***Lista doblemente enlazada***

**int mostrar\_dlista(t\_dlista \*dlista, void (\*mostrar)(void \*a))**

{

    t\_nodo \*actual;

    actual = \*dlista;

    while(actual && actual -> pant)

        actual = actual -> pant;

    while(actual)

    {

        mostrar(&(actual) -> info);

        actual = actual -> psig;

    }

    return 1;

}

**void crear\_dlista(t\_dlista \* dlista)**

{

    (\*dlista) = NULL;

}

**int dlista\_vacia(t\_dlista \* dlista)**

{

    return \*dlista == NULL;

}

**int insertar\_dlista\_ordenada\_sin\_repetidos(t\_dlista \*dlista, t\_info \*info, t\_cmp cmp)**

{

    int comp;

    t\_nodo \*nue, \*actual, \*pnant = NULL, \*pnsig  = NULL;

    actual = \*dlista;

    if(actual)

    {

        while(actual -> psig && cmp(info, &actual -> info) > 0)

            actual = actual -> psig;

        while(actual -> pant && cmp(info, &actual -> info) < 0)

            actual = actual -> pant;

        if( (comp = cmp(info, &actual -> info)) == 0)

            return CLAVE\_DUPLICADA;

        if(comp > 0)

        {         ///tengo q insertar derecho

            pnant = actual;

            pnsig = actual -> psig;

        }

        else

        {       /// tengo q insertar izquierda

            pnsig = actual;

            pnant = actual  -> pant;

        }

    }

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

    if(!nue)

        return 0;

    nue -> info = \*info;

    nue -> pant = pnant;

    nue -> psig = pnsig;

    if(pnant)

        pnant -> psig = nue;

    if(pnsig)

        pnsig -> pant = nue;

    (\*dlista) = nue;

    return 0;

}

Agrego está funcion para ir sacando de la lista a partir del primero

**int sacardelista(t\_listaD \*l,t\_infoLD \*d)**

{

    t\_nodoLD \*aux;

    if(\*l == NULL)

        return 0;

    while((\*l) && (\*l)->izq)

        l = &(\*l)->izq;

    aux = \*l;

    \*d = aux->info;

    if(!(\*l)->izq && !(\*l)->der) ///1 solo elemento///

    {

        \*l = NULL;

    }

    else

    {

        (\*l) = (\*l)->der;

        (\*l)->izq = NULL;

    }

    free(aux);

    return 1;

}

**int insertaralfinal(t\_lista\* p,t\_info\* d)**

{

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

if(nue!=NULL)

return 0;

nue->info=\*d;

nue->sig=NULL;

while(\*p)

p=&(\*p)->sig;

\*p=nue;

return 1;

}

**int insertaralprincipio(t\_lista\* p,t\_info\* d)**

{

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

if(nue==NULL)

return 0;

nue->info=\*d;

nue->sig=\*p;

\*p=nue;

return 1;

}

**int insertarsinohayclave(t\_lista\* p,t\_info\* d)**

{

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

if(nue!=NULL)

return 0;

nue->info=\*d;

nue->sig=NULL;

while(\*p)

{

if(p->info.clave==nue->info.clave)

{

p->info.cantrep++;

free(nue);

return 1;

}

p=&(\*p)->sig;

}

\*p=nue;

return 1;

}

**int insertarenlistasinohayclavetrenchi(t\_lista\* p,t\_info\* d)**

{

while(\*p&&(\*p)->info.clave!=clave)

p=&(\*p)->sig;

if(\*p)

{

((\*p)->info.cantrep)++;

return 1;

}

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

if(\*p==NULL)

return 0;

(\*p)->info.clave=clave;

(\*p)->info.cantrep=0;

(\*p)->sig=NULL;

return 1;

}

**int insertarordenadoactualizar(t\_lista \*p,const t\_info \*d, int(\*cmp)(const t\_info\*,const t\_info\*),void(\*ac)(t\_info\*))**

{

t\_nodo\* nue;

while(\*p&&cmp(&(\*p)->info,d)<=0)

p=&(\*p)\_>sig;

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

{

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

return TODO\_OK;

}

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

if(!nue)

return SIN\_MEMORIA;

nue->info=\*d;

nue->sig=\*p;

\*p=nue;

return TODO\_OK; }

**int eliminartodaslasrepeticionesdeunalistadesordenado(t\_lista\* p,const t\_info\* info)**

{

t\_nodo\* nodo;

int cantel=0;

while(\*p)

{

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

{

cantel++;

aux=\*p;

\*p=aux->sig;

free(aux);

}

else

p=&(\*p)->sig;

}

return cantel;

}