#define MAX 1000

//Modelo de nodo Arbol binario Busqueda.

struct nodo\_arbol\_binario

{

int dato;

struct nodo\_arbol\_binario\* iz;

struct nodo\_arbol\_binario\* de;

struct nodo\_arbol\_binario\* padre;

};

typedef struct nodo\_arbol\_binario NABinario;

//Modelo de nodo Pila estatica

struct pila\_estatica

{

NABinario\* entrada[MAX];

int tamanio;

int tope;

int entri;

};

typedef struct pila\_estatica PilaE;

NABinario\* **crear\_nodo**(int n,NABinario\* Padre);

void **insertar\_ABB**(NABinario\*&arbol,int n,NABinario\* Padre);

void **mostrar\_arbol**(NABinario\* arbol, int cont);

bool **Busqueda\_ABB**(NABinario\* arbol,int n);

NABinario\***minimo**(NABinario\*arbol);

void **destruir**(NABinario\*ndo);

void **eliminar\_nodo**(NABinario\*ndo\_eliminar);

void **reemplazar**(NABinario\* arbol,NABinario\*nodo\_2);

//Barridos Arboles Recursivos

void **abinario\_preorden\_recursivo**(NABinario \*arbol);

void **abinario\_inorden\_recursivo**(NABinario \*arbol);

void **abinario\_postorden\_recursivo**(NABinario \*arbol);

void **abinario\_mostrar\_recursivo** (NABinario\* arbol, int tabulado);

void **abinario\_mostrar\_recursivo2** (NABinario\* arbol, int n);

int **main**()

{

cout << "Hello World!" << endl;

return 0;

}

//Crea un nodo para un arbol binario de busqueda

NABinario\* **crear\_nodo**(int n, NABinario \*Padre){

NABinario\* nuevo\_nodo=new NABinario();

nuevo\_nodo->dato=n;

nuevo\_nodo->de=NULL;

nuevo\_nodo->iz=NULL;

nuevo\_nodo->padre=Padre;

return nuevo\_nodo;

}

//Algoritmo para insertar nodos a un arbol binario de busqueda

void **insertar\_ABB**(NABinario\*&arbol,int n,NABinario\* Padre){

if(arbol==NULL)//si el arbol esta vacio

{

NABinario\* nuevo\_nodo=crear\_nodo(n,Padre);

arbol=nuevo\_nodo;

}

else{//si el arbol tiene uno o mas nohttps://www.google.com/dos

int valor\_raiz=arbol->dato;//obtenemos el valor de la raiz

//si el elemento es menor , se inserta a la izquierda

if(n<valor\_raiz)

{ insertar\_ABB(arbol->iz,n,arbol);}

else{//si el elemento es mayor, se inserta a la derecha

insertar\_ABB(arbol->de,n,arbol);

}

}

}

//Mostrar Arbol Girado a la izquierda

void **mostrar\_arbol**(NABinario\* arbol,int cont){

if(arbol==NULL)return;

else{

mostrar\_arbol(arbol->de,cont+1);

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

{

cout<<" ";//3 espacios para separar nodos

}

cout<<arbol->dato<<" "<<endl;

mostrar\_arbol(arbol->iz,cont+1);

}

}

//Buscar Nodo en un arbol

bool **Busqueda\_ABB**(NABinario\* arbol,int n){

//si el arbol esta vacio no hago nada

if(arbol==NULL)return false;

//si encuentro el dato , retorno true que indica que lo encontro en la lista

else if(arbol->dato==n)return true;

//si no lo encontre , me voy bien por la izquierda

else if(n<arbol->dato)return Busqueda\_ABB(arbol->iz,n);

//si no lo encontre, me voy bien por la derecha

else return Busqueda\_ABB(arbol->de,n);

}

//Algoritmos para hacer eliminacion de nodos en un arbol

//funcion determinar nodo mas izquierdo posible

NABinario\***minimo**(NABinario\*arbol){

if(arbol==NULL)return NULL;//arbol vacio

if(arbol->iz)return minimo(arbol->iz);

else return arbol;

}

//Algoritmo para reemplazar un nodo por otro

void **reemplazar**(NABinario\* arbol,NABinario\*nodo\_2){

if(arbol->padre)

{

//di tiene padre procedemos a asignarle el nuevo hijo

if(arbol->dato==arbol->padre->de->dato)arbol->padre->iz=nodo\_2;

else if (arbol->dato==arbol->padre->de->dato)arbol->padre->de=nodo\_2;

}

//al hijo le decimos quien es su padre

if(arbol)nodo\_2->padre=arbol->padre;

}

//funcion destruir nodo

void **destruir**(NABinario\*ndo){

ndo->iz=NULL;

ndo->de=NULL;

delete ndo;

}

//Algoritmo para eliminar un nodo 3 casos vistos

void **eliminar\_nodo**(NABinario\*ndo\_eliminar){

if(ndo\_eliminar->iz && ndo\_eliminar->de)

{int dato;

NABinario\*menor=minimo(ndo\_eliminar->de);

ndo\_eliminar->dato=menor->dato;

eliminar\_nodo(menor);

}

else if(ndo\_eliminar->iz){

reemplazar(ndo\_eliminar,ndo\_eliminar->iz);

destruir(ndo\_eliminar);

}

else if(ndo\_eliminar->de){

reemplazar(ndo\_eliminar,ndo\_eliminar->de);

destruir(ndo\_eliminar);

}

else {

reemplazar(ndo\_eliminar,NULL);

destruir(ndo\_eliminar);

}

}

//Algoritmo para eliminar un nodo dvoid abinario\_inorden\_recursivo(NABinario \*arbol)el arbol y acomodandolo

void **eliminar\_ABB**(NABinario\*arbol,int n){

if(arbol==NULL)return;

else if (n<arbol->dato)eliminar\_ABB(arbol->iz,n);

else if (n>arbol->dato)eliminar\_ABB(arbol->de,n);

else eliminar\_nodo(arbol);

}

//Barridos de forma recursiva.

void **abinario\_preorden\_recursivo**(NABinario \*arbol){

if(arbol!=NULL){

cout<<arbol->dato<<"-->";

abinario\_preorden\_recursivo(arbol->iz);

abinario\_preorden\_recursivo(arbol->de);

}

}

//de este barrido salen ordenados

void **abinario\_inorden\_recursivo**(NABinario \*arbol){

if(arbol!=NULL){

abinario\_preorden\_recursivo(arbol->iz);

cout<<arbol->dato<<"-->";

abinario\_preorden\_recursivo(arbol->de);

}

}

void **abinario\_postorden\_recursivo**(NABinario \*arbol){

if(arbol!=NULL){

abinario\_preorden\_recursivo(arbol->iz);

abinario\_preorden\_recursivo(arbol->de);

cout<<arbol->dato<<"-->";

}

}

//Mostrvoid abinario\_mostrar\_recursivo2 (NABinario\* arbol, int n)ar de forma recursiva

void **abinario\_mostrar\_recursivo** (NABinario\* arbol, int tabulado)

{

if (arbol != NULL)

{

cout << string (tabulado, '\t');

cout << "Nodo: " << arbol->dato << " | " << "Iz-> ";

if (arbol->iz != NULL)

cout << arbol->iz->dato;

else

cout << "NULL";

cout << " " << "De-> ";

if (arbol->de != NULL)

cout << arbol->de->dato;

else

cout << "NULL";

cout << endl;

tabulado++;

abinario\_mostrar\_recursivo (arbol->iz, tabulado);

abinario\_mostrar\_recursivo (arbol->de, tabulado);

}

}

void **abinario\_mostrar\_recursivo2** (NABinario\* arbol, int n)

{

if (arbol == NULL)

return;

abinario\_mostrar\_recursivo2 (arbol->de, n+1);

cout << string (n, '\t') << arbol->dato << endl;

abinario\_mostrar\_recursivo2(arbol->iz, n+1);

}

//Alta recursiva..

void **abinariob\_alta\_recursivo** (NABinario\* &arbol, int nuevo\_dato)

{

if (arbol == NULL)

{

arbol = new (NABinario);

arbol->iz = NULL; arbol->de = NULL;

arbol->dato = nuevo\_dato;

}

else if (nuevo\_dato < arbol->dato)

abinariob\_alta\_recursivo (arbol->iz, nuevo\_dato);

else if (nuevo\_dato > arbol->dato)

abinariob\_alta\_recursivo (arbol->de, nuevo\_dato);

}

//implementacion de pila estatica para algoritmos iterativos

bool **pila\_vacia** (PilaE \*pila)

{

bool Pvac=false;

if(pila == NULL){

Pvac=true;

}

return Pvac;

}

void **pila\_agregar** (PilaE\* &pila,int n)

{

if(pila->tope<pila->tamanio){

pila->entrada[pila->tope]->dato=n;

pila->tope++;

}else{

cout<<"Error al agregar en pila verifique algoritmo"<<endl;

}

}

int **pila\_sacar** (PilaE\* &pila)

{ if(pila->tope>0){

pila->tope--;

}

else{

cout<<"Error al sacar elementos de una pila"<<endl;

}

return(pila->entrada[pila->tope]->dato);

}

//Barridos iterativos

void **abinario\_preorden\_iterativo** (NABinario\* arbol)

{

NABinario\* aux;

PilaE\* pila; pila->tamanio = MAX; pila->tope=0;

if (arbol != NULL)

pila\_agregar (pila, arbol->dato);

while (!pila\_vacia (pila))

{

aux->dato = pila\_sacar (pila);

cout << aux->dato << " ";

if (aux->de != NULL)

pila\_agregar (pila, aux->de->dato);

if (aux->iz != NULL)

pila\_agregar (pila, aux->iz->dato);

}

}

//pila estatica pero para InOrden SACAR

NABinario\* **pila\_sacarent**(PilaE\* &pila,int &entri) {

if (pila->tope > 0)

{

pila->tope--;

entri = pila->entrada[pila->tope]->dato;

return pila->entrada[pila->tope];

}

else return NULL;

}

//Pila agregar

void **pila\_agregarent**(PilaE\* &pila,int &entri,NABinario\* arbol)

{

if(pila->tope<pila->tamanio){

pila->entrada[pila->tope]->dato=arbol->dato;

pila->tope++;

pila->entri=entri;

}else{

cout<<"Error al agregar en pila verifique algoritmo"<<endl;

}

}

//Recorrido arbol IN-Orden

void **abinario\_inorden\_iterativo** (NABinario\* arbol)

{

NABinario\* aux;

PilaE\* pila; pila->tamanio = MAX; pila->tope = 0;

int ent;

int n=1;

pila\_agregarent(pila,n, arbol);

while (!pila\_vacia (pila))

{

aux = pila\_sacarent(pila,ent);

if (ent == 1 ) {

pila\_agregarent(pila,n,aux);

if (aux->iz != NULL) pila\_agregarent(pila,n, aux->iz);

} else {

cout << aux->dato << " ";

if (aux->de != NULL)

pila\_agregarent(pila,n, aux->de);

}

}

}

//Recorrido Post-Orden

void **abinario\_postorden\_iterativo** (NABinario\* arbol)

{

NABinario\* aux;

PilaE\* pila; pila->tamanio = MAX; pila->tope = 0;

int ent;

int n=1;

pila\_agregarent(pila,n, arbol);

while (!pila\_vacia (pila))

{

aux = pila\_sacarent(pila,ent);

switch(ent)

{

case 1:

pila\_agregarent(pila,n,aux);

if (aux->iz != NULL)

pila\_agregarent(pila,n, aux->iz);

break;

case 2:

pila\_agregarent(pila,n,aux);

if (aux->de != NULL)

pila\_agregarent(pila,n, aux->de);

break;

case 3:

cout << aux->dato << " ";

break;

}

}

}

//Mostrar todos los nodos Hojas guia nº3 ejercicio 4

void **abinario\_mostrarhoja\_recursivo**(NABinario \*arbol){

if(arbol==NULL){ cout<<"arbol vacio"<<endl;}

if((arbol->iz)&&(arbol->de)==NULL){

cout<<arbol->dato<<"-->";

}

abinario\_preorden\_recursivo(arbol->iz);

abinario\_preorden\_recursivo(arbol->de);

}

void **eliminar\_hoja**(NABinario\* &arbol,NABinario\* padre){

if(arbol==NULL)return;

padre=arbol;

if(arbol->iz!=NULL)eliminar\_hoja(arbol->iz,padre);

else if(arbol->de!=NULL)eliminar\_hoja(arbol->de,padre);

else if((arbol->de==NULL)&&(arbol->iz==NULL)){

if(arbol->dato==padre->iz->dato)padre->iz=NULL;

else padre->de=NULL;

delete arbol;

}

}

int **abinario\_altura\_recursivo**(NABinario \*arbol){

int alt\_iz,alt\_de;

if(arbol==NULL)return -1;

else

alt\_iz=abinario\_altura\_recursivo(arbol->iz);

alt\_de=abinario\_altura\_recursivo(arbol->de);

if(alt\_iz<alt\_de)return alt\_iz +1;

else return alt\_de+1;

}