**C++ Part I (INFO1-CE9264) Fall 2014 – Homework 8**

Clement Chan

**Question 2 – Rational Numbers**

**Header File**

#include<iostream>

#include<cmath>

#include<cstdlib>

using namespace std;

class Rational{

private:

int numerator;

int denominator;

int Wholenumber;

public:

Rational();

Rational(int);

Rational(int,int);

void setnum(int);

void setdenom(int);

void setwhole(int);

int getnum() const;

int getdenom() const;

int getwhole() const;

void normalize();

//Operator Overloading

friend ostream& operator<<(ostream& outputstream, const Rational& rationalnumbers);

friend istream& operator>>(istream& inputstream, const Rational& rationalnumbers);

//Numerical Operator Overloading

friend const Rational operator +(const Rational& rhs, const Rational& lhs);

friend const Rational operator -(const Rational& rhs, const Rational& lhs);

friend const Rational operator \*(const Rational& rhs, const Rational& lhs);

friend const Rational operator /(const Rational& rhs, const Rational& lhs);

};

bool operator ==(const Rational& rhs, const Rational& lhs);

bool operator >(const Rational& rhs, const Rational& lhs);

bool operator <(const Rational& rhs, const Rational& lhs);

bool operator >=(const Rational& rhs, const Rational& lhs);

bool operator <=(const Rational& rhs, const Rational& lhs);

**Rational.cpp**

#include"Rational.h"

//Constructors

Rational::Rational(){

Wholenumber = 0;

}

Rational::Rational(int a){

Wholenumber = a / 1;

}

Rational::Rational(int a, int b){

numerator = a;

denominator = b;

}

//Functions to return private numbers

void Rational::setnum(int a){

numerator = a;

}

void Rational::setdenom(int a){

denominator = a;

}

void Rational::setwhole(int a){

Wholenumber = a;

}

int Rational::getnum() const{

return numerator;

}

int Rational::getdenom() const{

return denominator;

}

int Rational::getwhole() const{

return Wholenumber;

}

//Operator Overloading with outstream and instream function

ostream& operator << (ostream& outputstream, const Rational& rationalnumbers){

int numer = abs(rationalnumbers.numerator);

int denom = abs(rationalnumbers.denominator);

int whole = abs(rationalnumbers.Wholenumber);

if(rationalnumbers.numerator < 0 && rationalnumbers.denominator > 0){

outputstream << "-";

}

else if(rationalnumbers.numerator > 0 && rationalnumbers.denominator < 0){

outputstream << "-";

}

else if (rationalnumbers.numerator < 0 && rationalnumbers.denominator < 0){

outputstream << "";

}

if (rationalnumbers.denominator == 1){

outputstream << numer;

}

else if(rationalnumbers.denominator == rationalnumbers.numerator){

outputstream << 1;

}

else if(rationalnumbers.denominator == 0){

cout<<"Denominator cannot be zero." << endl;

}

else if (rationalnumbers.numerator != 0 && rationalnumbers.denominator != 0){

outputstream << numer << "/" << denom;

}

return outputstream;

}

istream& operator >> (istream& inputstream, Rational& rationalnumbers){

int numer;

int denom;

char divide;

cout << "Enter The Rational Number you want: " << endl;

inputstream >> numer;

if(numer == 0){

rationalnumbers.setnum(0);

}

inputstream >> divide;

if(divide != '/' || divide == ' '){

rationalnumbers.setnum(numer);

}

inputstream >> denom;

rationalnumbers.setnum(numer);

rationalnumbers.setdenom(denom);

return inputstream;

}

//Normalizing denominators and numerators

void Rational::normalize(){

int GCD;

int maxsize;

if(abs(numerator) > abs(denominator)){

maxsize = abs(numerator);

}

else if(abs(numerator) < abs(denominator)){

maxsize = abs(denominator);

}

for(int i=1; i<=maxsize; i++){

if(abs(numerator) % i == 0 && abs(denominator) % i == 0){

GCD = i;

}

}

numerator /=GCD;

denominator /= GCD;

}

//Numerical Operator Overloading

const Rational operator +(const Rational& rhs, const Rational& lhs){

int total\_num;

int total\_den;

total\_num = rhs.numerator\*lhs.denominator + lhs.numerator\*rhs.denominator;

total\_den = rhs.denominator \* lhs.denominator;

return Rational(total\_num, total\_den);

}

const Rational operator -(const Rational& rhs, const Rational& lhs){

int total\_num;

int total\_den;

total\_num = rhs.numerator\*lhs.denominator - lhs.numerator\*rhs.denominator;

total\_den = rhs.denominator \* lhs.denominator;

return Rational(total\_num, total\_den);

}

const Rational operator \*(const Rational& rhs, const Rational& lhs){

int total\_num;

int total\_den;

total\_num = rhs.numerator \* lhs.numerator;

total\_den = rhs.denominator \* lhs.denominator;

return Rational(total\_num, total\_den);

}

const Rational operator /(const Rational& rhs, const Rational& lhs){

int total\_num;

int total\_den;

total\_num = rhs.numerator\*lhs.denominator;

total\_den = rhs.denominator \* lhs.numerator;

return Rational(total\_num, total\_den);

}

bool operator ==(const Rational& rhs, const Rational& lhs){

return ((rhs.getnum() \* lhs.getdenom()) == (lhs.getnum() \* rhs.getdenom()));

}

bool operator >(const Rational& rhs, const Rational& lhs){

return (rhs.getnum() \* lhs.getdenom() > lhs.getnum() \* rhs.getdenom());

}

bool operator <(const Rational& rhs, const Rational& lhs){

return (rhs.getnum() \* lhs.getdenom() < lhs.getnum() \* rhs.getdenom());

}

bool operator <=(const Rational& rhs, const Rational& lhs){

return (rhs.getnum() \* lhs.getdenom() <= lhs.getnum() \* rhs.getdenom());

}

bool operator >=(const Rational& rhs, const Rational& lhs){

return (rhs.getnum() \* lhs.getdenom() >= lhs.getnum() \* rhs.getdenom());

}

int main(){

Rational R\_input, R1(2,3), R2;

cin >> R\_input;

if(R\_input.getdenom()==0){

cout<<"The denominator is zero.."<<endl;

exit(0);

}

R2 = R\_input + R1;

R2.normalize();

cout << "The Result of Input Rational Number + Default Rational Number is: " <<

endl;

cout << R2 << endl;

R2 = R\_input - R1;

R2.normalize();

cout << "The Result of Input Rational Number - Default Rational Number is: " <<

endl;

cout << R2 << endl;

R2 = R\_input \* R1;

R2.normalize();

cout << "The Result of Input Rational Number \* Default Rational Number is: " <<

endl;

cout << R2 << endl;

R2 = R\_input / R1;

R2.normalize();

cout << "The Result of Input Rational Number / Default Rational Number is: " <<

endl;

cout << R2 << endl;

if(R\_input == R1)

{cout<<"The entered values are the same!"<<endl;}

else if(R\_input > R1)

{cout<<"The entered values is greater than fixed!"<<endl;}

else if(R\_input < R1)

{cout<<"The entered values is less than fixed!"<<endl;}

else if(R\_input >= R1)

{cout<<"The entered values is greater and equal to fixed!"<<endl;}

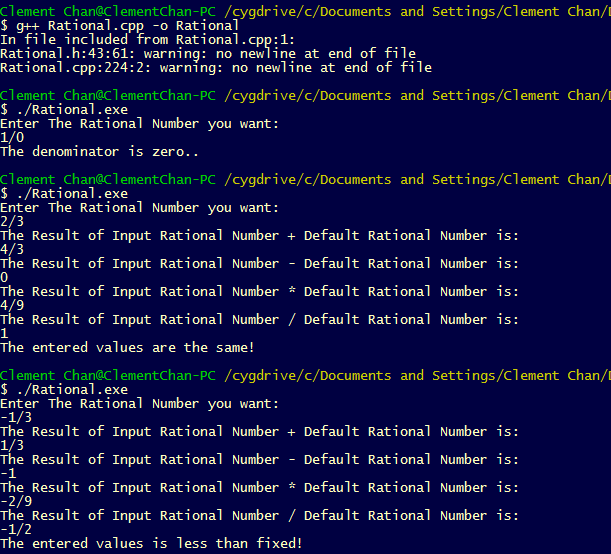
else if(R\_input <= R1)

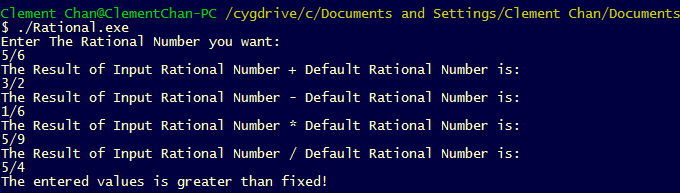
{cout<<"The entered values is less than and equal fixed!"<<endl;}

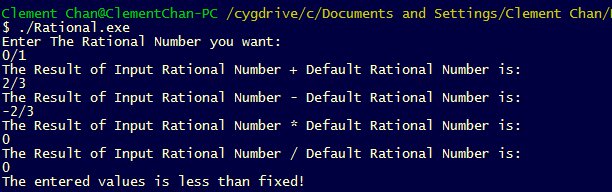
return 0;

}

**Output:**

****

****

****

**Question 5 – Dot Products**

**Dot Product.h**

#include <iostream>

#include <cmath>

#include <cstdlib>

using namespace std;

class Vector2D{

private:

int \*\*xycoord;

int dot\_product;

int max\_size;

public:

Vector2D();

Vector2D(int,int);

void init\_vector(int);

void set\_x();

void set\_y();

int get\_x(int) const;

int get\_y(int) const;

int get\_maxsize() const;

int get\_dot\_product() const;

//Operator Overloading

friend const Vector2D operator \*(const Vector2D &rhs, const Vector2D &lhs);

//Destructor

~Vector2D();

};

**Dot Products.cpp**

#include "Dot\_Products.h"

Vector2D::Vector2D(){

dot\_product = 0;

max\_size = 0;

}

Vector2D::Vector2D(int a, int b){

dot\_product = (a + b);

}

void Vector2D::init\_vector(int a){

max\_size = a;

int max\_col = 2;

xycoord = new int\*[max\_size];

for(int i = 0; i < max\_size; i++){

xycoord[i] = new int[max\_col];

}

}

void Vector2D::set\_x(){

for(int i = 0; i < max\_size; i++){

xycoord[i][1] = i;

cout<<xycoord[i][1]<<endl;

}

}

void Vector2D::set\_y(){

for(int i = 0; i < max\_size; i++){

xycoord[i][2] = i+1;

cout<<xycoord[i][2]<<endl;

}

}

int Vector2D::get\_x(int a)const{

return xycoord[a][1];

}

int Vector2D::get\_y(int a)const{

return xycoord[a][2];

}

int Vector2D::get\_maxsize()const{

return max\_size;

}

int Vector2D::get\_dot\_product()const{

return dot\_product;

}

const Vector2D operator \*(const Vector2D &rhs, const Vector2D &lhs){

int sum\_x = 0;

int sum\_temp\_x = 0;

int sum\_y = 0;

int sum\_temp\_y = 0;

for(int i=0; i < rhs.get\_maxsize(); i++){

sum\_temp\_x = rhs.get\_x(i) \* lhs.get\_x(i);

sum\_temp\_y = rhs.get\_y(i) \* lhs.get\_y(i);

sum\_x += sum\_temp\_x;

sum\_y += sum\_temp\_y;

}

return Vector2D(sum\_x, sum\_y);

}

bool operator ==(const Vector2D &rhs, const Vector2D &lhs){

return (rhs.get\_maxsize() == lhs.get\_maxsize());

}

Vector2D::~Vector2D(){

cout<<"Calling Destructor..."<<endl;

delete[] xycoord;

}

//Main section to test the program

int main(){

int Dot\_Product;

Vector2D V2D\_1, V2D\_2, V2D\_3;

V2D\_1.init\_vector(5);

V2D\_2.init\_vector(5);

if(V2D\_1 == V2D\_2){

cout<<"X coordinates of first set is: "<<endl;

V2D\_1.set\_x();

cout<<"Y coordinates of first set is: "<<endl;

V2D\_1.set\_y();

cout<<"X coordinates of second set is: "<<endl;

V2D\_2.set\_x();

cout<<"Y coordinates of second set is: "<<endl;

V2D\_2.set\_y();

V2D\_3 = V2D\_1 \* V2D\_2;

cout<<"The dot product of the two sets is: " << V2D\_3.get\_dot\_product()<<endl;

}

else{

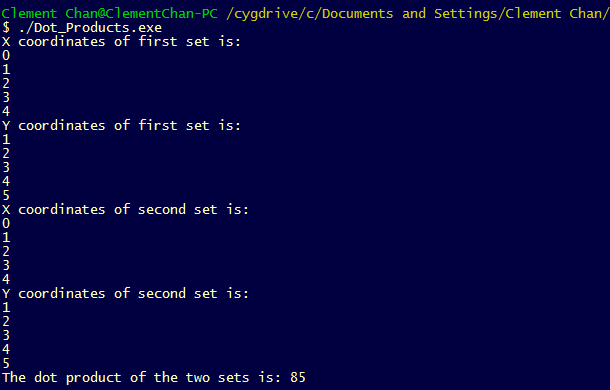
cout<<"Dimension mismatch..." <<endl;

}

return 0;

}

**Output:**

****

**Question 6 – My Integer**

**MyInteger.h**

#include<iostream>

#include<cmath>

using namespace std;

class MyInteger{

private:

int \*x;

int units;

int number;

public:

void set\_number(int);

void set\_units(int);

int get\_number();

int get\_units();

int get\_x(int);

int operator[](int);

};

**MyInteger.cpp**

#include "myinteger.h"

void MyInteger::set\_number(int a){

number= a;

}

int MyInteger::get\_number(){

return number;

}

void MyInteger::set\_units(int a){

int unit\_temp=0;

int divided;

int i = 9; //An integer can take up to 1 billion unit.

x = new int[i];

while(i >= 0){

divided = a / int(pow(10.0, i));

x[i] = divided;

if(divided > 1 && divided < 10){

if(i > unit\_temp){

unit\_temp = i;

}

}

a -= divided \* int(pow(10.0,i));

i--;

}

units = unit\_temp;

}

int MyInteger::get\_units(){

return units;

}

int MyInteger::get\_x(int a){

return x[a];

}

int MyInteger::operator[](int y){

if(y > units){

return -1;

}

else{

return x[y];

}

};

int main(){

MyInteger MI;

MI.set\_number(418);

MI.set\_units(418);

//Show the numbers entered and the number of units needed

cout<<"The number entered is: "<<MI.get\_number()<<endl;

cout<<"The number of rows (index) in the column (including 0) is: " << MI.get\_units()<<endl;

//Getting the number using [] operator

cout<<"The first unit using the [] operator is: " << MI[0]<<endl;

cout<<"The second unit using the [] operator is: " << MI[1]<<endl;

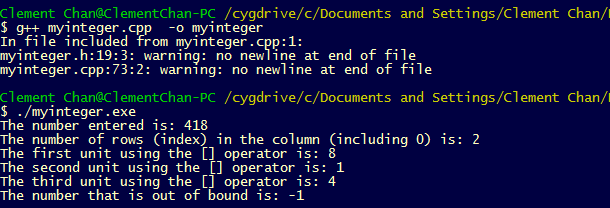
cout<<"The third unit using the [] operator is: " << MI[2]<<endl;

cout<<"The number that is out of bound is: " << MI[3]<<endl;

return 0;

}

**Output:**

****