```
/**//*            en los siguientes macroreemplazos indique:                */**/
/**//*    su(s)         APELLIDO(S)     completo(s)                          */**/
/**//*    su(s)         Nombre(S)       completo(s)                          */**/
/**//*    su legajo     NÚMERO DE DNI   con los puntos de millón y de mil */**/
/**//*    COMISIÓN                                                           */**/
/**//*              reemplazando los que están como ejemplo                  */**/
#define APELLIDO    "Gutierrez"
#define NOMBRE      "Damian, Edgardo"
#define DOCUMENTO   "36.902.267"
#define COMISION    " 02(3362)"
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
/** aquí insertaremos nuestras observaciones y / o correcciones              **/

/**//**/
/** RESULTADO EVALUACIÓN

NOTA: 4
ESTADO: APROBADO


"El formato del zip no es correcto. Con qué programa lo genera???
REA3 NO COMPILA falta ""}"" Elimina mal, elimina un nodo de más de la clave que debe
quedar. Funciona desde nMejores = 3 en adelante

REA3 JAVA
Esta bien resuelto

REA3 C++
- Los metodos los hace publicos como absoluto,mcd.."
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
#undef APELLIDO
#undef NOMBRE
#undef DOCUMENTO
#undef COMISION
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
/**//**//* CUALQUIER INCLUDE DE BIBLIOTECA QUE NECESITE, HÁGALO DESDE ACÁ */**/

int compararClave(const void * , const void *);
int compararNota(const void * , const void *);

/**//**//* CUALQUIER INCLUDE DE BIBLIOTECA QUE NECESITE, HÁGALO HASTA ACÁ */**/
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
#include "funciones.h"
#include <stdlib.h>
#include <string.h>

/** para el PUNTO 1 a.- **/


int ponerEnListaDoblNMejores_MIO(tListaDobl * lista, const void *nota,
                                 size_t cantBytes, int tope)
{
    tNodoDobl * actual = *lista;
    tNodoDobl *izquierdo, *derecho = NULL, *ultimo = NULL, *anteultimo;
    int cant=1, cmp, eliminados = 0;
    if(!actual)
        izquierdo=derecho=NULL;
    else
    {
        while(actual->izq)
            actual= actual->izq;
        while(actual->der && (!derecho|| cant<tope))
        {
            if(!derecho&&compararClave(nota,actual->info)>=0)
            {
```

```
                if(compararClave(nota,actual->info)>0)
                {
                    derecho = actual;
                    izquierdo = actual->izq;
                }
                else
                {
                    izquierdo = actual;
                    derecho = actual->der;
                }
            }
            actual = actual->der;
            cant++;
            if(cant==tope)
            {
                ultimo = actual;
                if(compararNota(actual->info,nota)>0)
                    return NO_SE_INSERTA;
            }
        }
        if(ultimo == NULL)
            ultimo = actual;
        anteultimo = ultimo->izq;
        cmp=compararNota(nota, ultimo->info);
        if(cmp < 0 && cant >= tope)
            return NO_SE_INSERTA;
        if(cant >= tope && cmp>0 && anteultimo && compararNota(anteultimo->info,ultimo->
info)!=0)
        {
            anteultimo->der = NULL;
            actual = anteultimo;
            while(ultimo)
            {
                tNodoDobl * auxiliar = ultimo->der;
                free(ultimo->info);
                free(ultimo);
                ultimo = auxiliar;
                eliminados++;
            }
        }
        if(!derecho)
        {
            if(compararClave(nota,actual->info)>0)
            {
                derecho = actual;
                izquierdo = actual->izq;
            }
            else
            {
                izquierdo = actual;
                derecho = actual->der;
            }
        }
    }
    tNodoDobl * nue = (tNodoDobl*)malloc(sizeof(tNodoDobl));
    if(!nue)
        return SIN_MEMORIA;
    nue->info = malloc(cantBytes);
    if(!nue->info)
    {
        free(nue);
        return SIN_MEMORIA;
    }
    memcpy(nue->info, nota, cantBytes);
    nue->tamInfo = cantBytes;
    nue->izq = izquierdo;
```

```
    nue->der = derecho;
    if(izquierdo)
        izquierdo->der = nue;
    if(derecho)
        derecho->izq = nue;
    *lista = nue;
    return eliminados>0?(-1)*eliminados: TODO_BIEN;
}
int compararNota(const void * d1, const void * d2)
{
    tEvalua * n1 = (tEvalua*)d1;
    tEvalua * n2 = (tEvalua*)d2;
    return n1->calif-n2->calif;
}

int compararClave(const void * d1, const void * d2)
{
    tEvalua * n1 = (tEvalua*)d1;
    tEvalua * n2 = (tEvalua*)d2;
    if(n1->calif == n2->calif)
        return strcmp(n2->clave,n1->clave);
    else
        return n1->calif-n2->calif;
}


/** para el PUNTO 1 b.- **/
int ponerEnListaSimpNMejores_MIO(tListaSimp *lista, const void *nota,
                                 size_t cantBytes, int tope)

{
    tListaSimp * act = lista,
                *nodoUlti = lista,
                 *nodoAnteUlt;
    int cantNodos=1,
        comp,
        cantEliminados =0;
    if(*act)
    {
        while(*nodoUlti && (*nodoUlti)->sig && cantNodos<tope)
        {

            nodoAnteUlt = nodoUlti;
            nodoUlti = &(*nodoUlti )->sig;
            cantNodos++;
            if(compararClave(nota,(*nodoUlti)->info)<0)
                act=nodoUlti;
        }
        if(!*nodoUlti)
        {
            nodoAnteUlt = NULL;
        }

        comp=compararNota(nota, (*nodoUlti)->info);
        if(cantNodos >= tope && comp < 0)
            return NO_SE_INSERTA;
        if(comp > 0 && *nodoAnteUlt && *nodoUlti &&
                compararNota((*nodoAnteUlt)->info,(*nodoUlti)->info)!=0)
        {
            while(*nodoUlti)
            {
                tNodoSimp * aux = *nodoUlti;
                *nodoUlti = aux->sig;
                free(aux->info);
                free(aux);
                cantEliminados++;
```

```
            }
        }
    }
    while(*act && compararClave(nota,(*act)->info)<0)
        act = &(*act)->sig;
    tNodoSimp * nue = (tNodoSimp*)malloc(sizeof(tNodoSimp));
    if(!nue)
        return SIN_MEMORIA;
    nue->info = malloc(cantBytes);
    if(!nue->info)
    {
        free(nue);
        return SIN_MEMORIA;
    }
    memcpy(nue->info, nota, cantBytes);
    nue->tamInfo = cantBytes;
    nue->sig = *act;
    *act = nue;
    return cantEliminados>0?(-1)*cantEliminados: TODO_BIEN;
}
/**                  FIN de PUNTO 1 **/
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
#ifndef QUE_HICE_H_
#define QUE_HICE_H_


#define CODIGO_ALUMNO

#ifdef CODIGO_ALUMNO
#define PUNTO_1_A
#define PUNTO_1_B
#endif // CODIGO_ALUMNO

#endif // QUE_HICE_H_
//////////////////
#include "../include/Cuerdas.h"

Cuerdas::Cuerdas(string nomb, char esc) : Instrumento(nomb,esc)
{}

string Cuerdas::getDetalles() const
{
    return "";
}

Cuerdas& Cuerdas::operator++()
{
    this->escala++;
    this->ajustarEscala();
    return *this;
}

Cuerdas Cuerdas::operator++(int)
{
    Cuerdas aux(*this);

    this->escala++;
    this->ajustarEscala();

    return aux;
}

/// complete el desarrollo de los métodos de la class

//////////////
#include <string.h>
```

```cpp
#include "../include/Percusion.h"
#include "../include/CambioEscalaException.h"

Percusion::Percusion(string nomb)
    : Instrumento(nomb,'X')
{}

string Percusion::getDetalles() const
{
    return "";
}

Percusion& Percusion::operator--()
{
    throw CambioEscalaException();
}
Percusion Percusion::operator--(int)
{
    throw CambioEscalaException();
}
Percusion& Percusion::operator++()
{
    throw CambioEscalaException();
}
Percusion Percusion::operator++(int)
{
    throw CambioEscalaException();
}

/// complete el desarrollo de los métodos de la class

//////////////////
#include "../include/Vientos.h"

Vientos::Vientos(string nomb, char esc, string det)
: Instrumento(nomb,esc), detalle(det)
{}

string Vientos::getDetalles() const
{
    return "material:" + this->detalle;
}

Vientos& Vientos::operator--()
{
    this->escala--;
    this->ajustarEscala();
    return *this;
}

Vientos Vientos::operator--(int)
{
    Vientos aux(*this);

    this->escala--;
    this->ajustarEscala();

    return aux;
}


/// complete el desarrollo de los métodos de la class

///////////////
#ifndef CAMBIOESCALAEXCEPTION_H_INCLUDED
#define CAMBIOESCALAEXCEPTION_H_INCLUDED
```

```cpp
#include <iostream>
using namespace std;

class CambioEscalaException : public exception
{
public:
    CambioEscalaException(){};
    string what()
    {
        return "No permite cambio de escala";
    }
};
/// complete la declaración de la class

#endif // CAMBIOESCALAEXCEPTION_H_INCLUDED
//////////////////
#ifndef CUERDAS_H_INCLUDED
#define CUERDAS_H_INCLUDED


#include "Instrumento.h"

class Cuerdas : public Instrumento
{
public:
    Cuerdas(string nomb, char esc);
    ~Cuerdas(){};
    string getDetalles() const;
    Cuerdas& operator++(); //devuelve el obj modificado;
    Cuerdas operator++(int); //devuelve copia;
};
/// complete la declaración de la class

#endif // CUERDAS_H_INCLUDED

////////////////
#ifndef PERCUSION_H_INCLUDED
#define PERCUSION_H_INCLUDED


#include "Instrumento.h"

class Percusion : public Instrumento
{
public:
    Percusion(string nomb);
    string getDetalles() const;
    ~Percusion(){};
    Percusion& operator--();
    Percusion operator--(int);
    Percusion& operator++();
    Percusion operator++(int);
};

/// complete la declaración de la class

#endif // PERCUSION_H_INCLUDED
//////////////////
#ifndef VIENTOS_H_INCLUDED
#define VIENTOS_H_INCLUDED


#include "Instrumento.h"

class Vientos : public Instrumento
{
```

```
private:
    string detalle;
public:
    Vientos(string nomb, char esc, string det);
    ~Vientos(){};
    string getDetalles() const;
    Vientos& operator--(); //devuelve el obj modificado;
    Vientos operator--(int);
};
/// complete la declaración de la class

#endif // VIENTOS_H_INCLUDED
//////////////////
#include "../include/Instrumento.h"
#include "../include/CambioEscalaException.h"

Instrumento::Instrumento(string nomb, char esc)
{
    this->nombre = nomb;
    this->escala = esc;
}

string Instrumento::getNombre() const
{
    return this->nombre;
}

char Instrumento::getEscala() const
{
    return this->escala;
}

void Instrumento::ajustarEscala()
{
    if(escala>'G')
        escala = 'A';
    if(escala<'A')
        escala = 'G';
}

/// complete el desarrollo de los métodos de la class

//////////////////
#ifndef INSTRUMENTO_H_INCLUDED
#define INSTRUMENTO_H_INCLUDED

#include <iostream>
#include <string>
using namespace std;


/* escalas musicales:
DO  RE  MI  FA  SOL LA  SI
C   D   E   F   G   A   B
*/

class Instrumento
{
protected:
    string nombre;
    char escala;
    void ajustarEscala();
public:
    Instrumento(string nomb, char esc);
    virtual ~Instrumento(){};
    string getNombre() const;
```

```
    char getEscala() const;
    virtual string getDetalles() const = 0;
};

/// complete la declaración de la class

#endif // INSTRUMENTO_H_INCLUDED
////

//////////////
REA3
//////////////
C++
//////////////////////

#include "../include/clase.h"
#include "../include/Div0exception.h"

int Racional::absoluto(int n)
{
    if(signo(n))
        n = -n;
    return n;
}

char Racional::signo(const int n)
{
    return n < 0 ? '-' : '\0';
}

int Racional::mcd(int a, int b)
{
//    if(a==0)
//        return b;
//
//    if(a<b)
//    {
//        int aux;
//        aux=a;
//        a=b;
//        b=aux;
//    }

    int resto = a % b;


    while(resto)
    {
        a = b;
        b = resto;
        resto = a % b;
    }
    return absoluto(b);
}

Racional::Racional(int nume, int deno)
{
    try
    {
        if(!deno)
            throw Div0exception();
    }
    catch(Div0exception &div0)
    {
        cout << "Excepcion: " << div0.what() << endl;
        _Exit(1);
```

```
    }
        if(signo(deno))
        {
            nume = -nume;
            deno = -deno;
        }

        int divisor = mcd(nume, deno);
        this->nume = nume / divisor;
        this->deno = deno / divisor;
}

ostream &operator <<(ostream &salida, Racional racio)
{
    cout << racio.nume << '/' << racio.deno;
    return salida;
}

int Racional::getNumerador() const
{
    return nume;
}

int Racional::getDenominador() const
{
    return deno;
}

int Racional::enteroMasCercano()
{
    return float(nume) / deno + (nume<0?-.5:.5);
}

void Racional::aEnteroYFraccion(const Racional r)
{

    int     enteros = r.nume / r.deno;
    cout << enteros << " " << r.nume % r.deno << '/' << r.deno;
}

Racional Racional::operator++()
{
    nume += deno;
    return *this;
}

////////
#ifndef CLASE_H_INCLUDED
#define CLASE_H_INCLUDED

#include <iostream>
#include <stdlib.h>


using namespace std;

class Racional
{
private:
    int     nume,
            deno;
    int absoluto(int);
    int mcd(int, int);
    char signo(const int);
friend ostream &operator <<(ostream &, Racional);
public :
```

```
    Racional(int = 1, int = 1);
    int getNumerador() const;
    int getDenominador()const;
    int enteroMasCercano();
    void aEnteroYFraccion(Racional);
    Racional operator++();
};

#endif // CLASE_H_INCLUDED
//////////////
#ifndef QUE_HICE_H_
#define QUE_HICE_H_

#define CODIGO_ALUMNO

#ifdef CODIGO_ALUMNO
#define PUNTO_1_C
#endif // CODIGO_ALUMNO

#endif // QUE_HICE_H_
//////////////
/**//*             en los siguientes macroreemplazos indique:             *//**/
/**//*    su(s)         APELLIDO(S)     completo(s)                       *//**/
/**//*    su(s)         Nombre(S)       completo(s)                       *//**/
/**//*    su legajo     NÚMERO DE DNI   con los puntos de millón y de mil *//**/
/**//*    COMISIÓN                                                        *//**/
/**//*              reemplazando los que están como ejemplo               *//**/
#define APELLIDO    "Gutierrez"
#define NOMBRE      "Edgardo Damian"
#define DOCUMENTO   "36.902.267"
#define COMISION    "02(3362)"
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
/** aquí insertaremos nuestras observaciones y / o correcciones              **/
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
#undef APELLIDO
#undef NOMBRE
#undef DOCUMENTO
#undef COMISION
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
/**//**//* CUALQUIER INCLUDE DE BIBLIOTECA QUE NECESITE, HÁGALO DESDE ACÁ *//**/



/**//**//* CUALQUIER INCLUDE DE BIBLIOTECA QUE NECESITE, HÁGALO HASTA ACÁ *//**/
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
#include "funciones.h"
#include <stdlib.h>
#include <string.h>
int compararClave(const void * d1, const void * d2);
int compararNota(const void * d1, const void * d2);

/** para el PUNTO 1 c.- **/
int compararNota(const void * d1, const void * d2)
{
    tEvalua * n1 = (tEvalua*)d1;
    tEvalua * n2 = (tEvalua*)d2;
    return n1->calif-n2->calif;
}

int compararClave(const void * d1, const void * d2)
{
    tEvalua * n1 = (tEvalua*)d1;
    tEvalua * n2 = (tEvalua*)d2;
    if(n1->calif == n2->calif)
        return strcmp(n2->clave,n1->clave);
```

```
    else
        return n1->calif-n2->calif;
}

int ponerEnListaDoblNCalif_MIO(tListaDobl *lista, const void *nota,
                               size_t cantBytes, int tope)
{
       /// printf("PEPE109\n");
      tNodoDobl * actual = *lista;
    tNodoDobl *izquierdo, *derecho = NULL, *ultimo = NULL, *anteultimo, *pBorrado;
    int cant=1, cmp, eliminados = 0, cambiosVal=0, marcaCoincide=0;
    if(!actual)
    {
        izquierdo=derecho=NULL;
    }
    else
    {
        while(actual->izq)
            actual= actual->izq;
        while(actual->der && (!derecho|| cambiosVal < tope))
        {
            if(compararNota(actual->info, actual->der->info ) != 0)
                    {
                        cambiosVal++;
                        pBorrado = actual;

                    }
            if(compararNota(nota, (actual)->info) == 0)
                marcaCoincide++;
            if(!derecho&&compararClave(nota,actual->info)>=0)
            {
                if(compararClave(nota,actual->info)>0)
                {
                    derecho = actual;
                    izquierdo = actual->izq;
                }
                else
                {
                    izquierdo = actual;
                    derecho = actual->der;
                }
            }

            actual = actual->der;
            cant++;
            if(marcaCoincide == tope-1)
            {
                ultimo = actual;
                if(compararNota(actual->info,nota) > 0)
                    {
                        return NO_SE_INSERTA;

                    }
            }
        }
        if(ultimo == NULL)
        {
            ultimo = actual;
        }
        anteultimo = ultimo->izq;

        cmp=compararNota(nota, ultimo->info);
        if(cmp < 0 && cant >= tope && cambiosVal == tope-1 )
        {
            printf("Borrando: %s \n", ((tEvalua*)pBorrado->info)->clave);
            return NO_SE_INSERTA;
```

```
        }
        else
        {
            if(cant >= tope && cmp>0 && anteultimo &&
               cambiosVal == tope-1 && marcaCoincide==0)
            {
                ultimo = pBorrado;
                anteultimo = pBorrado->izq;
                anteultimo->der = NULL;
                while(ultimo)
                {
                    ///printf("Borrando: %s\n", ((tEvalua*)actual->info)->clave);
                    tNodoDobl * auxiliar = ultimo->der;
                    free(ultimo->info);
                    free(ultimo);
                    ultimo = auxiliar;
                    eliminados++;
                }
            }
        }
        if(!derecho)
        {
            if(compararClave(nota,actual->info)>0)
            {
                derecho = actual;
                izquierdo = actual->izq;
            }
            else
            {
                izquierdo = actual;
                derecho = actual->der;
            }
        }
    }

    tNodoDobl * nue = (tNodoDobl*)malloc(sizeof(tNodoDobl));
    if(!nue)
        return SIN_MEMORIA;
    nue->info = malloc(cantBytes);
    if(!nue->info)
    {
        free(nue);
        return SIN_MEMORIA;
    }

    memcpy(nue->info, nota, cantBytes);
    nue->tamInfo = cantBytes;
    nue->izq = izquierdo;
    nue->der = derecho;
    if(izquierdo)
        izquierdo->der = nue;

    if(derecho)
        derecho->izq = nue;
    *lista = nue;
    return eliminados>0?(-1)*eliminados: TODO_BIEN;

/**                  FIN de PUNTO 1 **/
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
```