**ÁRBOL BINARIO DE BUSQUEDA CON VARIAS FUNCIONES;**

**Busqueda, eliminar, mostrar, pre Orden, in Orden, post orden**

#include <iostream>

using namespace std;

struct nArbol{

int dato;

struct nArbol\* izq;

struct nArbol\* der;

struct nArbol\* padre;

};

typedef struct nArbol NArbol;

NArbol\* arbol=NULL;

void pedirDatos();

void insertar\_arbol(NArbol\*&, int, NArbol\*);

void mostrar\_arbol\_girado\_a\_la\_Izquierda(NArbol\*, int);

bool busqueda\_dato(NArbol\*, int);

void eliminar(NArbol\*, int);

void eliminar\_nodo(NArbol\*);

NArbol\* minimo(NArbol\*);

void reemplazar(NArbol\*, NArbol\*);

void borrar\_nodo(NArbol\*);

void inOrden(NArbol\*);

void postOrden(NArbol\*);

void preOrden(NArbol\*);

int main()

{

int cont=0, op, dato;

pedirDatos();

cout<<endl<<endl;

mostrar\_arbol\_girado\_a\_la\_Izquierda(arbol, cont);

cout<<endl<<endl;

system("pause");

system("cls");

do{cout<<"\n\nDesea saber si un dato ingresado existe en el arbol?\n 1=Si ---- 2=No : ";

cin>>op;

if (op==1){

cout<<"\nIntroduzca el dato a buscar: ";

cin>>dato;

bool resp= busqueda\_dato(arbol, dato);

if(resp==true){cout<<"\nEl dato existe! "<<endl;}

else{cout<<"\nNo existe el dato en el arbol..."<<endl;}}

cout<<endl<<endl;}while(op!=2);

system("cls");

cout<<"\nMostrando en Pre Orden..."<<endl;

preOrden(arbol);

cout<<endl<<endl;

cout<<"\nMostrando en In Orden..."<<endl;

inOrden(arbol);

cout<<endl<<endl;

cout<<"\nMostrando en Post Orden..."<<endl;

postOrden(arbol);

cout<<endl<<endl;

do{

cout<<"\n\nDesea Eliminar un nodo? SI= 1, NO= 2 : ";

cin>>op;

cout<<"Ingrese el dato a eliminar: ";

cin>>dato;

if(busqueda\_dato(arbol, dato)==true){

eliminar(arbol, dato);

cout<<"\nMostrando arbol con nodo [ "<<dato<<"] eliminado"<<endl;

cont=0;

mostrar\_arbol\_girado\_a\_la\_Izquierda(arbol, cont);}

else{cout<<"\nEl dato ingresado no existe, por ende no fue eliminado"<<endl;

}

} while(op!=2);

return 0;

}

void pedirDatos(){

int dato, resp;

do{ cout<<" Ingrese el numero para agregar al arbol: ";

cin>>dato;

insertar\_arbol(arbol, dato, NULL);

cout<<"Otro dato? SI=1, NO=2: ";

cin>>resp;}while(resp!=2);

}

void insertar\_arbol(NArbol \*&arbol , int n\_dato, NArbol\* padre){

if(arbol==NULL){

NArbol\* n\_nodo= new NArbol();

n\_nodo->dato= n\_dato;

n\_nodo->der=NULL;

n\_nodo->izq=NULL;

n\_nodo->padre=padre;

arbol=n\_nodo;}

else{

int valor\_raiz= arbol->dato;

if(n\_dato<valor\_raiz){insertar\_arbol(arbol->izq, n\_dato, arbol);}

else if(n\_dato>valor\_raiz){insertar\_arbol(arbol->der, n\_dato, arbol);}

else return;

}}

void mostrar\_arbol\_girado\_a\_la\_Izquierda(NArbol\* arbol, int cont){

if(arbol==NULL)return;

else{ mostrar\_arbol\_girado\_a\_la\_Izquierda(arbol->der, cont+1);

{

for (int i=0; i<cont; i++) { cout<<" "; }

cout<<arbol->dato<<endl;

mostrar\_arbol\_girado\_a\_la\_Izquierda(arbol->izq, cont+1);

}} }

bool busqueda\_dato(NArbol\* arbol, int n\_dato){

if(arbol==NULL)return false;

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

else if(n\_dato<arbol->dato){return busqueda\_dato(arbol->izq, n\_dato);}

else{return busqueda\_dato(arbol->der, n\_dato);}

}

void eliminar(NArbol\* arbol , int dato){

if(arbol==NULL)return;

else if(dato<arbol->dato){eliminar(arbol->izq, dato);}

else if(dato>arbol->dato){eliminar(arbol->der, dato);}

else{ eliminar\_nodo(arbol); }

}

void eliminar\_nodo(NArbol\* a\_eliminar){

/\*Si el nodo a eliminar tiene dos hijos... va lo siguiente\*/

if(a\_eliminar->der&&a\_eliminar->izq){

NArbol\* menor=minimo(a\_eliminar->der);

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

eliminar\_nodo(menor);

}

//cuando el nodo tiene un solo hijo

else if(a\_eliminar->izq){/\*si tiene hijo izq\*/

reemplazar(a\_eliminar, a\_eliminar->izq);

borrar\_nodo(a\_eliminar);

}

else if(a\_eliminar->der){/\*si tiene hijo der\*/

reemplazar(a\_eliminar, a\_eliminar->der);

borrar\_nodo(a\_eliminar);

}

else{/\*si es un nodo hoja\*/

reemplazar(a\_eliminar, NULL);

borrar\_nodo(a\_eliminar);

}

}

/\*nos devuelve el num mas izquierdo posible\*/

NArbol\* minimo(NArbol\* arbol){

if(arbol==NULL)return NULL;

if(arbol->izq){return minimo(arbol->izq);}//si tiene hijo izq busco el menor

else{return arbol;}// Si no tiene hijo izq retornamos el mismo nodo

}

/\*Lo que sigue me sirve cuando el nodo a ser eliminado tiene un hijo solo, y cuando es nodo hoja\*/

void reemplazar(NArbol \* arbol, NArbol \* n\_nodo){

if(arbol->padre){

if (arbol->dato==arbol->padre->izq->dato){

arbol->padre->izq= n\_nodo;

}

else if(arbol->dato==arbol->padre->der->dato) {

arbol->padre->der= n\_nodo;

}

}

if(n\_nodo){n\_nodo->padre=arbol->padre;}

}

/\*definitivamente borra el nodo luego de las comprobaciones necesarias\*/

void borrar\_nodo(NArbol \* nodo){

nodo->izq=NULL;

nodo->der=NULL;

delete nodo;

}

void inOrden(NArbol\* arbol){

if(arbol==NULL)return;

else{inOrden(arbol->izq);

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

inOrden(arbol->der);

}

}

void postOrden(NArbol\* arbol){

if(arbol==NULL)return;

else{

postOrden(arbol->izq);

postOrden(arbol->der);

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

}

}

void preOrden(NArbol\* arbol){

if(arbol==NULL)return;

else{

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

preOrden(arbol->izq);

preOrden(arbol->der);

}

}