***Arbol B***

#include <iostream>

#include <sstream>

#include "ListaClase.cpp"

#include <ctime>

#include "ListaInt.cpp"

using namespace std;

NodoArbolB::NodoArbolB() {

for (int i=0;i<ORDEN+2;i++)

hijos[i]=NULL;

lastused=-1;

}

void NodoArbolB::set\_dato(int a) {

int i=lastused;

dato[lastused+1]=a;

while((a<dato[i])&&(i>=0)){

int temp=dato[i];

dato[i]=dato[i+1];

dato[i+1]=temp;

i--;

}

lastused++;

}

bool NodoArbolB::is\_dato(int a) {

bool ret=false;

int pos, Li=0,Ls=lastused;

while (Li<=Ls){

pos=(Ls+Li)/2;

if (dato[pos]==a) return true;

else{

if (a<dato[pos]) Ls=pos-1;

else Li=pos+1;

}

}

return ret;

}

int NodoArbolB::get\_pos(int a) {

int ret=-100, pos, Li=0, Ls=lastused;

while (Li<=Ls){

pos=(Ls+Li)/2;

if (dato[pos]==a) return pos;

else{

if (a<dato[pos]) Ls=pos-1;

else Li=pos+1;

}

}

return ret;

}

class ArbolB{

private:

int altura1(int suma,NodoArbolB \*a);

void insert2(int,NodoArbolB\*,Pila &,NodoArbolB\* Hi,NodoArbolB\* Hd);

void toPrint2(NodoArbolB\*);

void toLista2(Lista1\*,NodoArbolB\*);

void borrador(NodoArbolB\* &a);

public:

NodoArbolB \*nodo;

ArbolB() {nodo=NULL;};

ArbolB(NodoArbolB \*p) {nodo=p;};

~ArbolB(void) ;

NodoArbolB \*get\_nodo() {return nodo;};

bool esvacio(void) {return nodo==NULL;};

int altura();

void insert(int a);

Lista1 \*toLista();

void toPrint();

};

ArbolB::~ArbolB(void){

borrador(nodo);

}

void ArbolB::borrador(NodoArbolB\* &a){

if(a==NULL)

{}

else{

for (int i=0;i<=a->get\_lastused()+1;i++){

borrador(a->hijos[i]);

}

delete a;

a=NULL;

}

}

int ArbolB::altura(){

return altura1(0,nodo);

}

int ArbolB::altura1(int suma,NodoArbolB\* a){

if (a==NULL) return suma;

else return altura1(suma+1,a->get\_hijo(0));

}

void ArbolB::insert(int a){

if (nodo==NULL){

nodo=new NodoArbolB();

nodo->set\_dato(a);

}

else{

Pila pila;

insert2(a,nodo,pila,NULL,NULL);

}

}

void ArbolB::insert2(int a,NodoArbolB\* p,Pila &pila,NodoArbolB\* Hi,NodoArbolB\* Hd){

if(p->is\_dato(a)==false){

if((p->es\_hoja())||(Hi!=NULL)){

if (p->is\_dato(a)==false){

if(p->lleno()){

if(Hi==NULL) p->set\_dato(a);

else{

p->set\_dato(a);

int pos=p->get\_pos(a);

p->set\_hijo(pos,Hi);

int inicial=p->get\_lastused()+1;

int j=ORDEN+1;

p->set\_hijo(j,Hd);

for(int i=0;i<ORDEN-pos;i++){

NodoArbolB\* temp=p->get\_hijo(j-1);

p->set\_hijo(j-1,p->get\_hijo(j));

p->set\_hijo(j,temp);j--;

}

}

if (pila.pilavacia()){

//cout<<"entro ";

NodoArbolB\* padre=new NodoArbolB();

NodoArbolB\* hi=new NodoArbolB();

NodoArbolB\* hd=new NodoArbolB();

padre->set\_hijo(0,hi);

padre->set\_hijo(1,hd);

for(int i=0;i<ORDEN/2+1;i++){

if(i<ORDEN/2) hi->set\_dato(nodo->get\_dato(i));

hi->set\_hijo(i,nodo->get\_hijo(i));

}

padre->set\_dato(nodo->get\_dato(ORDEN/2));

int j=0;

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

if(i<ORDEN+1){hd->set\_dato(nodo->get\_dato(i));}

hd->set\_hijo(j,nodo->get\_hijo(i));j++;

}

nodo=padre;

delete p;

}

else{

NodoArbolB\* hi=new NodoArbolB();

NodoArbolB\* hd=new NodoArbolB();

a=p->get\_dato(ORDEN/2);

for(int i=0;i<ORDEN/2+1;i++){

if(i<ORDEN/2){hi->set\_dato(p->get\_dato(i));}

hi->set\_hijo(i,p->get\_hijo(i));

}

int j=0;

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

if(i<ORDEN+1){hd->set\_dato(p->get\_dato(i));}

hd->set\_hijo(j,p->get\_hijo(i));j++;

}

NodoArbolB\* anterior=pila.tope();

pila.pop();

delete p;

insert2(a,anterior,pila,hi,hd);

}

}

else{

if(Hi==NULL){p->set\_dato(a);}

else {

p->set\_dato(a);

int pos=p->get\_pos(a);

p->set\_hijo(pos,Hi);

int inicial=p->get\_lastused()+1;

int ultimoLugar=ORDEN+2;

int j=ORDEN+1;

p->set\_hijo(j,Hd);

for(int i=0;i<ORDEN-pos;i++){

NodoArbolB\* temp=p->get\_hijo(j-1);

p->set\_hijo(j-1,p->get\_hijo(j));

p->set\_hijo(j,temp);j--;

}

}

}

}

}

else{

int i=0;

while ((a>p->get\_dato(i))&&(i<=p->get\_lastused())){

i++;

}

pila.push(p);

insert2(a,p->get\_hijo(i),pila,NULL,NULL);

}

}

}

Lista1 \*ArbolB::toLista(){

Lista1\* lista=new Lista1();

toLista2(lista,nodo);

return lista;

}

void ArbolB::toLista2(Lista1\* lista,NodoArbolB\* a){

if(a==NULL)

{}

else{

int j=0;

for (int i=0;i<=a->get\_lastused()+1;i++){

toLista2(lista,a->get\_hijo(i));

if (i<=a->get\_lastused()){

lista->add(a->get\_dato(i));j++;

}

}

}

}

void ArbolB::toPrint(){

toPrint2 (nodo);

}

void ArbolB::toPrint2(NodoArbolB\* a){

if(a==NULL)

{}

else{

cout<<"\n";int j=0;

for (int i=0;i<=a->get\_lastused()+1;i++){

toPrint2(a->get\_hijo(i));

if (i<=a->get\_lastused()){

cout<<" "<<a->get\_dato(i);j++;

}

}

}

}

***ListaClase***

//#include <cstdlib>

#include <iostream>

#include <sstream>

#define ORDEN 4

using namespace std;

class NodoArbolB{

private:

int dato[ORDEN+1];

int lastused;

public:

NodoArbolB \*hijos[ORDEN+2];

NodoArbolB();

~NodoArbolB(void) {;};

bool lleno() {return lastused==ORDEN-1;};

void set\_dato(int a) ;

bool is\_dato(int a);

int get\_pos(int a);

int get\_dato(int i) {return dato[i];};

int get\_lastused() {return lastused;};

NodoArbolB\* get\_hijo(int a){return hijos[a];}

bool es\_hoja(){return hijos[0]==NULL;}

void set\_hijo(int a,NodoArbolB\* b){hijos[a]=b;}

};

typedef NodoArbolB\* tipolista;

class Nodo{

protected:

tipolista dato;

Nodo \*next;

public:

Nodo() {next=NULL;};

Nodo(tipolista a) {dato=a; next=NULL;};

void set\_dato(tipolista a) {dato=a; };

void set\_next(Nodo \*n) {next=n; };

tipolista get\_dato() {return dato; };

Nodo \*get\_next() {return next; };

bool es\_vacio() {return next==NULL;}

};

class Lista{

protected: Nodo \*czo;

public:

Lista() {czo=new Nodo();};

Lista(Nodo \*n) {czo=n;};

//~Lista(void);

//tipolista Lista::ultimo(tipolista x);

void del(void);

void add(tipolista d);

bool esvacia(void);

tipolista cabeza(void);

Lista \*resto(void);

tipolista toPrint(tipolista p);

string esta(tipolista p,bool t2);

int dondeesta(tipolista p,int t2,int pos);

tipolista maslargo(tipolista t2);

};

class Pila:public Lista{

public:

Pila(){Lista();};

void push(tipolista x){add(x);};

tipolista tope(void){return cabeza();};

void pop(void){del();};

bool pilavacia(){return esvacia();};

};

void Lista::del(void)

{ Nodo \*aux;

aux=czo;

czo=czo->get\_next();

delete aux;

}

void Lista::add(tipolista d)

{

Nodo \*nuevo=new Nodo(d);

nuevo->set\_next(czo);

czo=nuevo;

}

bool Lista::esvacia(void)

{

return czo->es\_vacio();

}

tipolista Lista::cabeza(void)

{

if(esvacia()){

cout<<" Error, Cabeza de lista vacia";

return NULL;

}

return czo->get\_dato();

}

Lista \*Lista::resto(void)

{

Lista \*l=new Lista(czo->get\_next());

return (l);

}

/\*tipolista Lista::toPrint(tipolista p)

{

if (this->esvacia()) {

return p;

} else {

std::ostringstream stm;

stm << this->cabeza()<<"-"<< this->resto()->toPrint(p) << endl;

return stm.str();

}

}

\*/

inline string Lista::esta(tipolista p, bool t2)

{

if (this->esvacia()) {

if(t2) return " si esta";

else return " no esta";

} else {

// esta es la parte que hace el procesamiento

if (p==this->cabeza()) {

t2=true;

}

//

return this->resto()->esta(p,t2);

}

}

int Lista::dondeesta(tipolista p, int t2, int pos)

{

if (this->esvacia()) {

return t2;

} else {

// esta es la parte que hace el procesamiento

if (p==this->cabeza()) {

t2=pos;

}

pos++;

//

return this->resto()->dondeesta(p,t2,pos);

}

}

/\*tipolista Lista::maslargo(tipolista t2)

{

if (this->esvacia()) {

return t2;

} else {

// esta es la parte que hace el procesamiento

if (this->cabeza().size()>t2.size()){

t2=this->cabeza();

}

//

return this->resto()->maslargo(t2);

}

}\*/

***ListaInt***

#include <iostream>

#include <sstream>

using namespace std;

class Nodo1{

private:

int dato;

Nodo1 \*next;

public:

Nodo1() {next=NULL;};

Nodo1(int a) {dato=a; next=NULL;};

void set\_dato(int a) {dato=a; };

void set\_next(Nodo1 \*n) {next=n; };

int get\_dato() {return dato; };

Nodo1 \*get\_next() {return next; };

bool es\_vacio() {return next==NULL;}

};

class Lista1{

private: Nodo1 \*czo;

Lista1 \*copy2(Lista1 \*l, Nodo1 \*last);

void take2(Nodo1 \*last);

public:

Lista1() {czo=new Nodo1();};

Lista1(Nodo1 \*n) {czo=n;};

~Lista1(void);

void add(int d);

bool esvacia(void);

int cabeza(void);

Lista1 \*resto(void);

Lista1 \*copy();

Nodo1 \*get\_czo() {return czo; };

void borrar(void); //borra la cabeza

void drop(int n);

void take(int n);

void concat(Lista1 \*l1);

string toPrint(string p);

int suma(int i);

};

void Lista1::add(int d)

{

Nodo1 \*nuevo=new Nodo1(d);

nuevo->set\_next(czo);

czo=nuevo;

}

bool Lista1::esvacia(void)

{

return czo->es\_vacio();

}

int Lista1::cabeza(void)

{

if(esvacia()){

cout<<" Error, Cabeza de Lista1 vacia";

return -1;

}

return czo->get\_dato();

}

Lista1 \*Lista1::resto(void)

{

Lista1 \*l=new Lista1(czo->get\_next());

return (l);

}

Lista1 \*Lista1::copy2(Lista1 \*l, Nodo1 \*last)

{

if (this->esvacia()){

return l;

} else {

last->set\_dato(this->cabeza());

Nodo1 \*nuevo=new Nodo1();

last->set\_next(nuevo);

return (this->resto())->copy2(l,nuevo);

}

}

Lista1 \*Lista1::copy(void)

{

Lista1 \*l=new Lista1();

return this->copy2(l,l->get\_czo());

}

void Lista1::borrar(void)

{

if(esvacia()){

} else {

Nodo1 \*tmp=czo;

czo=czo->get\_next();

delete tmp;

}

}

Lista1::~Lista1(void){

while (!esvacia()){

this->borrar();

}

}

void Lista1::drop(int n){

if (!(this->esvacia()) && (n>0)) {

this->borrar();

this->drop(n-1);

}

}

void Lista1::take2(Nodo1 \*last){

if (!(last->es\_vacio())){

this->take2(last->get\_next());

}

delete last;

}

void Lista1::take(int n)

{

if(!(this->esvacia())) {

if (n>0){

this->resto()->take(n-1);

} else {

this->take2((this->get\_czo())->get\_next());

(this->get\_czo())->set\_next(NULL);

}

}

}

void Lista1::concat(Lista1 \*l1)

{

if (!(l1->esvacia())){

this->concat(l1->resto());

this->add(l1->cabeza());

}

}

string Lista1::toPrint(string p)

{

if (this->esvacia()) {

return p;

} else {

std::ostringstream stm;

stm << this->cabeza()<<"-"<< this->resto()->toPrint(p) ;

return stm.str();

}

}

int Lista1::suma(int i)

{

if (this->esvacia()) {

return i;

} else {

return this->resto()->suma(i+this->cabeza());

}

}