**void crearArbol(t\_arbol\*a)**

{

\*a=NULL;

}

**int cargarInfo(t\_info \*d)**

{

printf("Ingresar la clave: ");

fflush(stdin);

scanf("%d",&d->clave);

return 1;

}

/\***int insertarEnArbolRecursivo(t\_arbol\*p,t\_info\*d,t\_cmp comp)** ///CARGA EN FORMA RECURSIVA

{

int cmp;

if(\*p)

{

if((cmp=comp(d,&(\*p)->info))>0)

return insertarEnArbolRecursivo(&(\*p)->der,d,comp);

else if(cmp<0)

return insertarEnArbolRecursivo(&(\*p)->izq,d,comp);

else return CLAVE\_DUP;

}

(\*p)=(t\_nodo\*)malloc(sizeof(t\_nodo));

if(!\*p) return SIN\_MEMORIA;

(\*p)->izq=(\*p)->der=NULL; ///pone en null izq y der

(\*p)->info=\*d;

return TODO\_BIEN;

}\*/

**int insertarEnArbolRecursivo(t\_arbol\*p,t\_info\*d,t\_cmp comp) ///CARGA RECURSIVA OTRA VERSION**

{

int cmp;

if(\*p==NULL)

{

(\*p)=(t\_nodo\*)malloc(sizeof(t\_nodo));

if(\*p != NULL)

{

(\*p)->info=\*d;

(\*p)->izq=(\*p)->der=NULL;

} else return SIN\_MEMORIA;

}

else

{

if((cmp=comp(d,&(\*p)->info))>0)

return insertarEnArbolRecursivo(&(\*p)->der,d,comp);

else if(cmp<0)

return insertarEnArbolRecursivo(&(\*p)->izq,d,comp);

else return CLAVE\_DUP;

}

return TODO\_BIEN;

}

int comp(const void \*e1, const void \*e2)

{

return \*(int\*)e1 - \*(int\*)e2; ///menor a mayor

//return \*(int\*)e2 - \*(int\*)e1; ///mayor a menor

}

void mostrarInfo(t\_info\*d)

{

printf(" %d",d->clave);

}

**void recorridoPreOrden(t\_arbol\* p)**

{

if(!\*p)

return;

mostrarInfo(&(\*p)->info);

recorridoPreOrden(&(\*p)->izq);

recorridoPreOrden(&(\*p)->der);

}

**void recorridoPostOrden(t\_arbol \*p)**

{

if(!\*p)

return;

recorridoPostOrden(&(\*p)->izq);

recorridoPostOrden(&(\*p)->der);

mostrarInfo(&(\*p)->info);

}

**void Arbol\_A\_Bin(t\_arbol \*a, FILE \*arch)** ///DE ARBOL LO PASO A UN ARCHIVO BIN

{

t\_info info;

if(!\*a)

return;

info = (\*a) -> info;

fwrite(&info, sizeof(t\_info), 1, arch);

Arbol\_A\_Bin(&(\*a) -> izq, arch);

Arbol\_A\_Bin(&(\*a) -> der, arch);

}

**void Arbol\_A\_Texto(t\_arbol \*a, FILE \*arch\_txt)** ///DE ARBOL LO PASO A ARCHIVO DE TEXTO

{

t\_info info;

if(!\*a)

return;

info = (\*a) -> info;

fprintf(arch\_txt,"%d", info.clave);

Arbol\_A\_Texto(&(\*a) -> izq,arch\_txt);

Arbol\_A\_Texto(&(\*a) -> der,arch\_txt);

}

**void ArchBin\_A\_Arbol(t\_arbol \*a,FILE \*arch, t\_cmp comp)**

{

t\_info info;

fread(&info, sizeof(t\_info), 1, arch);

while(!feof(arch))

{

insertarEnArbolRecursivo(&arbol, &info, comp);

fread(&info, sizeof(t\_info), 1, arch);

}

}

**int McontarHojas(const t\_arbol \*a) ///MUESTRA Y CUENTA HOJAS**

{

if(!\*a)

return 0;

if((!(\*a)->izq && !(\*a)->der))

{

mostrarInfo(&(\*a)->info);

return 1;

}

return McontarHojas(&(\*a)->izq)+McontarHojas(&(\*a)->der);

}

**int contarNodos(const t\_arbol \*a)**

{

if(!\*a)

return 0;

return contarNodos(&(\*a)->izq)+contarNodos(&(\*a)->der)+1;

}

**int contarNodosNoHojas(const t\_arbol \*a)**

{

if(!\*a || (!(\*a)->izq && !(\*a)->der))

return 0;

return contarNodosNoHojas(&(\*a)->izq)+contarNodosNoHojas(&(\*a)->der)+1;

}

**int ContarNodosSoloHijosPorIzq(const t\_arbol \*a)**

{

if(!\*a)

return 0;

if(!(\*a)->der && (\*a)->izq)

return 1+ContarNodosSoloHijosPorIzq(&(\*a)->izq);

else

return ContarNodosSoloHijosPorIzq(&(\*a)->izq)+ContarNodosSoloHijosPorIzq(&(\*a)->der);

}

**int contarNHI(const t\_arbol \*a)**

{

if(!\*a)

return 0;

if((\*a)->izq)

{

mostrarInfo(&(\*a)->info);

return 1+contarNHI(&(\*a)->izq)+contarNHI(&(\*a)->der);

}

return contarNHI(&(\*a)->izq)+contarNHI(&(\*a)->der);

}

**void eliminarArbol(t\_arbol \*a)**

{

if(!\*a)

return;

if((\*a)->izq)

eliminarArbol(&(\*a)->izq);

if((\*a)->der)

eliminarArbol(&(\*a)->der);

free(\*a);

\*a=NULL;

}

**void PodarHojas(t\_arbol \*a)**

{

if(!\*a)

return;

if(!(\*a)->izq && !(\*a)->der)

{

free(\*a);

\*a=NULL;

return;

}

PodarHojas(&(\*a)->izq);

PodarHojas(&(\*a)->der);

}

**int ContarHojasEliminadas(t\_arbol \*a)**

{

if(!\*a)

return 0;

if(!(\*a)->izq && !(\*a)->der)

{

free(\*a);

\*a=NULL;

return 1;

}

return ContarHojasEliminadas(&(\*a)->izq)+ContarHojasEliminadas(&(\*a)->der);

}

**int alturaArbol(const t\_arbol \*a)**

{

int hizq,hder;

if(!\*a)

return 0;

hizq=alturaArbol(&(\*a)->izq);

hder=alturaArbol(&(\*a)->der);

return (hizq>hder?hizq:hder)+1;

}

**void mostrarNodosDeAltura(const t\_arbol \*a,const int alt)** ///DEPENDIENDO LA ALTURA MUESTRO LOS NODOS DE ESA ALTURA

{

if(!\*a)

return;

if(!alt)

return;

if(alt!=1)

{

mostrarNodosDeAltura(&(\*a)->izq,alt-1);

mostrarNodosDeAltura(&(\*a)->der,alt-1);

}

else mostrarInfo(&(\*a)->info);

}

**int ContarNodosDeAltura(const t\_arbol \*a, const int alt)**

{

if(!\*a || !alt)

return 0;

if(alt==1)

{

return 1;

}

return ContarNodosDeAltura(&(\*a)->izq,alt-1)+ContarNodosDeAltura(&(\*a)->der,alt-1);

}

**void mostrarNodosHastaAltura(const t\_arbol \*a,const int alt)** ///EXCLUYE LA ALTURA PARA ABAJO

{

if(!\*a || !alt)

return;

if(alt>1)

{

mostrarInfo(&(\*a)->info);

mostrarNodosHastaAltura(&(\*a)->izq,alt-1);

mostrarNodosHastaAltura(&(\*a)->der,alt-1);

}

}

**int ContarNodosHastaAltura(const t\_arbol \*a, const int alt)**

{

if(!\*a)

return 0;

if(alt>1)

return ContarNodosHastaAltura(&(\*a)->izq,alt-1)+ContarNodosHastaAltura(&(\*a)->der,alt-1);

return 1;

}

**void mostrarNodosHastaAlturaI(const t\_arbol \*a,const int alt)**

{

if(!\*a || !alt)

return;

if(alt>=1)

{

mostrarInfo(&(\*a)->info);

mostrarNodosHastaAlturaI(&(\*a)->izq,alt-1);

mostrarNodosHastaAlturaI(&(\*a)->der,alt-1);

} }

**int ContarNodosHastaAlturaI(const t\_arbol \*a, const int alt)** ///CONSULTAR POR EL +1

{

if(!\*a || !alt)

return 0;

return ContarNodosHastaAlturaI(&(\*a)->izq,alt-1)+ContarNodosHastaAlturaI(&(\*a)->der,alt-1)+1;

}

**void mostrarNodosDespuesAlturaI(const t\_arbol \*a,const int alt)** ///INCLUYO LA ALTURA. CONSULTAR

{

if(!\*a)

return;

if(alt<=1)

{

mostrarInfo(&(\*a)->info);

}

mostrarNodosDespuesAlturaI(&(\*a)->izq,alt-1);

mostrarNodosDespuesAlturaI(&(\*a)->der,alt-1);

}

**void mostrarNodosDespuesAlturaE(const t\_arbol \*a,const int alt)** ///ENCLUYO LA ALTURA

{

if(!\*a)

return;

if(alt<1)

{

mostrarInfo(&(\*a)->info);

}

mostrarNodosDespuesAlturaE(&(\*a)->izq,alt-1);

mostrarNodosDespuesAlturaE(&(\*a)->der,alt-1);

}

**int contarNodosDespuesAlturaE(const t\_arbol \*a,const int alt)**

{

if(!\*a)

return 0;

if(alt<1)

return 1;

return contarNodosDespuesAlturaE(&(\*a)->izq,alt-1)+contarNodosDespuesAlturaE(&(\*a)->der,alt-1);

}

**int ArbolCompleto(t\_arbol \*a)** /// TODOS sus nodos, salvo las hojas tienen un subarbol por izq y der. TODAS las hojas tienen el mismo nivel/prof/alt

{

int h=alturaArbol(a);

return EsCompleto(a,h);

}

**int EsCompleto(t\_arbol \*a, int h)**

{

if(\*a)

return EsCompleto(&(\*a)->izq,h-1) && EsCompleto(&(\*a)->der,h-1);

return h==0;

}

**int ArbolBalanceado(t\_arbol\*a)** ///Si es arbol completo de alt/prof con una alt/prof UNA MENOS que la propia

{

int h=alturaArbol(a);

return EsBalanceado(a,h);

}

**int EsBalanceado(t\_arbol\*a,int h)**

{

if(\*a)

return EsBalanceado(&(\*a)->izq,h-1) && EsBalanceado(&(\*a)->der,h-1);

return h<=1;

}

**int EsAVL(const t\_arbol\*a)** ///La diferencia de los subarboles de cada uno de sus nodos es COMO MUCHO 1

{

if(!\*a)

return 1;

if(abs(alturaArbol(&(\*a)->izq - alturaArbol(&(\*a)->der))<=1))

if(EsAVL(&(\*a)->izq) && EsAVL(&(\*a)->der))

return 1;

return 0;

}

**int RecorrerPreOrdenyGuardar(t\_arbol \*arbol, FILE \*\*fp)**

{

    if(\*arbol==NULL)

        return 0;

    MostrarInfo(&(\*arbol)->info);

    fwrite(&(\*arbol)->info,sizeof(t\_info),1,\*fp);

    RecorrerPreOrdenyGuardar(&(\*arbol)->izq, &(\*fp));

    RecorrerPreOrdenyGuardar(&(\*arbol)->der, &(\*fp));

    return 1;

}

**int LeerDeArchivoyCargarArbol(t\_arbol \*arbol, FILE \*\*fp, int (\*cmp)(t\_info \*d1, t\_info \*d2))**

{

    t\_info info;

    fread(&info, sizeof(t\_info),1,\*fp);

    while(!feof(\*fp))

    {

        while(\*arbol!=NULL)

        {

            if(cmp(&info, &(\*arbol)->info)>0)

                arbol=&(\*arbol)->der;

            else if(cmp(&info, &(\*arbol)->info)<0)

                arbol=&(\*arbol)->izq;

            else

                break;

        }

        (\*arbol) = (t\_nodo \*)malloc(sizeof(t\_nodo));

        if(\*arbol==NULL)

            return 0;

        (\*arbol)->izq = NULL;

        (\*arbol)->der = NULL;

        (\*arbol)->info = info;

        fread(&info, sizeof(t\_info),1,\*fp);

    }

    return 1;

}