**/\*-------------------- NHC - PRACTICA 8 ---------------------\*/**

**CLista.h**

#if !defined (\_LISTA\_H)

#define \_LISTA\_H

#include "CNodoLista.h"

// Declaración anticipada de CNodoLista:

template <class T> class CNodoLista;

template <class T> class CLista //Declaración

{

private:

CNodoLista<T> \* m\_Primero;

CNodoLista<T> \* m\_Actual;

CNodoLista<T> \* m\_Ultimo;

public:

CLista();

CLista(const T& Obj);

CLista(const CLista<T>& Lista);

~CLista();

CLista<T>& operator=(const CLista<T>& Lista);

CNodoLista<T>\* operator[](int i) const;

bool EstaVacia() const { return m\_Primero == NULL; }

bool TieneMas() const { return m\_Actual != NULL; }

void AgregarObjeto(const T& Obj);

void Vaciar();

T& GetPrimero() const;

T& GetActual() const;

};

#include "CLista.cpp" //Definición

#endif

**CLista.cpp**

#if !defined(\_LISTA\_CPP)

#define \_LISTA\_CPP

#include "CLista.h"

#include "MemoryManager.h"

#include "CIndiceIncorrecto.h"

#include <iostream>

;using namespace std;

template<class T> CLista<T>::CLista() //Contructor de plantilla CLista

{

m\_Primero = NULL;

m\_Actual = NULL;

m\_Ultimo = NULL;

}

template<class T> CLista<T>::CLista(const T& Obj)

//Constructor de plantilla en base a un objeto

{

m\_Primero=NULL;

m\_Actual=NULL;

m\_Ultimo=NULL;

AgregarObjeto(Obj);

}

template <class T> CLista<T>::CLista(const CLista<T> &Lista)

//Constructor copia de plantilla CLista

{

m\_Primero = NULL;

m\_Actual = NULL;

m\_Ultimo = NULL;

\*this = Lista;

}

template <class T> CLista<T>& CLista<T>::operator=(const CLista<T>& Lista)

//Operator= de plantilla CLista

{

Vaciar();

if(!Lista.EstaVacia())

{

m\_Actual = m\_Ultimo = m\_Primero = new CNodoLista<T>(Lista.GetPrimero());

while(Lista.TieneMas())

{

m\_Actual->SetSigNodo(new CNodoLista<T>(Lista.GetActual()));

m\_Ultimo = m\_Actual = m\_Actual->GetSigNodo();

}

}

return \*this;

}

template <class T> void CLista<T>::AgregarObjeto(const T& Obj)

//Miembro AgregarObjeto de plantilla CLista

{

if(EstaVacia()) //Si la lista esta vacia(No existe)

m\_Primero = m\_Actual = m\_Ultimo = new CNodoLista<T>(Obj);

//Reserva memoria para las variables que la determinan

else

{

m\_Ultimo->SetSigNodo(new CNodoLista<T>(Obj));

//Si no, se reserva memoria para el siguiente nodo al que apunta m\_Ultimo

m\_Ultimo = m\_Ultimo->GetSigNodo();

//y m\_Ultimo pasa a ser el siguiente nodo al que apuntaba

}

}

template <class T> T& CLista<T>::GetActual() const //Miembro GetActual de plantilla CLista

{

CNodoLista<T> \*temp = m\_Actual;

const\_cast<CLista<T>\*>(this)->m\_Actual = m\_Actual->GetSigNodo();

return temp->GetDato();

}

template <class T> T& CLista<T>::GetPrimero() const

{

// PREGUNTA 11

const\_cast<CLista<T>\*>(this)->m\_Actual = m\_Primero->GetSigNodo();

return m\_Primero->GetDato();

}

// PREGUNTA 2: operator[] de CLista. Lanza CIndiceIncorrecto.

template <class T> CNodoLista<T>\* CLista<T>::operator[](int i) const

{

if(i<0) throw CIndiceIncorrecto::CIndiceIncorrecto();

if(i==0) return m\_Primero;

GetPrimero();

for(int j=1;(j<i)&&(m\_Actual!=NULL);j++)

{

GetActual();

}

if(m\_Actual == NULL) throw CIndiceIncorrecto::CIndiceIncorrecto();

return m\_Actual;

}

// PREGUNTA 1: CLista<T>::Vaciar

template <class T> void CLista<T>::Vaciar() //Miembro Vaciar de CLista

{

if(!EstaVacia())

{

CNodoLista<T> \*actual = NULL;

GetPrimero();

while(m\_Actual!=NULL)

{

actual=m\_Actual;

GetActual();

delete actual;

}

delete m\_Primero;

/\*--\*/if(m\_Ultimo == NULL) delete m\_Ultimo;

m\_Ultimo=NULL;

m\_Primero=NULL;

m\_Actual=NULL;

}

}

template<class T> CLista<T>::~CLista() //Destructor de platilla CLista

{

Vaciar();

}

#endif

**CNodoLista.h**

#if !defined(\_NODOLISTA\_H)

#define \_NODOLISTA\_H

#include "CLista.h"

// Declaración anticipada de CLista:

template<class T> class CLista;

template<class T>

class CNodoLista //Declaración

{

friend CLista<T>;

private:

T m\_Dato;

CNodoLista<T> \* m\_pSigNodo;

public:

CNodoLista() { m\_pSigNodo = NULL; }

CNodoLista(const CNodoLista<T>& NodoLista){\*this = NodoLista};

CNodoLista(const T& Obj, CNodoLista<T>\* pNodo = NULL): m\_Dato(Obj), m\_pSigNodo(pNodo){};

CNodoLista<T>& operator=(const CNodoLista<T>& NodoLista){};

CNodoLista<T> \*GetSigNodo() const { return m\_pSigNodo; }

T& GetDato() { return m\_Dato; }

void SetSigNodo(CNodoLista<T> \*pNodo) { m\_pSigNodo = pNodo; }

};

#include "CNodoLista.cpp" //Definición

#endif

**CSiniestro.h**

#if !defined(\_SINIESTRO\_H)

#define \_SINIESTRO\_H

#include <iostream>

#include <string>

class CSiniestro

{

private:

int m\_Codigo;

static int m\_SigCodigo;

protected:

std::string m\_Descripcion;

float m\_HorasMO;

float m\_CostePiezas;

float m\_Coste; // PREGUNTA 7

public:

CSiniestro(const std::string& Desc = "Sin Descripcion");

virtual ~CSiniestro() {};

virtual float GetPresupuesto() const = 0;

int GetCodigo() const { return m\_Codigo; }

float GetCoste() const { return m\_Coste; } // PREGUNTA 7

virtual void Presupuestar(float Horas, float Piezas) = 0;

virtual void Mostrar(std::ostream& os = std::cout) const; // PREGUNTA 12

virtual CSiniestro\* Clonar() const = 0;

};

#endif

**CSiniestro.cpp**

#include "CSiniestro.h"

#include "CUtils.h"

using namespace std;

using namespace utils;

// PREGUNTA 6: iniciar m\_SigCodigo

int CSiniestro::m\_SigCodigo = 1;

CSiniestro::CSiniestro(const std::string& Desc)

{

cout<<"Introduzca el codigo: ";

utils::CUtils::leerInt(m\_Codigo);

cout<<"Introduzca el numero de horas de mano de obra: ";

utils::CUtils::leerFloat(m\_HorasMO);

cout<<"Introduzca el coste de las piezas: ";

utils::CUtils::leerFloat(m\_CostePiezas);

cout<<"Introduzca el numero de piezas: ";

utils::CUtils::leerFloat(m\_Coste);

m\_Descripcion = Desc;

}

// PREGUNTA 10: CSiniestroNormal s("Rotura de tambor");

// PREGUNTA 12: GetCoste

void CSiniestro::Mostrar(ostream &os) const

{

os << " Codigo del siniestro: " << m\_Codigo << endl

<< " Descripcion de la averia: " << m\_Descripcion << endl

<< " Horas de mano de obra: " << m\_HorasMO << endl

<< " Coste de las piezas: " << m\_CostePiezas << endl

<< " Coste total de reparacion: " << m\_Coste << endl

<< endl;

}

**CSiniestroNormal.h**

#if !defined(\_SINIESTRONORMAL\_H)

#define \_SINIESTRONORMAL\_H

#include "CSiniestro.h"

class CSiniestroNormal : public CSiniestro

{

private:

static float m\_CosteHoraMO;

public:

CSiniestroNormal (const std::string& Desc = "Sin Descripcion");

void Presupuestar(float Horas = 0.5f, float Piezas = 0);

void Mostrar(std::ostream& os = std::cout) const;

CSiniestroNormal\* Clonar() const;

float GetPresupuesto() const;

};

#endif

**CSiniestroNormal.cpp**

#include "CSiniestroNormal.h"

//#include "MemoryManager.h"

using namespace std;

float CSiniestroNormal::m\_CosteHoraMO = 2.5;

CSiniestroNormal::CSiniestroNormal (const std::string& Desc)

:CSiniestro(Desc)

{

}

// PREGUNTA 10

CSiniestroNormal\* CSiniestroNormal::Clonar() const

{

return new CSiniestroNormal(\*this);

}

void CSiniestroNormal::Mostrar(ostream &os) const

{

CSiniestro::Mostrar(os);

os << " Coste de la hora de mano de obra: " << m\_CosteHoraMO << endl;

}

// PREGUNTA 7

void CSiniestroNormal::Presupuestar(float Horas, float Piezas)

{

m\_HorasMO = Horas;

m\_CostePiezas = Piezas;

m\_Coste = m\_HorasMO \* m\_CosteHoraMO + m\_CostePiezas;

}

float CSiniestroNormal::GetPresupuesto() const

{

return m\_Coste;

}

**CSiniestroUrgente.h**

#if !defined(\_SINIESTROURGENTE\_H)

#define \_SINIESTROURGENTE\_H

#include "CSiniestro.h"

enum TSituacion

{

local, nacional, internacional

};

class CSiniestroUrgente : public CSiniestro

{

private:

TSituacion m\_Situacion;

static float m\_Recargo;

static float m\_CosteHoraMO;

public:

CSiniestroUrgente(TSituacion s, const std::string& Desc = "Sin Descripcion");

void Presupuestar(float Horas = 0.5f, float Piezas = 0);

void Mostrar(std::ostream & os = std::cout) const;

CSiniestroUrgente\* Clonar() const;

float GetPresupuesto() const;

};

#endif

**CSiniestroUrgente.cpp**

#include "CSiniestroUrgente.h"

#include "MemoryManager.h"

using namespace std;

float CSiniestroUrgente::m\_Recargo = 5;

float CSiniestroUrgente::m\_CosteHoraMO = 3.5;

CSiniestroUrgente::CSiniestroUrgente(TSituacion s, const std::string& Desc)

: CSiniestro(Desc), m\_Situacion(s)

{

}

CSiniestroUrgente\* CSiniestroUrgente::Clonar() const

{

return new CSiniestroUrgente(\*this);

}

void CSiniestroUrgente::Mostrar(ostream &os) const

{

CSiniestro::Mostrar(os);

os << " Tipo de cobertura: " << m\_Situacion << endl

<< " Recargo por urgencia: " << m\_Recargo << endl

<< " Coste hora de mano de obra: " << m\_CosteHoraMO << endl;

}

// PREGUNTA 7

void CSiniestroUrgente::Presupuestar(float Horas, float Piezas)

{

m\_HorasMO = Horas;

m\_CostePiezas = Piezas;

m\_Coste = m\_CosteHoraMO \* m\_HorasMO + m\_CostePiezas + m\_Recargo;

}

float CSiniestroUrgente::GetPresupuesto() const

{

return m\_Coste;

}

**CCliente.h**

#if !defined(\_CLIENTE\_H)

#define \_CLIENTE\_H

#include <string>

#include <iostream>

#include "CLista.h"

#include "CContrato.h"

class CCliente

{

friend ostream& operator<<(std::ostream& os, const CCliente& cli);

friend void operator+=(long &suma, const CCliente& seg);

private:

std::string m\_Nombre;

CLista<CContrato> m\_Contratos;

public:

CCliente(const std::string& Nom="Sin Nombre") : m\_Nombre(Nom) {};

CCliente(const std::string& Nom, const CContrato& c);

void SetNombre(const std::string& Nom) {m\_Nombre = Nom; }

std::string GetNombre() const { return m\_Nombre; }

void AgregarContrato(const CContrato& c);

CLista<CContrato> GetContratos() const { return m\_Contratos; }

};

#endif

**CCliente.cpp**

#include "CCliente.h"

#include <iostream>

using namespace std;

// PREGUNTA 8: AgregarContrato de CCliente

void CCliente::AgregarContrato(const CContrato& c)

{

m\_Contratos.AgregarObjeto(c);

}

// PREGUNTA 5: operator<< para resolver: cout << unCliente

ostream& operator<<(ostream& os, const CCliente& cli)

{

CLista<CContrato> contrato;

contrato = cli.GetContratos();

cout << "Nombre del cliente: "<< cli.GetNombre() << endl;

if(!contrato.EstaVacia())

{

cout << "Contrato/s: \n"

<< contrato.GetPrimero()

<< endl

<< endl;

while(contrato.TieneMas())

{

cout << contrato.GetActual() << endl

<<endl;

}

}

return os;

}

// PREGUNTA 9: total += seguros[i]

void operator+=(long & suma,const CCliente& seg)

{

long i=0;

try

{

for(i=0;(!seg.m\_Contratos.EstaVacia());i++)

seg.m\_Contratos[i];

}

catch(CIndiceIncorrecto& e)

{

}

suma+=i;

}

**CContrato.h**

#if !defined(\_CONTRATO\_H)

#define \_CONTRATO\_H

#include <iostream>

#include <string>

#include "CSiniestro.h"

#include "CSiniestroUrgente.h"

#include "CSiniestroNormal.h"

#include "CLista.h"

// Declaración adelantada de la plantilla CLista

template<class T> class CLista;

class CContrato

{

friend ostream& operator<<(ostream& os, CContrato& c);

private:

int m\_NumSerie;

std::string m\_Descripcion;

std::string m\_FechaFin;

long m\_Poliza;

long m\_ValorCompra;

CLista<CSiniestro \*> m\_Siniestros;

public:

CContrato(int NumSerie, const std::string& Desc="", const std::string& FF="",

long Poliza = 0, long ValorCompra = 0);

CContrato(const CContrato& c);

CContrato& operator=(const CContrato &c);

~CContrato();

void SetNumSerie(int ns) { m\_NumSerie = ns; }

void SetDescripcion(const std::string& d) { m\_Descripcion = d; }

void SetFechaFin(const std::string& ff) { m\_FechaFin = ff; }

void SetPoliza(long p) { m\_Poliza = p; }

void SetValorCom(long vc) { m\_ValorCompra = vc; }

int GetNumSerie() const { return m\_NumSerie; }

std::string GetDescripcion() const { return m\_Descripcion; }

std::string GetFechaFin() const { return m\_FechaFin; }

long GetPoliza() const { return m\_Poliza; }

long GetValorCom() const { return m\_ValorCompra; }

void AgregarSiniestro(const CSiniestro & s);

void MostrarSiniestros(ostream& os);

void BorrarSiniestros();

};

#endif

**CContrato.cpp**

#include "CContrato.h"

#include "MemoryManager.h"

using namespace std;

CContrato::CContrato(int NumSerie, const string& Desc, const string& FF, long Poliza, long ValorCompra)

: m\_NumSerie(NumSerie), m\_Descripcion(Desc),

m\_FechaFin(FF), m\_Poliza(Poliza), m\_ValorCompra(ValorCompra)

{

}

// PREGUNTA 3: constructor copia de la clase CContrato

CContrato::CContrato(const CContrato& c)

{

int k=0;

CSiniestro \*sin;

m\_NumSerie = c.m\_NumSerie;

m\_Poliza = c.m\_Poliza;

m\_ValorCompra = c.m\_ValorCompra;

m\_Descripcion=c.m\_Descripcion;

m\_FechaFin = c.m\_FechaFin;

try

{

while(1)

{

sin = c.m\_Siniestros[k]->GetDato();

m\_Siniestros.AgregarObjeto(sin->Clonar());

k++;

}

}

catch(CIndiceIncorrecto& i)

{

}

}

// CContrato& CContrato::operator =(const CContrato &c)

CContrato& CContrato::operator=(const CContrato &c)

{

if(!m\_Siniestros.EstaVacia())

BorrarSiniestros();

if(!c.m\_Siniestros.EstaVacia())

{

AgregarSiniestro(\*c.m\_Siniestros.GetPrimero());

while(c.m\_Siniestros.TieneMas())

{

AgregarSiniestro(\*c.m\_Siniestros.GetActual());

}

}

m\_NumSerie=c.m\_NumSerie;

m\_Descripcion=c.m\_Descripcion;

m\_FechaFin=c.m\_FechaFin;

m\_Poliza=c.m\_Poliza;

return \*this;

}

// PREGUNTA 4: AgregarSiniestro de la clase CContrato

void CContrato::AgregarSiniestro(const CSiniestro & s)

{

CSiniestro \* sin = s.Clonar();

m\_Siniestros.AgregarObjeto(sin);

}

CContrato::~CContrato()

{

BorrarSiniestros();

}

// void CContrato::BorrarSiniestros()

void CContrato::BorrarSiniestros()

{

if (!m\_Siniestros.EstaVacia())

{

CSiniestro \* sin = m\_Siniestros.GetPrimero();

delete sin;

sin=NULL;

while(m\_Siniestros.TieneMas())

{

sin = m\_Siniestros.GetActual();

delete sin;

}

m\_Siniestros.Vaciar();

}

}

// void CContrato::MostrarSiniestros(ostream& os)

void CContrato::MostrarSiniestros(ostream& os)

{

if(m\_Siniestros.EstaVacia())

return;

cout << endl

<< "Siniestro:"

<< endl;

m\_Siniestros.GetPrimero()->Mostrar();

while(m\_Siniestros.TieneMas())

{

cout << endl

<< "Siniestro:"

<< endl;

m\_Siniestros.GetActual()->Mostrar();

}

}

// Necesaria para PREGUNTA 5

ostream& operator<<(ostream& os, CContrato& c)

{

cout << "Numero de serie: " << c.GetNumSerie() << endl

<< "Descripcion: "<< c.GetDescripcion() << endl

<< "Fecha Fin: " << c.GetFechaFin() << endl

<< "Poliza: " << c.GetPoliza() << endl

<< "Valor de compra: " << c.GetValorCom() << endl;

c.MostrarSiniestros(cout);

cout << endl;

return os;

}

**CEmpresa.h**

#if !defined(\_EMPRESA\_H)

#define \_EMPRESA\_H

#include "CCliente.h"

// PREGUNTA 14: plantilla para CEmpresa

template <class T> class CEmpresa

{

private:

T\* m\_pElem;

int m\_nElem;

public:

CEmpresa() : m\_nElem(0), m\_pElem(NULL) {};

CEmpresa(const CEmpresa<T>& a);

CEmpresa<T>& operator=(const CEmpresa<T>& a);

~CEmpresa();

void AgregarElemento(const T& elem);

int Tamanyo() const { return m\_nElem; }

T& GetElemento(int nElem) const;

T& operator[](int nElem) const;

}

#include "CEmpresa.cpp"

#endif

**CEmpresa.cpp**

#if !defined(\_EMPRESA\_CPP)

#define \_EMPRESA\_CPP

#include "CEmpresa.h"

#include "CIndiceIncorrecto.h"

#include "MemoryManager.h"

;template<class T>

CEmpresa<T>::CEmpresa(const CEmpresa<T> &a)

{

m\_pElem = NULL;

m\_nElem = 0;

\*this = a;

}

// Operador =

template<class T>

CEmpresa<T>& CEmpresa<T>::operator=(const CEmpresa<T>& a)

{

if(m\_pElem) delete [] m\_pElem;

if(a.Tamanyo()!=0)

{

for (int i=0;i<a.m\_nElem;i++)

AgregarElemento(a.GetElemento(i));

}

return \* this;

}

// Destructor

template<class T>

CEmpresa<T>::~CEmpresa()

{

if(m\_pElem) delete [] m\_pElem;

}

// AgregarElemento

template<class T>

void CEmpresa<T>::AgregarElemento(const T& elem)

{

m\_nElem++;

if (m\_nElem==1)

{

m\_pElem = new T [m\_nElem] ;

m\_pElem[0]=elem;

}

else

{

T\* nuevo=NULL;

nuevo = new T[m\_nElem] ;

nuevo[m\_nElem -1]=elem;

for(int i=0;i<(m\_nElem-1);i++)

nuevo[i]= m\_pElem[i];

delete []m\_pElem;

m\_pElem = nuevo;

}

}

// GetElemento

template<class T>

T& CEmpresa<T>::GetElemento(int nElem) const

{

if((nElem<0)||(nElem>m\_nElem)) throw CIndiceIncorrecto::CIndiceIncorrecto();

return m\_pElem[nElem];

}

// Operador de indexación: []

template<class T>

T& CEmpresa<T>::operator[](int nElem) const

{

if((nElem<0)||(nElem>m\_nElem)) throw CIndiceIncorrecto::CIndiceIncorrecto();

return m\_pElem[nElem];

}

#endif

**CIndiceIncorrecto.h**

#if !defined(\_INDICE\_INCORRECTO\_)

#define \_INDICE\_INCORRECTO\_

#include<string>

using namespace std;

class CIndiceIncorrecto

{

private:

string error;

public:

CIndiceIncorrecto()

{

error="ERROR. Indice fuera de rango.";

}

string what(){return error;}

}

#endif

**Main.cpp**

#include <iostream>

#include "CLista.h"

#include "CContrato.h"

#include "CSiniestroUrgente.h"

#include "CSiniestroNormal.h"

#include "CCliente.h"

#include "CUtils.h"

#include "MemoryManager.h"

#include "CEmpresa.h"

;using namespace std;

using namespace utils;

//Main de la practica

const int MAX\_CLIENTES = 10;

int main()

{

{

char \* menu[]={" Vaciar() de CLista.",

" Operador [] CLista.",

" Constructor copia de CContrato.",

" AgregarSiniestro() de CContrato.",

" Operador << CCliente.",

" m\_pSigCodigo static.",

" Presupuestar() de CSiniestro y derivadas",

" AgregarContrato() de CCliente",

" total += seguros[i]",

" Constructor CSiniestro y CSiniestroNormal",

" const\_cast<CLista<T>\*>(this)->",

" GetCoste()?",

" c.AgregarSiniestro()",

" Plantilla CEmpresa.",

" Salir."

};

const int numopciones = sizeof(menu)/sizeof(char \*);

int opcion=0;

do

{

opcion = CUtils::CrearMenu(menu,numopciones);

switch(opcion)

{

case 1:

{

// Crear una lista para después vaciarla

CLista<string> miLista;

string nombres[]={"Nombre 1", "Nombre 2", "Nombre 3", "Nombre 4", "Nombre 5",

"Nombre 6", "Nombre 7", "Nombre 8", "Nombre 9", "Nombre 10"};

// Rellenar la lista con nombres

for(int i=0; i<10;i++)

miLista.AgregarObjeto(nombres[i]);

// Mostrar la lista. Preveer execepción CIndiceIncorrecto.

try

{

cout<<miLista.GetPrimero()<<endl;

for(int i=1;i<10;i++)

cout <<miLista.GetActual()<<endl;

}

catch(CIndiceIncorrecto& i)

{

cout<<i.what()<<endl;

}

// Vaciar la lista y verificar después que está vacía.

miLista.Vaciar();

if(miLista.EstaVacia())

cout<<"La lista esta vacia"<<endl;

cout << "Respuestas a las preguntas: Si es necesario ya que hay algunos metodos de CLista, por ejemplo GetPrimero(),"

<<"que necesita usar metodos de CNodoLista, como GetDato()" << endl;

system("pause");

break;

}

case 2:

{

// Ejemplo operador [] con lista de enteros

// Crear un objeto lista que almacene 10 datos de tipo int

CLista<int> listaInt;

int entero;

for(int i=0; i<10; i++)

{

utils::CUtils::leerInt(entero);

listaInt.AgregarObjeto(entero);

}

// Sumar todos los números de la lista y provocar la excepción CIndiceIncorrecto

int suma=0;

try

{

for(int i=0;i<19;i++)

{

suma += listaInt[i]->GetDato();

}

}

catch(CIndiceIncorrecto& i)

{

cout<<i.what()<<endl;

}

cout<<"\nSuma"<<suma<<endl;

system("pause");

break;

}

case 3: // Pregunta 3

{

// Crear e iniciar el objeto CContrato a copiar

CContrato original1(12345,"Prueba constructor copia","11-11-11",1001,11);

CSiniestroNormal siniestro1("Bateria");

original1.AgregarSiniestro(siniestro1);

//Uso del constructor copia y del operador de asignacion

CContrato copia1(original1);

// Mostrar original y copia

cout<<"Original 1:"<<endl;

original1.MostrarSiniestros(cout);

cout<<endl;

cout<<"Copia del original 1:"<<endl;

copia1.MostrarSiniestros(cout);

cout<<endl;

break;

}

case 4:

{

CContrato prueba(12345);

// Agregar a "prueba" un siniestro urgente y otro normal

CSiniestroUrgente sinurge(nacional,"Frigorifico no enfria");

prueba.AgregarSiniestro(sinurge);

prueba.MostrarSiniestros(cout);

cout << endl;

cout << "Respuestas a las preguntas: Si, porque la lista de punteros es de la clase "

<<"base y al llamar a clonar se necesita llamar al de las derivadas." << endl;

system("pause");

break;

}

case 5:

{

// Crear cliente

CCliente cli("Juan");

// Agregar un contrato con dos siniestros, urgente y normal, a cli

CContrato concli(12,"Siniestros", "12-12-2015", 4586, 250);

CSiniestroNormal sinnorm("Golpe");

CSiniestroUrgente sinurge(local,"Fallo en el sistema");

concli.AgregarSiniestro(sinnorm);

concli.AgregarSiniestro(sinurge);

cli.AgregarContrato(concli);

// Mostrar datos

cout << " Datos del cliente: \n" << endl;

cout << cli << endl;

system("pause");

cout << "Llamada explícita:\n" /\*<< ...\*/;

system("pause");

break;

}

case 6:

{

cout << "Respuestas a las preguntas: La iniciación de m\_sigCodigo se hace en CSiniestro.cpp aunque se podría hacer en otro fichero si se hiciese referencia a la clase " << endl;

system("pause");

break;

}

case 7:

{

CSiniestroUrgente MiSiniestroUrgente(internacional,"Siniestro Urgente Ejemplo");

MiSiniestroUrgente.Presupuestar(7,3);

CSiniestroNormal MiSiniestroNormal("Siniestro Normal Ejemplo");

MiSiniestroNormal.Presupuestar(9,4);

cout << endl

<< "Respuestas a las preguntas: En el primer caso, m\_Coste debe ser protected "

<< "porque las clases derivadas no pueden acceder a los miembros privados de su"

<< " clase base. En el segundo caso, si podria ser privado porque se podria acceder"

<< " a el a traves del miembro publico GetCoste() de la clase base y eso si esta permitido."

<< endl;

system("pause");

break;

}

case 8:

{

// Crear un cliente cli

CCliente cli("Roberto");

// Agregar a cli un contrato con siniestros

CContrato con(791928, "Rotura motor", "8-3-2013", 90, 9999999);

cli.AgregarContrato(con);

// Mostrar cli

cout << "Datos del cliente: " << cli << endl;

system("pause");

break;

}

case 9:

{

// Crear una matriz dinámica de tipo CCliente apuntada por "seguros"

CCliente \*seguros = new CCliente[MAX\_CLIENTES];

// Agregar contratos a los clientes de la matriz "seguros"

CContrato con(36);

CSiniestroNormal sin("Atasco");

con.AgregarSiniestro(sin);

for(int i=0; i< MAX\_CLIENTES; i++)

seguros[i].AgregarContrato(con);

// Contar el número total de contratos

long total = 0;

for(int i=0; i < MAX\_CLIENTES; i++)

total += seguros[i];

cout << "\n El numero total de contratos de los clientes asciende a: ";

cout << total << " contratos\n";

system("pause");

delete [] seguros;

break;

}

case 10:

{

CSiniestroNormal s("Rotura de tambor");

cout << endl;

s.Mostrar(); cout << endl;

cout << "Respuesta a la pregunta:Accede a CSinietsroNormal::CSniestroNormal(const std::string& Desc) y a CSiniestro::CSiniestro(const std::string& Desc)" << endl;

system("pause");

break;

}

case 11:

{

cout << "Respuesta a la pregunta: No se puede quitar porque al ser GetPrimero()"

<< "un metodo const no se puede modificar un atributo privado como es m\_Actual"

<< endl;

system("pause");

break;

}

case 12:

{

cout << "Respuesta a la pregunta: Si, porque el metodo GetCoste() pertenece a la misma clase que Mostrar() y ambos son publicos." << endl;

system("pause");

break;

}

case 13:

{

CContrato c(12345, "Cafetera C1Z","2/1/2008", 100, 1000);

CSiniestroUrgente s(nacional, "Fallo general");

c.AgregarSiniestro(s);

cout << "Respuestas a las preguntas:\n"

<< " - void CContrato::AgregarSiniestro(const CSiniestro& s)\n"

<< " - CSiniestroUrgente\* CSiniestrorgente::Clonar() const\n"

<< " - void CLista<T>::AgregarObjeto(const T& Obj)\n"

<< " - bool EstaVacia()\n"

<< endl;

system("pause");

break;

}

case 14:

{

// Crear empresa1

CEmpresa<CCliente> empresa1;

// Agregar elementos a la empresa

CCliente cli("Juan");

empresa1.AgregarElemento(cli);

// Crear empresa2 como copia de empresa1

CEmpresa <CCliente> empresa2(empresa1);

// Mostrar la empresa2

cout << "\n CLIENTES DE LA EMPRESA: \n" << endl;

for(int i=0;i<empresa2.Tamanyo(); i++)

{

cout << "Cliente" << (i+1) << empresa2.GetElemento(i) << endl;

}

system("pause");

break;

}

}

}

while(opcion < numopciones);

}

MemoryManager::dumpMemoryLeaks();

system("pause");

}