```
/*Tema 4.- Clases en c++(ampliacion)

Introduccion
Tipo de datos abstracto(diseño): Coleccion de datos (heterogeneos) y un conjunto de
operaciones definidos mediante una especificacion independiente de cualquier lenguaje de
programacion(nmo hay detalles de implementacion)
TDA Celda:
    Datos:
        - valor (double)
        - enlace a siguiente celda
    Operaciones
        - acceso a los datos miembro
        - constuctor
        - ...

TDA Lista:
    Datos:
        - longitud
        - enlace a primera Celda
    Operaciones:
        - acceder a un elemento
        - asignar valor a un elemento
        - concatenar
        - ...


Tema5.- Funciones y clases amigas (friend)
- OJO: Rompe la encapsulacion

Class claseA{
    int xalloc
    public:
    ....
    friend class ClaseB; //Puede acceder a los datos miembro
    friend void funcion; //Puede acceder a los datos miembros

}


Tema 6.- Mejoras del uso de la clase.
    - Reduccion del numero de constructores
    - Libreacion de espacio: Clases con memoria dinamica ->destructor;

    Lista(double valor,int longitud)
    lista()
        Ambas pueden resumirse e un solo constructores
            Lista(double valor=0;int longitud=0);


Tema 7.- El destructor
    - Metood unico
    - Sin parametros
    - No devuelve nada
    - Se lanza automaticamente
        * Si son objetos locales: Se llama al finalizar la funcion o metodo de creacion
        * Si son objetos globales se llaman al finalizar el programa
    - Imprescindible si hay memoria dinamica

Tema 8.- Constructor de copia
    - Necesario si hay memoria deincamica (hay uno por defecto pero no va bien en estos casos)
    - Si no se cambia compartirian la memoria

    -Cuando se llama:
        * Paso por valor
        * Explicita
        * Devolucion de objeto ¿?


*/

/*-----------------------------------------------
-----------------Class Celda--------------------
-----------------------------------------------*/
```

```
class Celda{
    private:
    double info;
    Celda *sig;
    public:
    Celda(double info);
    Celda * obtenerSiguiente();
    double obtenerValor();
    void asignarSiguiente(Celda *sig);
    void asignarValor(double valor);
    void imprimir();
};

Celda::Celda(double valor){
    info=valor;
    sig=0;
}

Celda * Celda::obtenerSiguiente(){
    return sig;
}

double Celda::obtenerValor(){
    return valor;
}

void Celda::asignarValor(double valor){
    this->valor=valor;
}

void Celda::asignarSiguiente(Celda *sig){
    this->sig=sig;
}

void Celda::imprimir(){
    cout << this->valor<<" ";
}




/*-----------------------------------------------
----------------Class Lista--------------------
-----------------------------------------------*/
class lista{
    private:
    celda* contenido;
    int longitud;
    Celda * reservarCeldas(int numero,double valor,int incremento);
    Celda * obtenerCelda(int indice)const;
    public:
    Lista();
    ~Lista();
    Lista(const lista & otra);
    Lista(double valor, itn longitud);
    void eliminarFinal();
    void agregarFinal(double valor);
    void imprimir();
    double obtenerElemento(int indice)const;
    void asignarElemento(int indice,double valor)const;
    bool borrarElemento(int indice);
    void insertarElemento(int indice, double valor);


};

//***********************
//Metodos privado
//***********************
Celda * Lista::reservarCeldas(int numero,double valor,int incremento=1){
    Celda *pCelda,*pPrimera,*pCeldaAnterior=0;
    for(int i=0;i<numero;i++){
        pCelda=new Celda(valor+i*incremento);
```

```
        if(pAnterior!=0)
            pPrimera=pCelda;
        else
            pCeldaAnterior->asignarSiguiente(pCelda);

        pCeldaAnterior=pCelda;
    }
    return pPrimera;
}

Celda * Lista::obtenerCelda(int indice)const{
    Celda *pCelda=0;
    if(conteidoi!=0&&indice>=0&&indice<longitud){
    pCelda=contenido;
        for(int i=0;i<indice;i++){
            pCelda=pCelda->obtenerSiguiente();
        }
    }
    return pCelda;
}

//***********************
//Metodos privado
//***********************

Lista::~Lista(){
    Celda*pCelda=contenido,*pSiguiente;
    while(pCelda!=0){
        pSiguiente=pCelda->obtenerSiguiente();
        delete pCelda;
        pCelda=pSiguiente;
    }
    longitud=0;
    contenido=0;
}

Lista::Lista(const lista & otra){
    conteido=reservarCeldas(otra.longitud,0);
    celda *pcelda=contenido;
    for(int i=0;i<otra.longitud;i++){
        pCelda->asignarValor(otra.obtenerElemento(i));
        pCelda=pCelda->obtenerSiguiente();
    }
    longitud=otra.longitud;
}
Lista::Lista(){
    contenido=0;
    longitud=0;
}

Lista::Lista(double valor, int longitud){
    this->longitud=longitud;
    contenido=0;

    if(longitud!=0){
        contenido=reservarCeldas(longitud,valor);
    }
}

Lista::Lista(int desde, int hasta, int incremento=1){
    this->longitud=(hasta-desde)/incremento+1;
    contenido=0;

    if(longitud!=0){
        contenido=reservarCeldas(longitud,desde,incremento);
    }
}

void Lista::agregarFinal(double valor){
    Celda *pCelda;

    //Si la lista no esta vacia
    if(contenido!=0){
```

```
        pCelda=contenido;
        while(pCelda->obtenerSiguiente()!=0){
            pCelda=pCelda->obtenerSiguiente();
        }
        pCelda->asignarSiguiente(valor);
    }
    else{
        contenido=new Celda(valor);
    }
    longitud++;
}

void Lista::imprimir(){
    Celda * pCelda;
    pCelda=contenido;

    while(pCelda!=0){
        pCelda->imprimir();
        pCelda=pCelda->obtenerSiguiente();
    }
}

void Lista::eliminarFinal(){
    Celda * pCelda,*penultimo,*ultimo;
    if(contenido!=0){
        pCelda=penultimo=ultimo=contenido;
        while(pCelda->obtenerSiguiente()!=0){
            if(pCelda->obtenerSiguiente()->obtenerSiguiente()!=0)
                penultimo=pCelda->obtenerSiguiente();
            pCelda=pCelda->obtenerSiguiente();
        }
        ultimo=penultimo->obtenerSiguiente();
        penultimo->asignarSiguiente(0);
        if(ultimo!=0)
            delete ultimo;
        if(penultimo==contenido)
            contenido=0;
    longitud--;
    }

}

double Lista::obtenerElemento(int indice)const{
    Celda *pCelda=obtenerCelda(indice);
    return pCelda->obtenerValor();
}


void Lista::asignarElemento(int indice,double valor)const{
    Celda *pCelda=obtenerCelda(indice);
    pCelda->asignarValor(valor);
}

bool Lista::borrarElemento(int indice){
    bool resultado=false;
    Celda * pCelda = obtenerCelda(indice);
    if(pCelda!=0){
        if(pCelda==contenido){
            contenido=pCelda->obtenerSiguiente();
        }
        else{
            Celda * previa=obtenerCelda(indice-1);
            previa->asignarSiguiente(pCelda->obtenerSiguiente());
        }
        delete pCelda;
        reultado=true;
        longitud--;
    }
    return resultado;
}

void Lista::insertarElemento(int indice, double valor){
    Celda *pCelda=obtenerCelda(indice);
```

```
    if(pCelda!=0){
        Celda *ncelda=new Celda(valor);
        if(pCelda==contenido){
            contenido->asignarSiguiente(ncelda);
            ncelda->asignarSiguiente(pCelda);
        }
        else{
            Celda *previa=obtenerCelda(indice-1);
            previa->asignarSiguiente(ncelda);
            ncelda->asignarSiguiente(pCelda);
        }
    }
}


void Lista::concatenar(cont lista &otra){
    Celda *pCelda = reservarCeldas(otra.longitud,0);
    Celda *pPrimera=pCelda;
    for(int i=0;i<otra.longitud;i++){
        pPrimera->asignarValor(otra.obtenerElemento(i));
        pPrimera=pPrimera->obtenerSiguiente();
    }
    Celda *pultima=obtenerCelda(this.longitud-1);
    pultima->asignarSiguiente(pCelda);
    this.longitud+=otra.longitud;
}
```