**Universidad Nacional Autónoma de Honduras Tarea2-II-Parcial**

**Asignatura: Programación II**

**Profesor: David Motiño**

**Alumno: Oseas Enmanuel Mejia Calona**

**N° Cuenta: 20141030181**

**Fecha Entrega: 11/04/2018**

**Herencia Clase Matriz con Simétrica**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <iomanip>

using namespace std;

class matriz{

friend matriz\* operator-(const matriz&,const matriz&);

friend matriz\* operator\*(float, const matriz&);

private:

protected:

int n, m;

float \*\*elementos;

public:

matriz();

matriz(int, int);

~matriz();

virtual void imprimir() const;

matriz\* operator+(const matriz&)const;

matriz\* operator\*(const matriz&)const;

bool esSimetrica();

virtual float traza() const;

matriz\* transpuesta();

};

class simetrica:public matriz{

friend simetrica\* operator-(const simetrica&,const simetrica&);

friend simetrica\* operator\*(float, const simetrica&);

private:

public:

simetrica();

simetrica(int);

//~simetrica();

virtual void imprimir()const;

simetrica\* operator+(const simetrica&)const;

simetrica\* operator\*(const simetrica&)const;

virtual float traza()const;

simetrica\* transpuesta()const;

};

int main(int argc, char\*\* argv) {

srand(time(0));

simetrica s1(3), s2(3);

matriz &refs1 = s1;

matriz &refs2 = s2;

cout<<"Taza s1: "<<refs1.traza()<<endl;

matriz \*m3 = refs1+refs2;

refs1.imprimir();

refs2.imprimir();

return 0;

}

//##############################Inicio de funciones de la clase Matriz.##############################

matriz::matriz(){

n=3;

m=3;

elementos = new float\*[n];

for(int i=0;i<n;i++)

elementos[i] = new float[m];

elementos[0][0] = 1;

elementos[0][1] = 0;

elementos[0][2] = 0;

elementos[1][0] = 0;

elementos[1][1] = 1;

elementos[1][2] = 0;

elementos[2][0] = 0;

elementos[2][1] = 0;

elementos[2][2] = 1;

}

matriz::matriz(int nF, int nC){

n=nF;

m=nC;

elementos = new float\*[n];

for(int i=0;i<n;i++)

elementos[i] = new float[m];

for(int i=0;i<n;i++){

for(int j=0;j<m;j++)

elementos[i][j]=rand()%100;

}

}

matriz::~matriz(){

for(int i=0;i<n;i++)

delete[] elementos[i];

delete[] elementos;

}

void matriz::imprimir() const{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

for(int i=0;i<n;i++){

for(int j=0;j<m;j++)

cout<<setw(5)<<elementos[i][j];

cout<<endl;

}

}

matriz\* matriz::operator+(const matriz& b)const{

cout<<"ESTAMOS EN LA SUMA DE LA CLASE MATRIZ\n\n";

if(n==b.n && m==b.m){

matriz\* c = new matriz(n, m);

for(int i=0;i<n;i++){

for(int j=0;j<m;j++)

c->elementos[i][j]=elementos[i][j] + b.elementos[i][j];

}

return c;

}else{

matriz\* c = new matriz();

return c;

}

}

matriz\* operator-(const matriz& a,const matriz& b){

cout<<"ESTAMOS EN LA RESTA DE LA CLASE MATRIZ\n\n";

if(a.n==b.n && a.m==b.m){

matriz\* c = new matriz(a.n, a.m);

for(int i=0;i<a.n;i++){

for(int j=0;j<a.m;j++)

c->elementos[i][j]=a.elementos[i][j] - b.elementos[i][j];

}

return c;

}else{

matriz\* c = new matriz();

return c;

}

}

matriz\* operator\*(float k, const matriz& b){

cout<<"ESTAMOS EN LA MULTIPLICACION DE MATRIZ\n\n";

matriz \*mul = new matriz(b.n, b.m);

for(int i=0;i<b.n;i++)

for(int j=0;j<b.m;j++)

mul->elementos[i][j]=k\*b.elementos[i][j];

return mul;

}

matriz\* matriz::operator\*(const matriz& b)const{

if(m==b.n){

matriz \*mul = new matriz(n, b.m);

for(int i=0;i<n;i++)

for(int j=0;j<b.m;j++)

mul->elementos[i][j]=0;

for(int i=0;i<n;i++)

for(int j=0;j<b.m;j++)

for(int k=0;k<b.m;k++)

mul->elementos[i][j]+=elementos[i][k]\*b.elementos[k][j];

return mul;

}else{

matriz \*mul = new matriz();

return mul;

}

}

bool matriz::esSimetrica(){

bool resp = false;

matriz\* a= this->transpuesta();

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(elementos[i][j]==(a->elementos[i][j])){

resp=true;

}else{

resp=false;

break;

}

}

}

return resp;

}

float matriz::traza() const{

float s=0;

for(int i=0;i<n;i++)

s+=elementos[i][i];

return s;

}

matriz\* matriz::transpuesta(){

matriz \*t = new matriz(n,m);

for(int i=0;i<m;i++){

for(int j=0;j<n;j++)

t->elementos[i][j]=elementos[j][i];

}

return t;

}

//##############################Fin de funciones de la clase Matriz.##############################

//##############################Inicio de funciones de la clase Simetrica.##############################

simetrica::simetrica(){

n=3;

elementos = new float\*[n];

for(int i=0;i<n;i++)

elementos[i] = new float[n];

elementos[0][0] = 1;

elementos[0][1] = 0;

elementos[0][2] = 0;

elementos[1][0] = 0;

elementos[1][1] = 1;

elementos[1][2] = 0;

elementos[2][0] = 0;

elementos[2][1] = 0;

elementos[2][2] = 1;

}

simetrica::simetrica(int dim){

n=dim;

elementos = new float\*[n];

for(int i=0;i<n;i++){

elementos[i]=new float[i+1];

for(int j=0;j<i+1;j++)

elementos[i][j] = rand()%10;

}

}

void simetrica::imprimir() const{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

for(int i=0;i<n;i++){

for(int j=0;j<i+1;j++)

cout<<setw(5)<<elementos[i][j];

for(int k=i+1;k<n;k++)

cout<<setw(5)<<elementos[k][i];

cout<<endl;

}

cout<<endl;

}

simetrica\* simetrica::operator+(const simetrica& s)const{

if(n==s.n && m==s.m){

simetrica\* c = new simetrica(n);

for(int i=0;i<n;i++){

for(int j=0;j<i+1;j++)

c->elementos[i][j]=elementos[i][j] + s.elementos[i][j];

}

return c;

}else{

simetrica\* c = new simetrica();

return c;

}

}

simetrica\* operator-(const simetrica&s1,const simetrica&s2){

if(s1.n==s2.n && s1.m==s2.m){

simetrica\* c = new simetrica(s1.n);

for(int i=0;i<s1.n;i++){

for(int j=0;j<i+1;j++)

c->elementos[i][j]=s1.elementos[i][j] - s2.elementos[i][j];

}

return c;

}else{

simetrica\* c = new simetrica();

return c;

}

}

simetrica\* operator\*(float k, const simetrica& s){

simetrica \*mul = new simetrica(s.n);

for(int i=0;i<s.n;i++)

for(int j=0;j<i+1;j++)

mul->elementos[i][j]=k\*s.elementos[i][j];

return mul;

}

simetrica\* simetrica::operator\*(const simetrica& s)const{

if(m==s.n){

simetrica \*mul = new simetrica(n);

for(int i=0;i<n;i++)

for(int j=0;j<s.n;j++)

mul->elementos[i][j]=0;

for(int i=0;i<n;i++)

for(int j=0;j<s.n;j++)

for(int k=0;k<s.n;k++)

mul->elementos[i][j]+=elementos[i][k]\*s.elementos[k][j];

return mul;

}else{

simetrica \*mul = new simetrica();

return mul;

}

}

float simetrica::traza()const{

float s=0;

for(int i=0;i<n;i++)

s+=elementos[i][i];

return s;

}

simetrica\* simetrica::transpuesta()const{

simetrica \*t = new simetrica(n);

for(int i=0;i<n;i++){

for(int j=0;j<i+1;j++)

t->elementos[i][j]=elementos[i][j];

}

return t;

}

//##############################Fin de funciones de la clase Simetrica.##############################

**Herencia Clase matriz con tridiagonal**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <iomanip>

using namespace std;

class matriz{

friend matriz\* operator-(const matriz&,const matriz&);

friend matriz\* operator\*(float, const matriz&);

private:

protected:

int n, m;

float \*\*elementos;

public:

matriz();

matriz(int, int);

~matriz();

virtual void imprimir() const;

matriz\* operator+(const matriz&)const;

matriz\* operator\*(const matriz&)const;

bool esSimetrica();

virtual float traza() const;

matriz\* transpuesta();

};

class tridiagonal:public matriz{

friend tridiagonal\* operator\*(float, const tridiagonal&);

private:

float\* diagprin;

float\* diaginf;

float\* diagsup;

public:

tridiagonal();

tridiagonal(int);

~tridiagonal();

virtual void imprimir()const;

tridiagonal\* operator+(const tridiagonal&)const;

tridiagonal\* operator-(const tridiagonal&)const;

tridiagonal\* operator\*(const tridiagonal&)const;

virtual float traza()const;

tridiagonal\* transpuesta()const;

};

int main(int argc, char\*\* argv) {

srand(time(0));

tridiagonal t1(3), t2(3);

matriz& reft1 = t1;

matriz& reft2 = t2;

matriz\* s= reft1 + reft2;

t1.imprimir();

t2.imprimir();

s->imprimir();

delete s;

return 0;

}

//##############################Inicio de funciones de la clase Matriz.##############################

matriz::matriz(){

n=3;

m=3;

elementos = new float\*[n];

for(int i=0;i<n;i++)

elementos[i] = new float[m];

elementos[0][0] = 1;

elementos[0][1] = 0;

elementos[0][2] = 0;

elementos[1][0] = 0;

elementos[1][1] = 1;

elementos[1][2] = 0;

elementos[2][0] = 0;

elementos[2][1] = 0;

elementos[2][2] = 1;

}

matriz::matriz(int nF, int nC){

n=nF;

m=nC;

elementos = new float\*[n];

for(int i=0;i<n;i++)

elementos[i] = new float[m];

for(int i=0;i<n;i++){

for(int j=0;j<m;j++)

elementos[i][j]=rand()%100;

}

}

matriz::~matriz(){

for(int i=0;i<n;i++)

delete[] elementos[i];

delete[] elementos;

}

void matriz::imprimir() const{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

for(int i=0;i<n;i++){

for(int j=0;j<m;j++)

cout<<setw(5)<<elementos[i][j];

cout<<endl;

}

}

matriz\* matriz::operator+(const matriz& b)const{

cout<<"ESTAMOS EN LA SUMA DE LA CLASE MATRIZ\n\n";

if(n==b.n && m==b.m){

matriz\* c = new matriz(n, m);

for(int i=0;i<n;i++){

for(int j=0;j<m;j++)

c->elementos[i][j]=elementos[i][j] + b.elementos[i][j];

}

return c;

}else{

matriz\* c = new matriz();

return c;

}

}

matriz\* operator-(const matriz& a,const matriz& b){

cout<<"ESTAMOS EN LA RESTA DE LA CLASE MATRIZ\n\n";

if(a.n==b.n && a.m==b.m){

matriz\* c = new matriz(a.n, a.m);

for(int i=0;i<a.n;i++){

for(int j=0;j<a.m;j++)

c->elementos[i][j]=a.elementos[i][j] - b.elementos[i][j];

}

return c;

}else{

matriz\* c = new matriz();

return c;

}

}

matriz\* operator\*(float k, const matriz& b){

cout<<"ESTAMOS EN LA MULTIPLICACION DE MATRIZ\n\n";

matriz \*mul = new matriz(b.n, b.m);

for(int i=0;i<b.n;i++)

for(int j=0;j<b.m;j++)

mul->elementos[i][j]=k\*b.elementos[i][j];

return mul;

}

matriz\* matriz::operator\*(const matriz& b)const{

if(m==b.n){

matriz \*mul = new matriz(n, b.m);

for(int i=0;i<n;i++)

for(int j=0;j<b.m;j++)

mul->elementos[i][j]=0;

for(int i=0;i<n;i++)

for(int j=0;j<b.m;j++)

for(int k=0;k<b.m;k++)

mul->elementos[i][j]+=elementos[i][k]\*b.elementos[k][j];

return mul;

}else{

matriz \*mul = new matriz();

return mul;

}

}

bool matriz::esSimetrica(){

bool resp = false;

matriz\* a= this->transpuesta();

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(elementos[i][j]==(a->elementos[i][j])){

resp=true;

}else{

resp=false;

break;

}

}

}

return resp;

}

float matriz::traza() const{

float s=0;

for(int i=0;i<n;i++)

s+=elementos[i][i];

return s;

}

matriz\* matriz::transpuesta(){

matriz \*t = new matriz(n,m);

for(int i=0;i<m;i++){

for(int j=0;j<n;j++)

t->elementos[i][j]=elementos[j][i];

}

return t;

}

//##############################Fin de funciones de la clase Matriz.##############################

//##############################Inicio de funciones de la clase Tridiagonal.##############################

tridiagonal::tridiagonal(){

n=3;

m=3;

diagprin = new float[n];

diaginf = new float[n-1];

diagsup = new float[n-1];

for(int i=0;i<n-1;i++){

diagprin[i]=1;

diaginf[i]=1;

diagsup[i]=1;

}

diagprin[n-1]=1;

}

tridiagonal::tridiagonal(int orden){

n=orden;

m=orden;

diagprin = new float[n];

diaginf = new float [n-1];

diagsup = new float [n-1];

for(int i=0;i<n-1;i++){

diagprin[i] = rand()%10;

diaginf[i] = rand()%10;

diagsup[i] = rand()%10;

}

diagprin[n-1] = rand()%10;

}

tridiagonal::~tridiagonal(){

delete[] diagprin;

delete[] diaginf;

delete[] diagsup;

}

void tridiagonal::imprimir() const{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(i-j==0)

cout<<setw(5)<<diagprin[i];

else

if(i-j==1)

cout<<setw(5)<<diaginf[j];

else

if(i-j==-1)

cout<<setw(5)<<diagsup[i];

else

cout<<setw(5)<<0;

}

cout<<endl;

}

}

tridiagonal\* tridiagonal::operator+(const tridiagonal& t)const{

tridiagonal\* s = new tridiagonal(n);

for(int i=0;i<n-1;i++){

s->diagprin[i] = diagprin[i] + t.diagprin[i];

s->diaginf[i] = diaginf[i] + t.diaginf[i];

s->diagsup[i] = diagsup[i] + t.diagsup[i];

}

s->diagprin[n-1]=diagprin[n-1] + t.diagprin[n-1];

return s;

}

tridiagonal\* tridiagonal::operator-(const tridiagonal& t)const{

tridiagonal\* r = new tridiagonal(n);

for(int i=0;i<n-1;i++){

r->diagprin[i] = diagprin[i] - t.diagprin[i];

r->diaginf[i] = diaginf[i] - t.diaginf[i];

r->diagsup[i] = diagsup[i] - t.diagsup[i];

}

r->diagprin[n-1]=diagprin[n-1] - t.diagprin[n-1];

return r;

}

tridiagonal\* operator\*(float k, const tridiagonal& t){

tridiagonal \*m = new tridiagonal(t.n);

for(int i=0;i<t.n-1;i++){

m->diagprin[i] = k\*t.diagprin[i];

m->diaginf[i] = k\*t.diaginf[i];

m->diagsup[i] = k\*t.diagsup[i];

}

m->diagprin[t.n-1]=k\*t.diagprin[t.n-1];

return m;

}

tridiagonal\* tridiagonal::transpuesta()const{

tridiagonal \*t = new tridiagonal(n);

for(int i=0;i<n-1;i++){

t->diagprin[i]=diagprin[i];

t->diaginf[i]=diagsup[i];

t->diagsup[i]=diaginf[i];

}

t->diagprin[n-1]=diagprin[n-1];

return t;

}

float tridiagonal::traza()const{

float t=0;

for(int i=0;i<n;i++){

t+=diagprin[i];

}

return t;

}

//##############################Fins de funciones de la clase Tridiagonal.##############################

**Clase Polinomio**

#include <iostream>

#include <cmath>

#include <cstdlib>

#include <ctime>

using namespace std;

class polinomio{

friend ostream& operator<<(ostream&, polinomio&);

private:

int grado;

float\* coe;

public:

polinomio();

polinomio(int);

~polinomio();

polinomio\* operator+(const polinomio&)const;

polinomio\* operator-(const polinomio&)const;

float valor\_numerico(float)const;

polinomio\* derivada()const;

polinomio\* antiderivada();

float integral\_definida(float, float);

};

int main(int argc, char\*\* argv) {

srand(time(0));

polinomio p1;

polinomio p2(4);

polinomio \*s = p1 + p2;

cout<<"p1: "<<p1<<endl;

cout<<"p2: "<<p2<<endl;

cout<<"Suma p1 + p2: "<<\*s<<endl;

delete s;

return 0;

}

polinomio::polinomio(){

grado = 2;

coe = new float[grado+1];

for(int i=0;i<grado+1;i++){

coe[i] = 1;

}

}

polinomio::polinomio(int n){

grado = n;

coe = new float[grado+1];

for(int i=0;i<grado;i++){

coe[i] = rand()%10;

}

coe[grado]=1+rand()%9;

}

polinomio::~polinomio(){

delete[] coe;

}

ostream& operator<<(ostream& escribir, polinomio& p){

//escribir<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

for(int i=p.grado;i>0;i--){

if(p.coe[i-1]>=0)

escribir<<p.coe[i]<<"x^"<<i<<"+";

else{

escribir<<p.coe[i]<<"x^"<<i;

}

}

escribir<<p.coe[0]<<endl<<endl;

return escribir;

}

polinomio\* polinomio::operator+(const polinomio& p)const{

polinomio \*s;

if(grado>p.grado){

s= new polinomio(grado);

for(int i=0;i<p.grado+1;i++)

s->coe[i]=coe[i]+p.coe[i];

for(int i=p.grado+1;i<grado+1;i++)

s->coe[i]=coe[i];

}else{

s= new polinomio(p.grado);

for(int i=0;i<grado+1;i++)

s->coe[i]=coe[i]+p.coe[i];

for(int i=grado+1;i<p.grado+1;i++)

s->coe[i]=p.coe[i];

}

return s;

}

polinomio\* polinomio::operator-(const polinomio& p)const{

polinomio \*r;

if(grado>p.grado){

r= new polinomio(grado);

for(int i=0;i<p.grado+1;i++)

r->coe[i]=coe[i]-p.coe[i];

for(int i=p.grado+1;i<grado+1;i++)

r->coe[i]=coe[i];

}else{

r= new polinomio(p.grado);

for(int i=0;i<grado+1;i++)

r->coe[i]=coe[i]-p.coe[i];

for(int i=grado+1;i<p.grado+1;i++)

r->coe[i]=p.coe[i];

}

return r;

}

float polinomio::valor\_numerico(float k)const{

float s=0;

for(int i=0;i<grado;i++)

s+=coe[i]\*pow(k,i);

return s;

}

polinomio\* polinomio::derivada()const{

polinomio \*der = new polinomio(grado-1);

for(int i=grado;i>=1;i--){

der->coe[i-1] = coe[i]\*i;

}

return der;

}

polinomio\* polinomio::antiderivada(){

polinomio \*c = new polinomio(grado+1);

c->coe[0]=0;

for(int i=1;i<grado+2;i++){

c->coe[i]=(coe[i-1]/i);

}

return c;

}

float polinomio::integral\_definida(float a,float b){

polinomio \*c=this->antiderivada();

float t=0,s=0,m=0;

t=c->valor\_numerico(a);

s=c->valor\_numerico(b);

m=t-s;

return m;

}

**Clase BigInt**

#include<iostream>

#include<cstdlib>

#include<cmath>

using namespace std;

class bigint{

friend bigint operator \*(int,const bigint&);

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

private:

int\*digitos;

int dim;

int signo;

public:

bigint();

bigint(int,int);

~bigint();

bool operator ==(const bigint&)const;

bool operator <(const bigint&)const;

bool operator >(const bigint&)const;

bigint operator ++(int)const;

bigint operator --(int)const;

bigint operator +(const bigint&)const;

bigint operator-(const bigint&)const;

bigint operator%(const bigint&)const;

bigint operator\*(const bigint&)const;

};

int main(){

bigint a;

bigint b(10,1);

bigint c(10,1);

bigint g(10,1);

bigint f(5,1);

bigint s(4,1);

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

cout<<"c="<<c<<endl;

cout<<"g="<<g<<endl;

cout<<"f="<<f<<endl;

cout<<"s="<<s<<endl;

if (a==a){

cout<<"a es igual a a"<<endl;

}

else{

cout<<"a no es igual a a"<<endl;

}

cout<<"a++="<<a++<<endl;

cout<<"a--="<<a--<<endl;

cout<<"b+c="<<b+c<<endl;

cout<<"c-b="<<c-b<<endl;

cout<<"c%b="<<c%b<<endl;

// cout<<"2\*a="<<2\*a<<endl;

// cout<<"2\*c="<<2\*c<<endl;

// cout<<"5\*a="<<5\*a<<endl;

cout<<"f\*s="<<s\*f<<endl;

cout<<"a\*b="<<a\*b<<endl;

return 0;

}

bigint::bigint(){

dim=10;

signo=1;

digitos=new int[dim];

digitos[0]=0;

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

digitos[i]=10-i;

}

}

bigint::bigint(int longitud,int sig){

dim=longitud;

signo=sig;

digitos=new int[dim];

digitos[dim-1]=1+rand()% 10;

for(int i=0;i<dim-1;i++){

digitos[i]=rand()% 10;

}

}

bigint::~bigint(){

delete[]digitos;

}

bool bigint::operator ==(const bigint&c)const{

int t=0;

if(signo==c.signo){

if(dim==c.dim){

for(int i=0;i<dim;i++){

if(digitos[i]==c.digitos[i]){

t+=1;

}

}

return t==dim;

}

return false;

}

return false;

}

bool bigint::operator<(const bigint&n)const{

if(signo<n.signo)

return true;

else

if(signo>n.signo)

return false;

else

if(signo<0)

{if(dim<n.dim)

return true;

else

if(dim>n.dim)

return false;

else{

bool resp=true;

int i=dim-1;

while(i<=1&&digitos[i]==n.digitos[i])

{i--;

}

if(digitos[i]<n.digitos[i])

resp=true;

else

resp=false;

return resp;

}

}

else{

if(dim>n.dim)

return true;

else

if(dim>n.dim)

return false;

else{

bool resp=true;

int i=dim-1;

while(i<=1&&digitos[i]==n.digitos[i])

{ i--;

}

if(digitos[i]>n.digitos[i])

resp=true;

else

resp=false;

return resp;

}

}

}

bool bigint::operator>(const bigint&n)const{

if(signo>n.signo)

return true;

else

if(signo<n.signo)

return false;

else

if(signo>0)

{if(dim>n.dim)

return true;

else

if(dim<n.dim)

return false;

else{

bool resp=true;

int i=dim-1;

while(i>=1&&digitos[i]==n.digitos[i])

{i--;

}

if(digitos[i]>n.digitos[i])

resp=true;

else

resp=false;

return resp;

}

}

else{

if(dim<n.dim)

return true;

else

if(dim>n.dim)

return false;

else{

bool resp=true;

int i=dim-1;

while(i>=1&&digitos[i]==n.digitos[i])

{ i--;

}

if(digitos[i]<n.digitos[i])

resp=true;

else

resp=false;

return resp;

}

}

}

bigint bigint::operator ++(int n)const{

if(signo>0){

digitos[0]+=1;

if(digitos[0]==10){

digitos[1]+=1;

}

return \*this;

}

}

bigint bigint::operator --(int n)const{

if(signo>0){

if(digitos[0]==0){

digitos[0]=9;

digitos[1]-=1;

}

else{

digitos[0]-=1;

}

return \*this;

}

}

bigint bigint:: operator+(const bigint&b)const{

int i,mayor,menor;

if(dim>b.dim){

mayor=dim,

menor=b.dim;

}

else{

mayor=b.dim,

menor=dim;

}

bigint\*c=new bigint(mayor+1,signo);

int t,s=0;

if(dim==mayor){

for(i=0;i<mayor;i++){

if(i<menor){

s+=digitos[i]+b.digitos[i];

t=s%10;

c->digitos[i]=t;

if(s==t)

s=0;

else

s=1;

}

else{

s+=digitos[i];

t=s%10;

c->digitos[i]=t;

if(s==t)

s=0;

else

s=1;

}

}

}

else{

for(i=0;i<mayor;i++){

if(i<menor){

s+=digitos[i]+b.digitos[i];

t=s%10;

c->digitos[i]=t;

if(s==t)

s=0;

else

s=1;

}

else{

s+=b.digitos[i];

t=s%10;

c->digitos[i]=t;

if(s==t)

s=0;

else

s=1;

}

}

}

if(s!=0)

c->digitos[mayor]=s;

else

c->dim=mayor;

return \*c;

}

bigint bigint::operator-(const bigint&b)const{

int i,mayor,menor;

int r=0;

bigint\*c;

if(dim>b.dim){

mayor=dim,

menor=b.dim;

}

else{

mayor=b.dim,

menor=dim;

}

if(dim==b.dim){

if((\*this)>b)

{c=new bigint(dim,signo);

for(i=0;i<dim;i++)

{if(digitos[i]+r<b.digitos[i])

{c->digitos[i]=10+digitos[i]-b.digitos[i]+r;

r=-1;

}

else{c->digitos[i]=digitos[i]-b.digitos[i]+r;

r=0;

}

}

}

else{c=new bigint(b.dim,-1);

for(i=0;i<b.dim;i++)

{if(b.digitos[i]+r<digitos[i])

{c->digitos[i]=10+b.digitos[i]-digitos[i]+r;

r=-1;

}

else{c->digitos[i]=b.digitos[i]-digitos[i]+r;

r=0;

}

}

}

}

else{

if(dim==mayor)

{c=new bigint(mayor,signo);

for(i=0;i<mayor;i++)

{if(i<menor)

{if(digitos[i]+r<b.digitos[i])

{c->digitos[i]=10+digitos[i]-b.digitos[i]+r;

r=-1;

}

else{c->digitos[i]=digitos[i]-b.digitos[i]+r;

r=0;

}

}

else{

if(digitos[i]+r<0)

{c->digitos[i]=10+digitos[i]+r;

r=-1;

}

else{c->digitos[i]=digitos[i]+r;

r=0;

}

}

}

}

else{c=new bigint(mayor,-1);

for(i=0;i<mayor;i++)

{if(i<menor)

{if(b.digitos[i]+r<digitos[i])

{c->digitos[i]=10+b.digitos[i]-digitos[i]+r;

r=-1;

}

else{c->digitos[i]=b.digitos[i]-digitos[i]+r;

r=0;

}

}

else{

if(b.digitos[i]+r<0)

{c->digitos[i]=10+b.digitos[i]+r;

r=-1;

}

else{c->digitos[i]=b.digitos[i]+r;

r=0;

}

}

}

}

}

i=c->dim-1;

while(i>=1&&c->digitos[i]==0)

{i--;

}

c->dim=i+1;

return \*c;

}

bigint bigint::operator%(const bigint&d)const{

bigint\*temporal=new bigint(dim,1);

bigint\*residuo=new bigint(dim,1);

\*residuo=\*this;

while(\*residuo>d)

{\*temporal=\*residuo;

\*residuo=\*temporal-d;

}

return \*residuo;

}

bigint operator \*(int n,const bigint&b){

int i,t=0,s=0;

bigint \*c;

c=new bigint(b.dim+1,1);

for(int i=0;i<b.dim+1;i++){

s+=n\*b.digitos[i];

t=s%10;

c->digitos[i]=t;

if(s>=10){

s=(s-t)/10;

}

else{

s=0;

}

}

i=c->dim-1;

while(i>=1&&c->digitos[i]==0)

{i--;

}

c->dim=i+1;

return \*c;

}

bigint bigint::operator\*(const bigint&b)const{

int t;

bigint \*c;

c=new bigint(dim\*dim,1);

for(int i=0;i<dim\*dim;i++){

c->digitos[i]=0;

}

if(dim>=b.dim){

bigint\*elementos=new bigint[b.dim];

bigint\*elem=new bigint[b.dim];

for(int i=0;i<b.dim;i++){

elementos[i]=b.digitos[i]\*(\*this);

}

elem[0]=elementos[0];

for(int i=1;i<b.dim;i++){

elem[i].dim=elementos[i].dim+i;

}

for(int i=1;i<b.dim;i++){

for(int j=0;j<elem[i].dim-i;j++){

elem[i].digitos[j+i]=elementos[i].digitos[j];

}

}

for(int i=1;i<b.dim;i++){

for(int j=0;j<i;j++){

elem[i].digitos[j]=0;

}

}

for(int i=0;i<b.dim;i++){

\*c=\*c+elem[i];

}

t=c->dim-1;

while(t>=1&&c->digitos[t]==0)

{t--;

}

c->dim=t+1;

if(signo!=b.signo){

c->signo=-1;

}

return \*c;

}

else{

bigint\*elementos=new bigint[dim];

bigint\*elem=new bigint[dim];

for(int i=0;i<dim;i++){

elementos[i]=digitos[i]\*b;

}

elem[0]=elementos[0];

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

elem[i].dim=elementos[i].dim+i;

}

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

for(int j=0;j<elem[i].dim-i;j++){

elem[i].digitos[j+i]=elementos[i].digitos[j];

}

}

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

for(int j=0;j<i;j++){

elem[i].digitos[j]=0;

}

}

for(int i=0;i<dim;i++){

\*c=\*c+elem[i];

}

t=c->dim-1;

while(t>=1&&c->digitos[t]==0)

{t--;

}

c->dim=t+1;

if(signo!=b.signo){

c->signo=-1;

}

return \*c;

}

}

ostream& operator <<(ostream&cout,const bigint&b){

if(b.signo<0){

b.digitos[b.dim-1]=-1\*b.digitos[b.dim-1];

}

for(int i=b.dim-1;i>=0;i--){

cout<<b.digitos[i];

}

}