << "\ny: " << y << " y--: " << y--;

cout << "\ny: " << y ;

cout << "\n\nTesting Overloaded Prefix Operators:";

cout << "\ny: " << y << " ++y: " << ++y;

cout << "\ny: " << y << " --y: " << --y;

cout << "\ny: " << y ;

cout << "\n\nTesting Operator Concatenation:"

<< "\nx + y + x + y: " << (x + y + x + y)<<"\n";

}

// comsc 200

// boli zhang

// completed

// Frac.h

// lab1

//

// Created by Jeff on 9/7/16.

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

//

#ifndef Frac\_h

#define Frac\_h

#include <string>

class Frac;

std::ostream &operator << (std::ostream &, const Frac &);

class Frac{

private:

long num;

long den;

long gcd(long a, long b){

long c = a%b;

while(c!=0){

a = b;

b = c;

c = a%b;

}

return b;

};

void lowTerm(){

long g = gcd(num,den);

num = num/g;

den = den/g;

};

public:

Frac(){

num=0;

den=1;

};

Frac(long n){

num=n;

den=1;

};

Frac(long n,long d){

num=n;

den=d;

lowTerm();

};

Frac(const Frac &x){ //copy constructor: Frac x(y) that accept a Frac object as an argument.

num = x.num;

den = x.den;

};

void set(long n, long d){

num=n;

den=d;

lowTerm();

};

//post pre increment

Frac operator++ (int){Frac copy = \*this;num+=1;

return copy;};

Frac operator++ (){num++; return \*this;};

Frac operator-- (int){Frac copy = \*this;num-=1;

return copy;};

Frac operator-- (){num--; return \*this;};

Frac operator + (const Frac &x){

Frac temp;

temp.num = num\*x.den + x.num\*den;

temp.den = den\*x.den;

return temp;

};

Frac& operator+= (const Frac &x){

Frac temp;

temp.num = num;//\*x.den; // - x.num\*den;

temp.den = den;//\*x.den;

return temp;

};

Frac& operator - (const Frac &right){

Frac left;

left.num = num\*right.den - right.num\*den;

left.den = den\*right.den;

return left;

};

Frac& operator \* (const Frac &right){

Frac left;

left.num = num\*right.num;

left.den = den\*right.den;

return left;

};

Frac& operator / (const Frac &right){

Frac left;

left.num = num\*right.den;

left.den = den\*right.num;

return left;

};

std::string returnObj(){

std::string s = std::to\_string(num) + "/" + std::to\_string(den);

return s;

}

void show(){

std::cout<<num << "/" <<den ;

}

std::string toString(){

return std::to\_string(num) + "/" + std::to\_string(den);

}

// friend ostream operator << function definition

friend std::ostream &operator << (std::ostream &strm, const Frac &right){

strm << right.num << "/" << right.den;

return strm;

}

bool operator > (const Frac &x){

if(num\*x.den > den\*x.num)

return true;

else return false;

}

bool operator < (const Frac &x){

if(num\*x.den < den\*x.num)

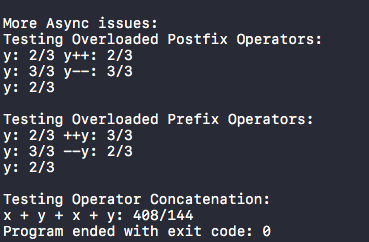
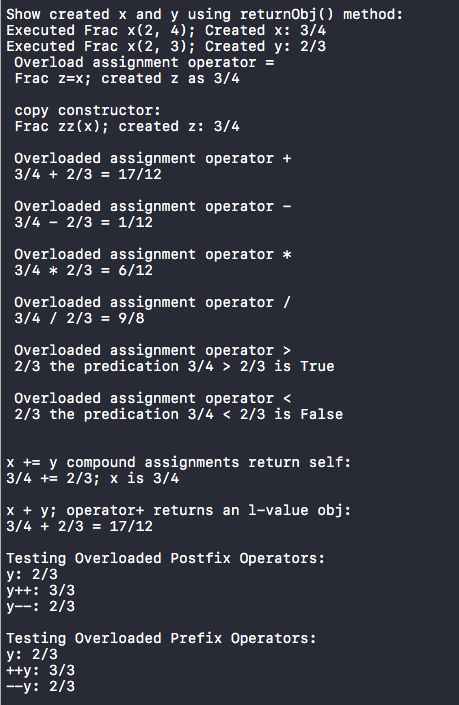
return true;

else return false;

}

};

#endif /\* Frac\_h \*/



//comsc 200

//completed

//lab5b

//overloadsubscript

//name : boli zhang

#include <iostream>

#include <cstdlib> // exit()

using namespace std;

const int LIMIT = 100; //array size

////////////////////////////////////////////////////////////////

class safearay

{

private:

int arr[LIMIT];

public:

int& operator [](int n) //note: return by reference

{

if( n< 0 || n>=LIMIT )

{ cout << "\nIndex out of bounds"; exit(1); }

return arr[n];

}

};

int main()

{

safearay sa1;

cout << "Enter lower and upper index: " << endl;

int j,temp1;

cin >> j >> temp1;

//cout << "upper bounds: " << endl;

//cin >> temp1;

for( int j=0; j<LIMIT; j++){ //insert elements

sa1[j] = j\*10;

}//\*left\* side of equal sign

for(j; j<temp1+1; j++) //display elements

{

// int temp = sa1[j]; //\*right\* side of equal sign

cout << "Element " << j << " is " << sa1[j] << endl;

}

cout << "To modify which cell?: " << endl;

int Modnum;

cin >> Modnum;

safearay sa2;

cout << "Enter lower and upper index: " << endl;

int k,temp2;

cin >>k >> temp2;

for( int k=0; k<LIMIT; k++){ //insert elements

sa1[k] = k\*10;

}//\*left\* side of equal sign

for(k; k<Modnum; k++) //display elements

{

int temp = sa1[k]; //\*right\* side of equal sign

cout << "Element " << k << " is " << temp << endl;

}

cout << "Element " << Modnum << " is " << Modnum << endl;

for(Modnum; Modnum<temp2; Modnum++) //display elements

{

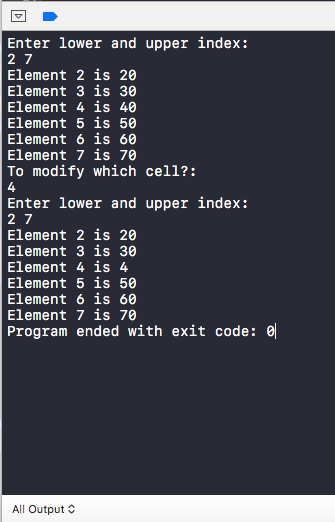
int temp = sa1[Modnum+1]; //\*right\* side of equal sign

cout << "Element " << Modnum+1 << " is " << temp << endl;

}

return 0;

}



// comsc 200

// boli zhang

// completed

// friend.cpp

//

// Created by Jeff on 9/7/16.

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

//

#include <iostream>

#include <cstdlib> // labs()

using namespace std;

class frac; // Forward Declaration <--- necessary for using frac before it is defined

// Function Prototypes for Overloaded Stream Operators

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

// Define the GoodFriend class which has a Friend outputfrac()

class GoodFriend {

public:

void showFrac(frac &);

};

class BestFriend {

public:

void showFrac(frac &);

};

class frac {

long num, den;

public:

frac() { num = 0; den = 1; }

frac(long n, long d) { num = n; den = d; }

frac(const frac &f) {num = f.num; den = f.den;}

// 2. Friend function ---> GoodFriend::outputfrac() prototype here

friend void GoodFriend::showFrac(frac &);

friend class BestFriend;

friend ostream &operator << ( ostream &strm, const frac &right) {

strm << right.num << "/" << right.den;

return strm;

}

};

// 3. Friend class to frac - method definition here

// GoodFriend showFrac() defintion

void GoodFriend::showFrac(frac &f) { cout << " " << f.num << "/" << f.den << " ";}

void BestFriend::showFrac(frac &f) { cout << " " << f.num << "/" << f.den << " ";}

int main()

{

// testing friendship

frac f(3,8);

BestFriend bf;

bf.showFrac(f);

GoodFriend gf;

gf.showFrac(f);

cout << f ;

return 0;

}

#include <iostream>

#include <cstdlib> // labs()

using namespace std;

class frac; // Forward Declaration <--- necessary for using frac before it is defined

// Function Prototypes for Overloaded Stream Operators

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

// Define the GoodFriend class which has a Friend outputfrac()

class GoodFriend {

public:

void showFrac(frac &);

};

class BestFriend {

public:

void showFrac(frac &);

};

class Friend {

public:

void showFrac(frac &);

};

class best{

public:

void displayFrac(frac &);

};

class frac {

long num, den;

public:

frac() { num = 0; den = 1; }

frac(long n, long d) { num = n; den = d; }

frac(const frac &f) {num = f.num; den = f.den;}

// 2. Friend function ---> GoodFriend::outputfrac() prototype here

friend void GoodFriend::showFrac(frac &);

friend class BestFriend;

friend class Friend;

friend class best;

friend ostream &operator << ( ostream &strm, const frac &right) {

strm << right.num << "/" << right.den;

return strm;

}

};

// 3. Friend class to frac - method definition here

// GoodFriend showFrac() defintion

void GoodFriend::showFrac(frac &f) { cout<< "GoodFriend good.outputFrac(f) is" << " " << f.num << "/" << f.den << " "<<endl;}

void BestFriend::showFrac(frac &f) { cout <<"BestFriend Best.outputFrac(f) is"<< " " << f.num << "/" << f.den << " "<<endl;}

void Friend::showFrac(frac &f) {cout<< "Friend showFrac(f) is" << " " << f.num << "/" << f.den << " "<<endl; }

void best::displayFrac(frac &f) {cout<< "Buddy of BestFriend best.displayFrac(f) is" << " " << f.num << "/" << f.den << " "<<endl; }

int main()

{

// testing friendship

frac f(3,8);

cout << "frac f(3,8) is " << frac(3,8) <<endl;

Friend fd;

fd.showFrac(f);

BestFriend bf;

bf.showFrac(f);

GoodFriend gf;

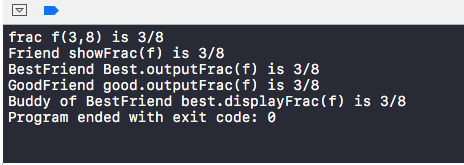
gf.showFrac(f);

best bb;

bb.displayFrac(f);

return 0;

}



//

// main.cpp

// 5e funcTemplate

// completed

// Created by Jeff on 9/14/16.

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

//

#include<iostream>

#include<string>

using namespace std;

// absoluteValue template

template <typename T>

inline T const& absoluteValue(T const& a){

if(a >0)

return a;

else return a\*(-1);

}

// Template for the minimum function

template <typename T>

inline T const& minimum (T const& a, T const& b)

{

return a < b ? b:a;

}

// Template for the maximum function

template <typename T>

inline T const& maximum (T const& a, T const& b)

{

return a > b ? b:a;

}

// The main function

int main()

{

// MinMax and Absolute Test with int arguments.

int num1 = -5;

int num2 = 3;

cout << "minimum of " << num1 << " and " << num2

<< " is: " << minimum(num1, num2) << endl;

cout << "maximum of " << num1 << " and " << num2

<< " is: " << maximum(num1, num2) << endl;

cout << "Absolute value of " << num1

<< " is: " << absoluteValue(num1) << endl;

cout << "Absolute value of " << num2

<< " is: " << absoluteValue(num2) << endl;

// MinMax and Absolute Test with double arguments.

double num3 = -5.5;

double num4 = 3.5;

cout << "minimum of " << num3 << " and " << num4

<< " is: " << minimum(num3, num4) << endl;

cout << "maximum of " << num3 << " and " << num4

<< " is: " << maximum(num3, num4) << endl;

cout << "Absolute value of " << num3

<< " is: " << absoluteValue(num3) << endl;

cout << "Absolute value of " << num4

<< " is: " << absoluteValue(num4) << endl;

// MinMax Test with string arguments.

string s1 = "hello";

string s2 = "hi";

cout << "minimum of " << s1 << " and " << s2

<< " is: " << minimum(s1, s2) << endl;

cout << "maximum of " << s1 << " and " << s2

<< " is: " << maximum(s1, s2) << endl;

return 0;

}

