Facultad de Ingeniería y Ciencias Hídricas

FICH

UNL

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Ingeniería en Informática

PROGRAMACIÓN ORIENTADA A OBJETOS

UNIDAD 3

Ejercicio 3.01

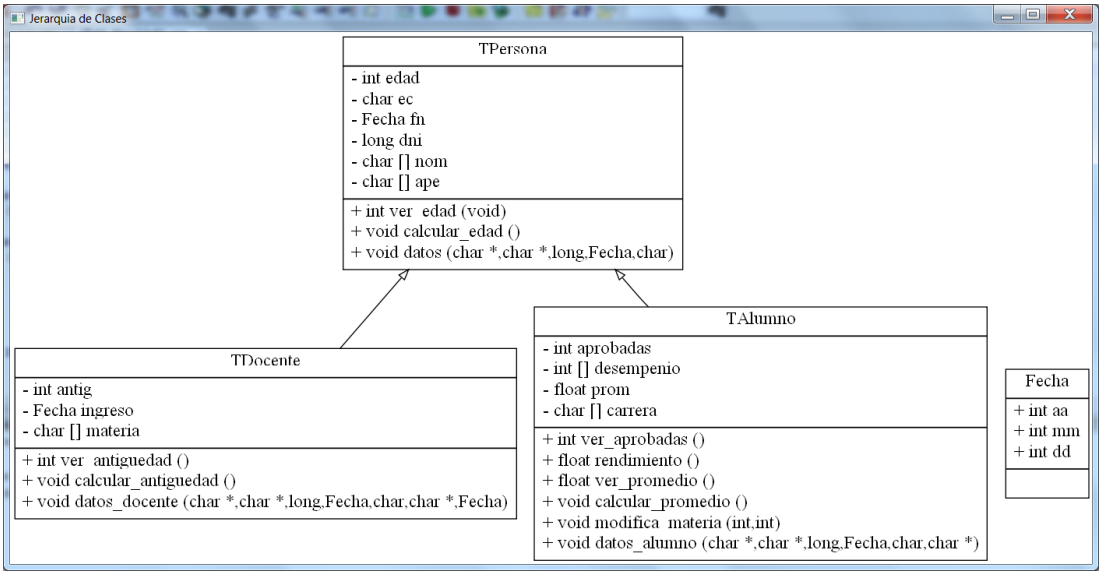
Diseñe una clase *Persona* que contenga los siguientes atributos: apellido y nombre, DNI, año de nacimiento y estado civil. La clase debe poseer, además, un método *Edad()* que calcule la edad actual de la persona en base al año de nacimiento.

Implemente una clase *Alumno* para contener la siguiente información de un alumno: apellido y nombre, DNI, año de nacimiento, estado civil, promedio y cantidad de materias aprobadas. La clase debe poseer, además, un método *MeritoAcadémico()* que devuelve el mérito académico del alumno (este se calcula como el cociente entre el promedio y la cantidad de materias aprobadas.

Cree, también, una clase *Docente* para modelar un docente a partir de la siguiente información: apellido y nombre, DNI, año de nacimiento, estado civil y año de ingreso. La clase debe poseer, además, un método *Antiguedad()* que calcule la antigüedad del docente en base a su año de ingreso y la fecha actual.

Proponga una jerarquía de clases adecuada para evitar repetir atributos. Implemente constructores y métodos extra que considere adecuados. Codifique un programa cliente que cree instancias de *Alumno* y *Docente* y utilice sus funciones.

Respuestas:




```
#include <iostream>
#include <cstring>
#include <ctime>
using namespace std;
typedef struct
{ int dd, mm, aa;}Fecha;

class TPersona //Clase Base
{ private:
        string ape, nom;
        long  dni;
        Fecha fn;
        char ec;
        int edad;
public:
        void datos (string ap, string no, long dn, Fecha f, char e);
        void calcular_edad();
        int ver_edad(void);
};

//observe la sintaxis:
//    class nombre_clase_derivada: public nombre_clase_base;

class TAlumno: public TPersona {
private:
        string carrera;
        float prom;
        int desempenio[40];
        // desempenio[ ] guarda  'Regular = 0','Libre = -1', '1','2','3',..,'9','10', '-10' si no
cursó la  materia
        int aprobadas;
public:
```

```
        void datos_alumno (string ap, string no, long dn, Fecha f, char e, string ca-
rrera);
        void modifica_materia(int nro, int resultado);
        void calcular_promedio();
        float ver_promedio() {return(prom);};
        float rendimiento() {return prom/aprobadas;};
        int ver_aprobadas() {return(aprobadas);};
};

class TDocente: public TPersona {
private:
        string materia;
        Fecha ingreso;
        int antig;
public:
        void datos_docente(string ap, string no, long dn, Fecha f, char e, string mate-
ria1, Fecha i);
        void calcular_antiguedad();
        int ver_antiguedad(){return antig;};
};
//*** CLASE TPersona ***
void TPersona::datos (string ap, string no, long dn, Fecha f, char e){
ape= ap;
nom= no;
dni= dn;
fn= f;
ec= e;}

void TPersona::calcular_edad(){
        Fecha pf;
        time_t actual = time(NULL);
        struct tm fechaHoy = *(localtime(&actual));
        pf.dd= fechaHoy.tm_mday;
        pf.mm= fechaHoy.tm_mon+1;
        pf.aa= fechaHoy.tm_year+1900;
        edad = pf.aa - fn.aa;
        if (fn.mm > pf.mm)
                edad= edad-1;
        else
                if ((fn.mm == pf.mm)&&(fn.dd > pf.dd))
                        edad= edad-1;
}
//******** CLASE TAlumno *************
void TAlumno::datos_alumno (string ap, string no, long dn, Fecha f, char e, string ca)
{datos( ap, no, dn, f, e);
carrera= ca;
}

void TAlumno::modifica_materia(int nro, int resultado)
{desempenio[nro]= resultado;}
```

```
void TAlumno::calcular_promedio()
{int c=0, j;
float suma = 0;
aprobadas = 0;
for ( j=0; j < 10; j++)
{  if (desempenio[j]>=0)
{ c++;
suma=suma + (int) desempenio[j];
};
if (desempenio[j]>=6)
        aprobadas++;
};
prom = suma/c;
}
//******* CLASE TDocente ********
void TDocente::datos_docente(string ap, string no, long dn, Fecha f, char e, string materia1, Fecha i)
{datos( ap, no, dn, f, e);
materia= materia1;
ingreso= i;}

void TDocente::calcular_antiguedad(){
        Fecha hoy;
        time_t actual = time(NULL);
        struct tm fechaHoy = *(localtime(&actual));
        hoy.dd= fechaHoy.tm_mday;
        hoy.mm= fechaHoy.tm_mon+1;
        hoy.aa= fechaHoy.tm_year+1900;
        antig = hoy.aa - ingreso.aa;
        if (ingreso.mm > hoy.mm)
                antig--;
        else
                if ((ingreso.mm == hoy.mm)&&(ingreso.dd > hoy.dd))
                        antig--;
}

//CUERPO PRINCIPAL DEL PROGRAMA
//(c)Profesor Gerardo Sas.
int main(int argc, char *argv[]) {
        TAlumno A;
        TDocente D;
        string ap1, no1;
        long dn1;
        Fecha f1;
        char e1;
        int r1;
        Fecha hoy;
        time_t actual = time(NULL);
        struct tm fechaHoy = *(localtime(&actual));
        hoy.dd= fechaHoy.tm_mday;
        hoy.mm= fechaHoy.tm_mon+1;
```

```
        hoy.aa= fechaHoy.tm_year+1900;
        cout<<"fecha de Hoy: "<<hoy.dd<<"/"<<hoy.mm<<"/"<<hoy.aa<<endl;
        string carrera11;
        cout<<endl<<"DATOS del Alumno"<<endl;
        cout<<endl<<" Apellido: "; cin>> ap1;
        cout<<endl<<" Nombre: "; cin>> no1;
        cout<<endl<<" D.N.I.: "; cin>> dn1;
        cout<<endl<<" Nacimiento dia: "; cin>> f1.dd;
        cout<<endl<<" Nacimiento mes: "; cin>> f1.mm;
        cout<<endl<<" Nacimiento anio: "; cin>> f1.aa;
        cout<<endl<<" Estado Civil: "; cin>> e1;
        cout<<endl<<" Carrera: "; cin>> carrera11;
        A.datos_alumno(ap1, no1, dn1, f1, e1, carrera11);
        cout<<endl<<"'Regular = 0','Libre = -1', '1','2','3',..,'9','10', '-10' si no cursó la
materia";
        for (int x=0; x<10; x++){
                cout<<endl<<x<<"- Nota: ";
                cin>> r1;
                A.modifica_materia(x, r1);
        };
        A.calcular_promedio();
        cout<<endl<<"PROMEDIO: "<<A.ver_promedio();
        cout<<endl<<"RENDIMIENTO ACADEMICO: "<<A.rendimiento();
        cout<<endl<<"Pulse <<Enter>> para continuar";
        cin>>e1;
        Fecha i1;
        string  materia1;
        cout<<endl<<"DATOS DEL DOCENTE"<<endl;
        cout<<endl<<" Apellido: "; cin>> ap1;
        cout<<endl<<" Nombre: "; cin>> no1;
        cout<<endl<<" D.N.I.: "; cin>> dn1;
        cout<<endl<<" Nacimiento dia: "; cin>> f1.dd;
        cout<<endl<<" Nacimiento mes: "; cin>> f1.mm;
        cout<<endl<<" Nacimiento anio: "; cin>> f1.aa;
        cout<<endl<<" Ingreso dia: "; cin>> i1.dd;
        cout<<endl<<" Ingreso mes: "; cin>> i1.mm;
        cout<<endl<<" Ingreso anio: "; cin>>i1.aa;
        cout<<endl<<" Estado Civil: "; cin>> e1;
        cout<<endl<<" Materia: "; cin>> materia1;
        D.datos_docente(ap1, no1, dn1, f1, e1, materia1,i1);
        D.calcular_edad();
        D.calcular_antiguedad();
        cout<<endl<<"ANTIGUEDAD: "<<D.ver_antiguedad();
        cout<<endl<<"Pulse <<Enter>> para continuar";
        cin>>e1;
}//(c)Profesor Gerardo Sas.
```

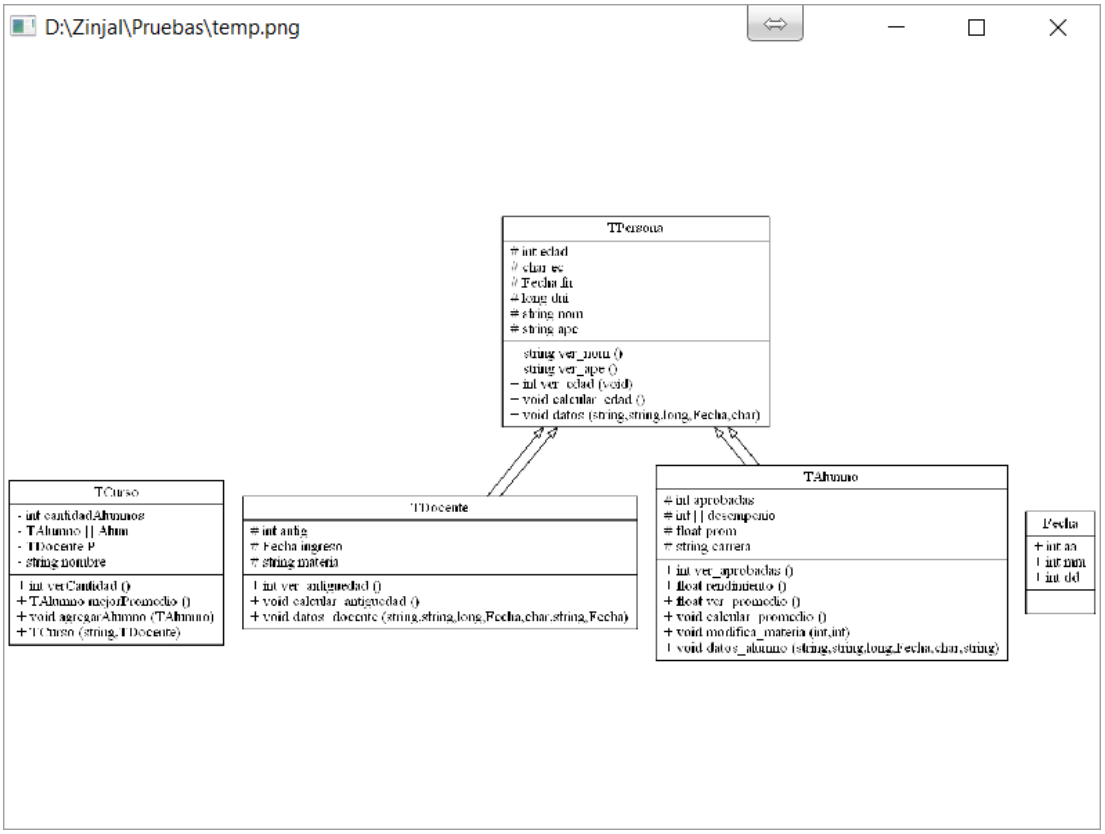
Responda:

- ¿Puede crearse un objeto de tipo persona? SI
- ¿Para qué sirve esto? NO SIRVE
- ¿Existe alguna clase abstracta en la jerarquía? **NO**

Una clase abstracta, es una que está diseñada sólo como clase padre de las cuales se deben derivar clases hijas. Una clase abstracta se usa para representar aquellas entidades o métodos que después se implementarán en las clases derivadas, pero la clase abstracta en sí no contiene ninguna implementación -- solamente representa los métodos que se deben implementar. Por ello, no es posible instanciar una clase abstracta, pero sí una clase concreta que implemente los métodos definidos en ella.
Las clases abstractas son útiles para definir interfaces, es decir, un conjunto de métodos que definen el comportamiento de un módulo determinado. Estas definiciones pueden utilizarse sin tener en cuenta la implementación que se hará de ellos. En C++ los métodos de las clases abstractas se definen como funciones virtuales puras.

Ejercicio 3.02

Utilice las clases *Alumno* y *Docente* del ejercicio anterior para crear una clase *Curso* que modele el cursado de una materia. Cada curso tiene un nombre, un profesor a cargo y un número máximo de 10 alumnos. Implemente un método *AgregarAlumno* que permita agregar un alumno al curso y otro método *MejorPromedio()* que devuelva el alumno con mejor promedio. Proponga los constructores y métodos extra que considere necesarios.



```
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar
//U3 Ejercicio Nº 02
#include <iostream>
#include <ctime>
#include <cstring>
#include <cstdio>
using namespace std;

typedef struct { int dd, mm, aa;}Fecha;
class TPersona //Clase Base
{ protected:
        string ape, nom;
        long  dni;
        Fecha fn;
        char ec;
        int edad;
public:
        void datos (string ap, string no, long dn, Fecha f, char e);
        void calcular_edad();
        int ver_edad(void);
        string ver_ape(){return ape;};
        string ver_nom(){return nom;};
};
//observe la sintaxis: class nombre_clase_derivada: public nombre_clase_base;
class TAlumno: public TPersona {
protected:
        string carrera;
        float prom;
        int desempenio[40];
        // desempenio[ ] guarda  'Regular = 0','Libre = -1', '1','2','3',..,'9','10', '-10' si no
cursó la  materia
        int aprobadas;
public:
        void datos_alumno (string ap, string no, long dn, Fecha f, char e, string ca-
rrera);
        void modifica_materia(int nro, int resultado);
        void calcular_promedio();
        float ver_promedio() {return(prom);};
        float rendimiento() {return prom/aprobadas;};
        int ver_aprobadas() {return(aprobadas);};
};
class TDocente: public TPersona {
protected:
        string materia;
        Fecha ingreso;
        int antig;
public:
        void datos_docente(string ap, string no, long dn, Fecha f, char e, string mate-
ria1, Fecha i);
        void calcular_antiguedad();
```

```
        int ver_antiguedad(){return antig;};
};
//*** CLASE TPersona ***
void TPersona::datos (string ap, string no, long dn, Fecha f, char e){
        ape= ap;
        nom, no;
        dni= dn;
        fn= f;
        ec= e;}
void TPersona::calcular_edad(){
        Fecha pf;
        time_t actual = time(NULL);
        struct tm fechaHoy = *(localtime(&actual));
        pf.dd= fechaHoy.tm_mday;
        pf.mm= fechaHoy.tm_mon+1;
        pf.aa= fechaHoy.tm_year+1900;
        edad = pf.aa - fn.aa;
        if (fn.mm > pf.mm)
                edad= edad-1;
        else
                if ((fn.mm == pf.mm)&&(fn.dd > pf.dd))
                        edad= edad-1;}
//******** CLASE TAlumno *************
void TAlumno::datos_alumno (string ap, string no, long dn, Fecha f, char e, string ca){

        datos( ap, no, dn, f, e);
        carrera=  ca;}
void TAlumno::modifica_materia(int nro, int resultado){
desempenio[nro]= resultado;}
void TAlumno::calcular_promedio(){
int c=0, j;
float suma = 0;
aprobadas = 0;
for ( j=0; j < 10; j++){
  if (desempenio[j]>=0)
        { c++; suma=suma + (int)desempenio[j];};
if (desempenio[j]>=6)
        aprobadas++;};
prom = suma/c;
}
//******* CLASE TDocente ********
void TDocente::datos_docente(string ap, string no, long dn, Fecha f, char e, string
materia1, Fecha i)
{datos( ap, no, dn, f, e);
materia= materia1;
ingreso= i;}
void TDocente::calcular_antiguedad(){
        Fecha hoy;
        time_t actual = time(NULL);
        struct tm fechaHoy = *(localtime(&actual));
        hoy.dd= fechaHoy.tm_mday;
```

```
	hoy.mm= fechaHoy.tm_mon+1;
	hoy.aa= fechaHoy.tm_year+1900;
	antig = hoy.aa - ingreso.aa;
	if (ingreso.mm > hoy.mm)
		antig--;
	else
		if ((ingreso.mm == hoy.mm)&&(ingreso.dd > hoy.dd))
			antig--;}
/*curso tiene un nombre, un profesor a cargo y un número máximo de 10 alumnos.
Implemente un método AgregarAlumno que permita agregar un alumno al curso y
otro método MejorPromedio() que devuelva el alumno con mejor promedio.
Proponga los constructores y métodos extra que considere necesarios.*/
class TCurso{
private:
	string nombre;
	TDocente P;
	TAlumno Alum[10];
	int cantidadAlumnos;
public:
	TCurso(string nom, TDocente prof);
	void agregarAlumno(TAlumno);
	TAlumno mejorPromedio();
	int verCantidad(){return cantidadAlumnos;};
};
//Implementacion clase TCurso
TCurso::TCurso(string nom, TDocente prof){
	nombre=  nom;
	P= prof;
	P.calcular_antiguedad();
	P.calcular_edad();
	cantidadAlumnos=0;
}
void TCurso::agregarAlumno(TAlumno Alu){
	Alum[cantidadAlumnos]= Alu;
	Alum[cantidadAlumnos].calcular_edad();
	Alum[cantidadAlumnos].calcular_promedio();
	cantidadAlumnos++;
}
TAlumno TCurso::mejorPromedio(){
	TAlumno max = Alum[0];
	max.calcular_promedio();
	for(int x=0; x< cantidadAlumnos; x++){
		Alum[x].calcular_promedio();
		//cout<<"max= "<<max.ver_promedio()<<" <   "<<Alum[x].ver_prome-
dio()<<endl;
		if(max.ver_promedio() < Alum[x].ver_promedio()){
			max= Alum[x];
			max.calcular_promedio();
		}
	}
	return max;
```

```
}

//CUERPO PRINCIPAL DEL PROGRAMA
int main(){
        TAlumno A;    TDocente D;
        string ap1, no1, nomc;
        long dn1; Fecha f1; char e1; int r1;
        string carrera11; char opcion; Fecha i1;
        Fecha hoy; string materia1;
        time_t actual = time(NULL);
        struct tm fechaHoy = *(localtime(&actual));
        hoy.dd= fechaHoy.tm_mday;
        hoy.mm= fechaHoy.tm_mon+1;
        hoy.aa= fechaHoy.tm_year+1900;
        cout<<"fecha de Hoy: "<<hoy.dd<<"/"<<hoy.mm<<"/"<<hoy.aa<<endl;
        cout<<endl<<" Ingrese Nombre del Curso: "; cin>>(nomc);
        cout<<endl<<"DATOS DEL DOCENTE"<<endl;
        cout<<endl<<" Apellido: "; cin>> ap1;
        cout<<endl<<" Nombre: "; cin>> no1;
        cout<<endl<<" D.N.I.: "; cin>> dn1;
        cout<<endl<<" Nacimiento dia: "; cin>> f1.dd;
        cout<<endl<<" Nacimiento mes: "; cin>> f1.mm;
        cout<<endl<<" Nacimiento anio: "; cin>> f1.aa;
        cout<<endl<<" Ingreso dia: "; cin>> i1.dd;
        cout<<endl<<" Ingreso mes: "; cin>> i1.mm;
        cout<<endl<<" Ingreso anio: "; cin>>i1.aa;
        cout<<endl<<" Estado Civil: "; cin>> e1;
        cout<<endl<<" Materia: "; cin>> materia1;
        D.datos_docente(ap1, no1, dn1, f1, e1, materia1,i1);
        D.calcular_antiguedad();D.calcular_edad();
        TCurso Cur(nomc, D);
        cout<<endl<<"Ingrese los datos de los alumnos"<<endl;
        opcion='N';
        do{
                cout<<endl<<"DATOS del Alumno N: "<<Cur.verCantidad()<<endl;
                cout<<endl<<" Apellido: "; cin>> ap1;
                cout<<endl<<" Nombre: "; cin>> no1;
                cout<<endl<<" D.N.I.: "; cin>> dn1;
                cout<<endl<<" Nacimiento dia: "; cin>> f1.dd;
                cout<<endl<<" Nacimiento mes: "; cin>> f1.mm;
                cout<<endl<<" Nacimiento anio: "; cin>> f1.aa;
                cout<<endl<<" Estado Civil: "; cin>> e1;
                cout<<endl<<" Carrera: "; cin>> carrera11;
                A.datos_alumno(ap1, no1, dn1, f1, e1, carrera11);
                cout<<endl<<"'Regular = 0','Libre = -1', '1','2','3',..,'9','10', '-10' si no
cursó la  materia";
                for (int x=0; x<10; x++){
                        cout<<endl<<x<<"- Nota: ";
                        cin>> r1;
                        A.modifica_materia(x, r1);
                };
```

```
                Cur.agregarAlumno(A);
                cout<<endl<<"Ingresa otro (S/N): ";
                cin>>opcion;
                opcion= toupper(opcion);
        }while(opcion != 'N') ;
        cout<<"Se ingresaron "<<Cur.verCantidad()<<" alumnos."<<endl;
        TAlumno AA(Cur.mejorPromedio());

        cout<<endl<<" MEJOR PROMEDIO: "<<endl;
        cout<<endl<<" Apellido: "<<AA.ver_ape()<<endl;
        cout<<endl<<" Nombre:   "<<AA.ver_nom()<<endl;
        cout<<endl<<" Promedio: "<<AA.ver_promedio()<<endl;

        //(string ap, string no, long dn, Fecha f, char e, string materia1, Fecha i)
}//(c)Profesor Gerardo Sas.
```

Ejercicio 3 (Se resuelve con el Ej. Nº 4)

Proponga una clase Punto con atributos y métodos para definir un punto en el plano (coordenadas x,y). Luego, proponga la clase RectaExplicita para definir la ecuación de la recta y=mx+b a partir de dos puntos. La declaración de dicha clase se muestra en el recuadro.

```
class TExplicita {
private:
        Punto p1, p2
        float m, b;
public:
        TExplicita(punto p1, punto p2);
                void obtener_ecuacion();
                void mostrarEcuacion();
                float ver_m() {return m;};
                float ver_b() {return b;};
        };
```

```
class TPunto {
private:
        float x,y;
public:
void asignar_valor(float xx, float yy);
        float ver_x() {return x;};
        float ver_y() {return y;};
};
```

El método MostrarEcuacion() debe mostrar en pantalla la ecuación explícita de la recta.

Ecuación General de la Recta: Ax+By+C=0 **(1)**

La ecuación **(1)** representa una linea recta, cuya pendiente es $m = -\frac{A}{B}$ y cuyo intercepto con el eje *y* viene dado por $b = -\frac{C}{B}$.

O sea que la Ecuación Explicita **(2) Y = m.X + b** puede expresarse en función de **(1)** con los coeficientes A, B, C.

A partir de las coordenadas de los puntos P1(x1,y1) y P2(x2,y2) obtenemos A,B,C:

```
A= -(y2-y1)
B=  (x2-x1)
C= -((y1*(x2-x1)-x1*(y2-y1)))
```

```
#include <iostream>
#include <sstream>
using namespace std;
class TPunto {
private:
	double x,y;
public:
	void asignar_valor(double xx, double yy);
	double ver_x() {return x;};
	double ver_y() {return y;};
};

class TExplicita{
private:
	TPunto P1, P2;
	double m, b;
public:
	TExplicita(TPunto &p1, TPunto &p2):P1(p1),P2(p2){};
	string obtenerEcuacion();
	double ver_m() {return m;};
	double ver_b() {return b;};
};

//*** Implementacion de los métodos de las clases ***
void TPunto::asignar_valor(double xx, double yy){
	x= xx; y= yy;
}
string TExplicita::obtenerEcuacion(){
	m = (P2.ver_y()-P1.ver_y())/(P2.ver_x()-P1.ver_x());
	b = P1.ver_y()  - (m * P1.ver_x());
	/*stringstream:  Los objetos de esta clase usan un buffer de cadena que
	contiene una secuencia de caracteres. Se puede acceder a esta secuencia
	de caracteres directamente como un objeto de cadena, utilizando miembro
str.
	*/
	stringstream aux;
	aux.str("");
	aux << " Y = ";
	aux << ver_m();
	aux << " . X + ";
	aux << ver_b();
	return aux.str();
}

int main(int argc, char *argv[]) {
	TPunto punto1, punto2;
	punto1.asignar_valor(1,1);
	punto2.asignar_valor(4,4);
	cout<<"P1.x= "<<punto1.ver_x()<<" , P1.y= "<<punto1.ver_y()<<endl;
	cout<<"P2.x= "<<punto2.ver_x()<<" , P2.y= "<<punto2.ver_y()<<endl;
	TExplicita R(punto1, punto2);
```

```
        string ecuacion="";
        ecuacion= R.obtenerEcuacion();
        cout<<ecuacion<<endl;;
        return 0;
}
```

Ejercicio 4

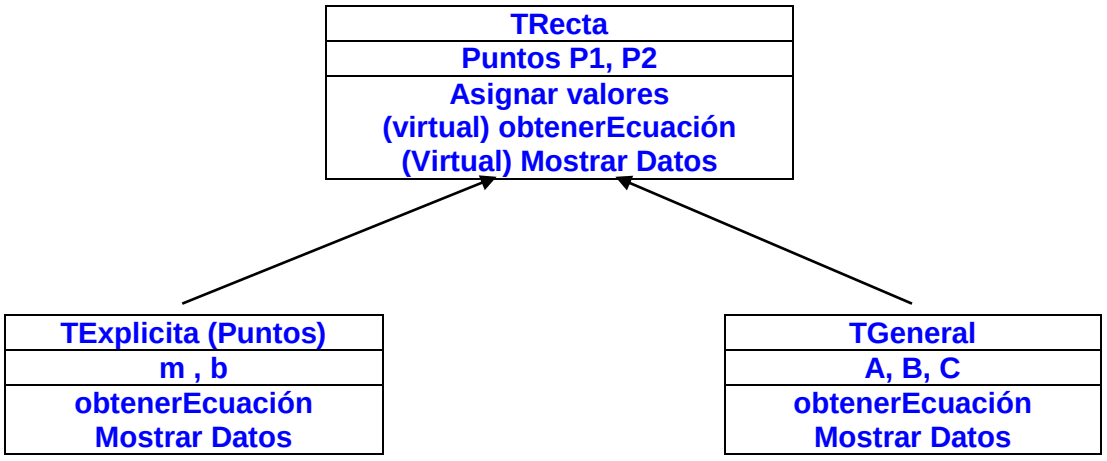
a) Proponga una clase RectaGeneral para representar una recta general, cuya ecuación es Ax+By+C=0, a partir de dos puntos. El prototipo de la clase se muestra en el siguiente recuadro.

```
class RectaGeneral {
private:
   Punto &P1, &P2;
   float a, b, c;
public:
    RectaGeneral(Punto &p1, Punto &p2);
    string obtener_Ecuacion();
    float Ver_a();
    float Ver_b();
    float Ver_c();
}
```

```
#include <iostream>
#include <sstream>
using namespace std;
class TPunto {
private:
        double x,y;
public:
        void asignar_valor(double xx, double yy);
        double ver_x() {return x;};
        double ver_y() {return y;};
};
class RectaGeneral {
private:
        TPunto P1, P2;
        double A,B,C;
public:
        RectaGeneral(TPunto &p1, TPunto &p2):P1(p1),P2(p2){};
        string obtenerEcuacion();
        double Ver_A(){return A;};
        double Ver_B(){return B;};
        double Ver_C(){return C;};
};
//*** Implementacion de los métodos de las clases ***
void TPunto::asignar_valor(double xx, double yy){
        x= xx; y= yy;
}
```

```
string RectaGeneral::obtenerEcuacion()
{/* A= -(y2-y1)
        B=  (x2-x1)
        C= -(y1*(x2-x1) - x1*(y2-y1))*/
        A = -(P2.ver_y()-P1.ver_y());
        B = (P2.ver_x()-P1.ver_x());
        C = -((P1.ver_y()* (P2.ver_x()-P1.ver_x())) - P1.ver_x()*(P2.ver_y()-
P1.ver_y()));
        stringstream aux;
        aux.str("");
   aux << Ver_A()<<"  X + "<<Ver_B()<<" Y + "<<Ver_C()<<"  = 0"<<endl;
        return aux.str();
}
//*****************************************************************************
int main(int argc, char *argv[]) {
        TPunto punto1, punto2;
        punto1.asignar_valor(1,1);
        punto2.asignar_valor(4,4);
        RectaGeneral R(punto1,punto2);
        cout<<"P1.x= "<<punto1.ver_x()<<" , P1.y= "<<punto1.ver_y()<<endl;
        cout<<"P2.x= "<<punto2.ver_x()<<" , P2.y= "<<punto2.ver_y()<<endl;
        cout<<R.obtenerEcuacion();
        return 0;
}
```

b) Diseñe un árbol de herencia que incluya una clase Recta, y dos herederas llamadas RectaExplicita y RectaGeneral. Utilizando los conceptos de polimorfismo, métodos virtuales y abstractos, desarrolle una estructura con métodos polimórficos.

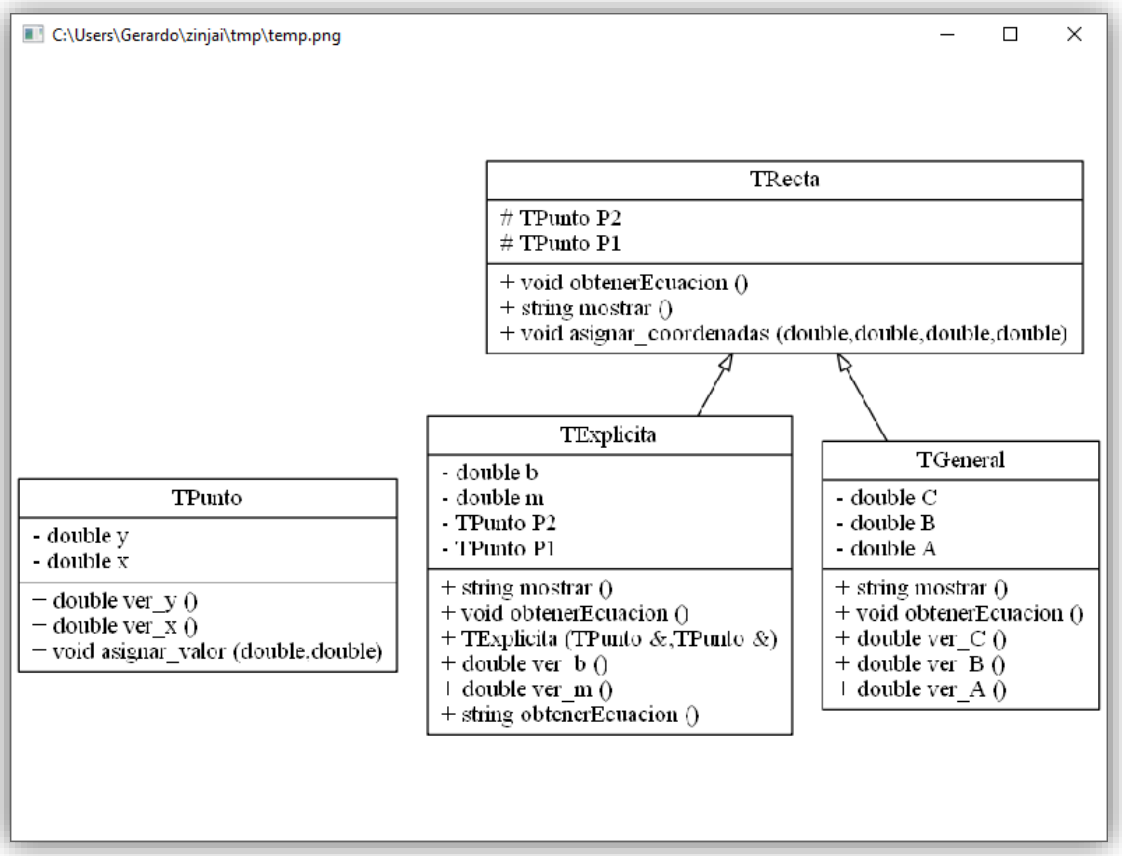


Codificacaión en C++ del algoritmo:

```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar
//U3 Ejercicio Nº 4 - Item c)
#include <iostream>
#include <sstream>
using namespace std;
//*********************************************************************
class TPunto {
private:
        double x,y;
public:
        void asignar_valor(double xx, double yy);
        double ver_x() {return x;};
        double ver_y() {return y;};
};
//*********************************************************************
class TRecta{ //Clase BASE Abstracta porque tiene al menos 1 metodo virtual puro
protected:
        TPunto P1, P2;//Por agregacion o composición de clases
public:
        void asignar_coordenadas(double, double, double, double);
        virtual void obtenerEcuacion()=0;//Metodo virtual puro
        virtual string mostrar()=0;//Metodo virtual puro
        virtual ~TRecta(){};
};
//*** TGeneral herencia simple de TRecta ******************************
class TGeneral: public TRecta{
private:
        double A,B,C;
public:
        void obtenerEcuacion();
        string mostrar();
        double ver_A() {return A;};
        double ver_B() {return B;};
        double ver_C() {return C;};
};
//*** TExplicita herencia simple de TRecta ****************************
class TExplicita: public TRecta
{private:
        double m, b;
public:
        void obtenerEcuacion();
        string mostrar();
        double ver_m() {return m;};
        double ver_b() {return b;};
};
//*** Implementacion de los métodos de las clases ***
void TPunto::asignar_valor(double xx, double yy){
        x= xx; y= yy;
```

```
}
void TRecta::asignar_coordenadas(double x1, double y1, double x2, double y2){
        P1.asignar_valor(x1,y1);
        P2.asignar_valor(x2,y2);
}
void TExplicita::obtenerEcuacion() {
        m = (P2.ver_y()-P1.ver_y())/(P2.ver_x()-P1.ver_x());
        b = P1.ver_y() - (m * P1.ver_x());
}
void TGeneral::obtenerEcuacion()
{/* A= -(y2-y1)
        B= (x2-x1)16
        C= -(y1*(x2-x1) - x1*(y2-y1))*/
        A = -(P2.ver_y()-P1.ver_y());
        B = (P2.ver_x()-P1.ver_x());
        C = -((P1.ver_y()* (P2.ver_x()-P1.ver_x())) - P1.ver_x()*(P2.ver_y()-

                                        P1.ver_y()));
}
string TExplicita::mostrar() {
        stringstream aux;
        aux.str("");
        aux << " Y = "<<ver_m()<<" . X + "<<ver_b()<<endl;
        return aux.str();
}
string TGeneral::mostrar(){
        stringstream aux;
        aux.str("");
        aux <<endl<< ver_A()<<" X + "<<ver_B()<<" Y + "<<ver_C()<<" = 0"<<endl;
        return aux.str();
}
//-------------Cuerpo principal del Programa -------------------------
int main(int argc, char *argv[]) {
        double x1, x2, y1, y2;
        TExplicita R1;
        TGeneral R2;
        TRecta *PR;
        cout<<" FICH - P.O.O. "<<endl<<endl<<endl;
        cout << "DATOS DE LOS PUNTOS P1 Y P2."<<endl<< endl;
        cout << "Ingrese X1: "; cin>> x1;
        cout << "Ingrese Y1: "; cin>> y1;
        cout << endl;
        cout << "Ingrese X2: "; cin>> x2;
        cout << "Ingrese Y2: "; cin>> y2;
        cout << endl;
        PR= &R1;
        PR->asignar_coordenadas(x1,y1,x2,y2);
        PR->obtenerEcuacion();
        cout<<PR->mostrar();
        PR= &R2;
        PR->asignar_coordenadas(x1,y1,x2,y2);
```

```
        PR->obtenerEcuacion();
        cout<<PR->mostrar();
        return 0;
}
```



c) Utilizando los conceptos de polimorfismo, métodos virtuales y abstractos, complemente el diseño con dos métodos virtuales: MostrarEcuacion(…), para mostrar en pantalla la ecuación que corresponda para cada recta, y Pertenece(…) para saber si un tercer punto dado está en la recta. ¿Qué consideraciones especiales debe realizar al implementar la segunda función? ¿Qué problema de diseño puede marcar respecto al primer método?
Nota: al comparar flotantes en el segundo método, no debe utilizar ==, sino preguntar de alguna otra forma si son "muy parecidos" en lugar de exactamente iguales.

Rta: En general comparar dos variables double no se hace con ==, debido a la representación interna de esos valores reales (mantisa, exponente).

Lo más común es evaluar si la diferencia entre esos dos valores es menor que un EPSILON que previamente se define (de acuerdo a la precisión que se requiera), y si la diferencia es menor se dice que los números son iguales.

Ejemplo:

```
bool esIgual(double d1, double d2, double epsilon = 0.00000000001)
{
   return fabs(d1 - d2) < epsilon;
}
```

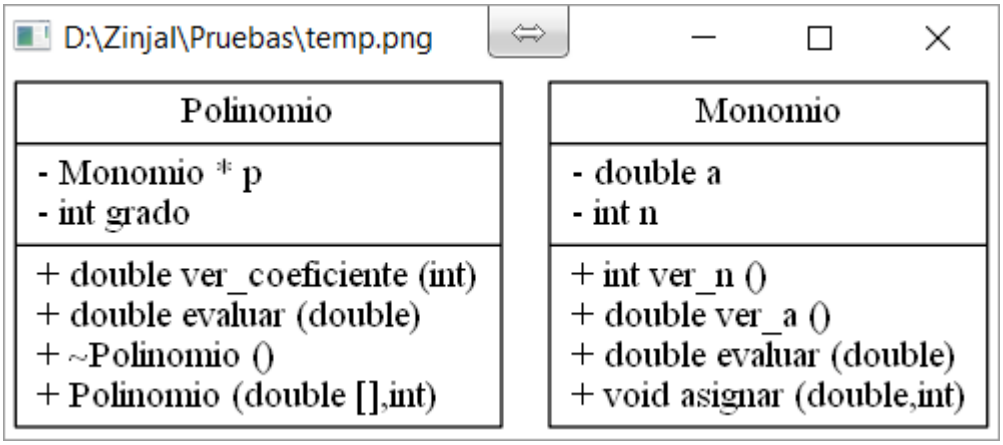
```
 //© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar
//U3 Ejercicio Nº 4 - Item c)
#include <iostream>
#include <sstream>
#include <cmath>
using namespace std;
//*************************************************************************
bool esIgual(double d1, double d2, double epsilon = 0.00000000001){
        return fabs(d1 - d2) < epsilon;
}
//*************************************************************************
class TPunto {
private:
        double x,y;
public:
        void asignar_valor(double xx, double yy);
        double ver_x() {return x;};
        double ver_y() {return y;};
};
//*************************************************************************
class TRecta{ //Clase BASE Abstracta porque tiene al menos 1 metodo virtual puro
protected:
        TPunto P1, P2;//Por agregacion o composición de clases
public:
        void asignar_coordenadas(double, double, double, double);
        virtual void obtenerEcuacion()=0;//Metodo virtual puro
        virtual string mostrar()=0;//Metodo virtual puro
        virtual bool pertenece(TPunto P)=0;//Metodo virtual puro
        virtual ~TRecta(){};
};
//*** TGeneral herencia simple de TRecta ******************************
class TGeneral: public TRecta{
private:
        double A,B,C;
public:
        void obtenerEcuacion();
        string mostrar();
        bool pertenece(TPunto P);
        double ver_A() {return A;};
```

```
        double ver_B() {return B;};
        double ver_C() {return C;};
};
//*** TExplicita herencia simple de TRecta ****************************
class TExplicita: public TRecta
{private:
        double m, b;
public:
        void obtenerEcuacion();
        string mostrar();
        bool pertenece(TPunto P);
        double ver_m() {return m;};
        double ver_b() {return b;};
};
//*** Implementacion de los métodos de las clases ***
void TPunto::asignar_valor(double xx, double yy){
        x= xx; y= yy;
}
void TRecta::asignar_coordenadas(double x1, double y1, double x2, double y2){
        P1.asignar_valor(x1,y1);
        P2.asignar_valor(x2,y2);
}
void TExplicita::obtenerEcuacion() {
        m = (P2.ver_y()-P1.ver_y())/(P2.ver_x()-P1.ver_x());
        b = P1.ver_y() - (m * P1.ver_x());
}
void TGeneral::obtenerEcuacion()
{/* A= -(y2-y1)
        B= (x2-x1)16
        C= -(y1*(x2-x1) - x1*(y2-y1))*/
        A = -(P2.ver_y()-P1.ver_y());
        B = (P2.ver_x()-P1.ver_x());
        C = -((P1.ver_y()* (P2.ver_x()-P1.ver_x())) - P1.ver_x()*(P2.ver_y()-

                                        P1.ver_y()));
}
string TExplicita::mostrar() {
        stringstream aux;
        aux.str("");
        aux << " Y = "<<ver_m()<<" . X + "<<ver_b()<<endl;
        return aux.str();
}
string TGeneral::mostrar(){
        stringstream aux;
        aux.str("");
        aux <<endl<< ver_A()<<" X + "<<ver_B()<<" Y + "<<ver_C()<<" = 0"<<endl;
        return aux.str();
}
bool TGeneral::pertenece(TPunto P){
        return esIgual((A*P.ver_x()+ B*P.ver_y()+C), 0);
}
```

```
bool TExplicita::pertenece(TPunto P){
        return esIgual(P.ver_y(),(ver_m()*P.ver_x()+ver_b()));
}
//-------------Cuerpo principal del Programa -------------------------
int main(int argc, char *argv[]) {
        double x1, x2, x3, y1, y2, y3;
        TPunto P3;
        TExplicita R1;
        TGeneral R2;
        TRecta *PR;
        cout<<" FICH - P.O.O. "<<endl<<endl<<endl;
        cout << "DATOS DE LOS PUNTOS P1 Y P2."<<endl<< endl;
        cout << "Ingrese X1: "; cin>> x1;
        cout << "Ingrese Y1: "; cin>> y1;
        cout << endl;
        cout << "Ingrese X2: "; cin>> x2;
        cout << "Ingrese Y2: "; cin>> y2;
        cout << endl;
        cout << "Ingrese X3: "; cin>> x3;
        cout << "Ingrese Y3: "; cin>> y3;
        cout << endl;
        PR= &R1;
        PR->asignar_coordenadas(x1,y1,x2,y2);
        PR->obtenerEcuacion();
        cout<<PR->mostrar();
        PR= &R2;
        PR->asignar_coordenadas(x1,y1,x2,y2);
        PR->obtenerEcuacion();
        cout<<PR->mostrar();
        P3.asignar_valor(x3,y3);
        if(PR->pertenece(P3))
                cout<<"El tercer punto pertenece a la Recta"<<endl;
        else
                cout<<"El tercer punto NO pertenece a la Recta"<<endl;
        return 0;
}
```

Ejercicio 5

Implemente una clase **Monomio** para representar un monomio (a*x^n) a partir de un coeficiente y un exponente, con un método *Evaluar (...)* que reciba un real y retorne el valor del monomio evaluado con ese real, y los demás métodos que considere necesarios. Implemente una clase **Polinomio** que reutilice la clase **Monomio** para modelar un polinomio, y añada un método *Evaluar (...)* para evaluar el polinomio en un valor x real dado. ¿Qué relación debe haber entre las clases monomio y polinomio?



Rta: Relacion de agregación

```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar
//U3 Ejercicio Nº 5
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;
//************************************************
class Monomio{//para representar un monomio (a*x^n)
        double a;//Coeficiente
        int n;//Grado +1 = Tamaño
public:
        void asignar(double aa, int gr){
                a=aa;
                n= gr;
        }
        double ver_a(){return a;}
        int ver_n(){return n;}
        double evaluar(double val_x){return a*pow(val_x,n);}
};
//************************************************
class Polinomio{
        Monomio *p;//Agregacion puntero a objetos Monomio
        int grado;//Tamaño
public://Constructor recibe vector de coeficientes y grado del Polinomio
                Polinomio(double v[], int g){
                        grado= g;//Asigno al atributo grado el parametro g
                        p= new Monomio[g];//Creo dinamicamente el arreglo de mo-
niomios
                        for(int x= grado-1; x>= 0; x--){
                                p[x].asignar(v[x],x);
```

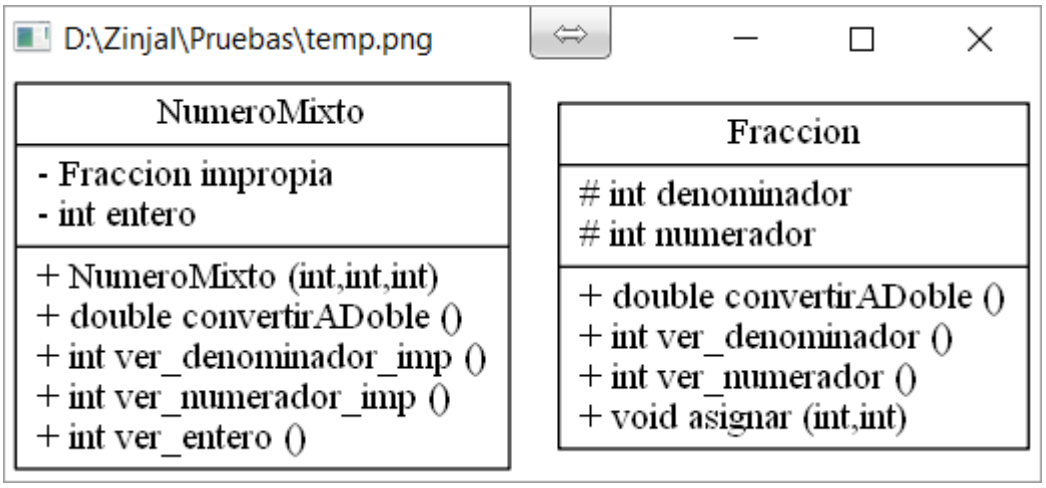
```
                        cout<<p[x].ver_a()<<"*X^"<<p[x].ver_n()<<"+";
                    }
                }
                ~Polinomio(){ delete []p;}
                double evaluar(double vx){
                    double vp= 0;
                    for(int i=0;i< grado; i++){
                        vp += p[i].evaluar(vx);
                    }
                    return vp;
                }
                double ver_coeficiente(int grado){return p[grado].ver_a();}
};
//*****************************************************************
int main(int argc, char *argv[]){
    double *coeficientes; int x;//Declaro vector para coeficientesç
    int grado;
    cout<<"Ingrese el Grado del Polinomio"<<endl;
    cin>>grado;
    int tamanio= grado+1;//Asigno tamaño un termino mas que el grado
    coeficientes = new double[tamanio];
    for (int k= 0; k<tamanio; k++){
        coeficientes[k]= rand()%10;
    }
    //Creo el polinomio con los coeficientes.
    Polinomio Poli(coeficientes, tamanio);
    //Muestro el Polinomio
    for (int k=0; k<tamanio; k++){
        cout<<"\nTermino "<<k<<": "<<Poli.ver_coefi-
ciente(k)<<"X^"<<k<<endl; }
                    //Evaluo el polinomio en un valor determinado que ingreso por
teclado.
                    cout<<"Ingrese un valor para X: ";cin>>x;
                    cout<<"Elvalor del polinomio en X= "<<x<<" es =
"<<Poli.evaluar(x)<<endl;
    return 0;
}
```

Ejercicio 6

Implemente una clase **Fraccion** para representar una fracción a partir de un numerador y un denominador, con un método *ConvertirADoble* para obtener el real que representa, y los demás métodos que considere necesarios.

Implemente una clase **NumeroMixto** para representar un número formado por una parte entera y una fracción impropia (fracción menor a 1). Reutilice la clase **Fraccion** al implementar la clase **NumeroMixto**. La clase NumeroMixto también debe tener un método ConvertirADouble. ¿Qué relación entre clases puede utilizar en este caso?

Rta: **Fracción:** Número que expresa una cantidad determinada de porciones que se toman de un todo dividido en partes iguales; se representa con una barra oblicua u horizontal que separa la primera cantidad (el numerador) de la segunda (el denominador). "un tercio (1/3) y cinco novenos (5/9) son fracciones".
Un número mixto es lo mismo que tener una suma abreviada, como en el caso de 3¼, que es lo mismo que tener 3+ 1/4. Para expresarlo como fracción lo que debemos entonces es realizar la suma de fracciones.




```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar
//U3 Ejercicio Nº 6
#include <iostream>
using namespace std;

class Fraccion{
protected:
        int numerador, denominador;
public:
        void asignar(int n, int d){ numerador=n; denominador=d;};
        int ver_numerador(){return numerador;};
        int ver_denominador(){return denominador;};
        double convertirADoble(){return (double)numerador/(double)denominador;};
};

class NumeroMixto{
        int entero;
        Fraccion impropia;
public:
        NumeroMixto(int e, int n, int d){entero= e; impropia.asignar(n,d);}
        int ver_entero(){return entero;}
        int ver_numerador_imp(){return impropia.ver_numerador();}
        int ver_denominador_imp(){return impropia.ver_denominador();}
        double convertirADoble(){
```

```
                return (entero+impropia.convertirADoble());}
};

int main(int argc, char *argv[]) {
        NumeroMixto Num(2,3,4);
        Num.convertirADoble();
        cout<<Num.ver_entero()<<" y "<<Num.ver_nu-
merador_imp()<<"/"<<Num.ver_denominador_imp()<<endl;
        cout<<"Decimal: "<<Num.convertirADoble()<<endl;
        return 0;
}
```

Ejercicio 7

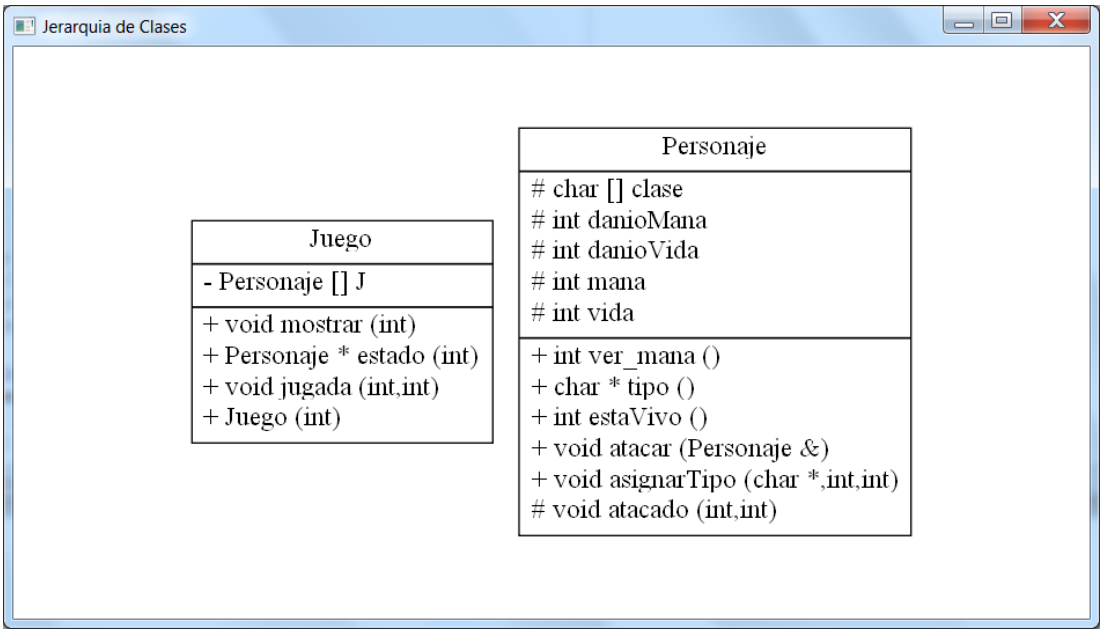
En un videojuego existen tres tipos de personajes: caballero, arquero y mago. Cada uno debe guardar sus cantidades de vida y maná (magia). Todos los personajes tienen un método Atacar() que recibe por referencia otro personaje al que le quitará puntos de vida y mana. Sin embargo cada personaje produce distintas cantidades de daño:

- el caballero quita 6 unidades de vida y ninguna de maná al personaje que ataca.
- el arquero resta 2 puntos de vida y 4 de maná al atacar.
- el mago quita 3 puntos de maná y 3 de vida al adversario al que ataca.

Cada personaje posee además un método *EstaVivo()* que permite saber si el personaje aún tiene algo de energía, y una función *Tipo()* que devuelve una cadena de texto indicando la clase del personaje (caballero, arquero o mago).
Con estas especificaciones **modele las clases** que considere necesarias y cree además una clase *Juego* para representar un videojuego simple que contenga 30 personajes de distinto tipo en único arreglo. Cada personaje deberá elegir a otro al azar y atacarlo (recuerde que ningún personaje puede atacar si no está vivo). Esto se debe repetir hasta que solo quede un personaje vivo. Mostrar de que tipo es el mismo y la posición que ocupa en el vector.

Antes de comenzar a codificar, responda ¿Existen clases abstractas en la jerarquía? En caso de responder afirmativamente, indique cuales, de lo contrario, indique por qué.

(OPCIONAL) Proponga reglas más complejas para el comportamiento de los personajes, por ejemplo, que la acción de atacar consuma unidades de maná al personaje.
(OPCIONAL) Codifique en C++.




```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//U3 Ejercicio Nº 9

#include <iostream>
#include <cstring>
#include <cstdlib>
#include <cstdio>
using namespace std;
//Interfase de Personaje
class Personaje{
protected:
        int vida, mana;//Atributos de energia del Personaje
        int danioVida, danioMana;//Atributos de daño del Personaje
        char clase[10];//Tipo de personaje Caballero, arquero o mago
        void atacado(int, int);//Metodo que decrementa la energia
public:
        void asignarTipo(char *c,int dv, int dm);//Constructor
        void atacar(Personaje &enemigo);//Metodo para restar eergia al enemigo
        int estaVivo();//Devuelve la cantidad de vida
```

```
        char *tipo();//Devuelve el tipo de Personaje
        int ver_mana();//Devuelve cantidad de Mana

};
///*** Interfase de la case Juego *********************************
class Juego{
private:
        Personaje J[30];//Contiene a todos los jugadores
public:
        Juego(int);//Constructor con el parametro de la cantidad de jugadores
        void jugada(int x, int y);//el jugador x ataca al jugador y
        Personaje *estado(int x);//Retorna un puntero al personaje x
        void mostrar(int caJ);//Muestra a todos los jugadores y su energia
};
///--Implementacion de Personaje-----------------------------------
void Personaje::asignarTipo(char *c,int dv, int dm){
        strcpy(clase, c);
        vida= mana= 10;
        danioVida= dv;
        danioMana= dm;
}
void Personaje::atacar(Personaje &enemigo){
        enemigo.atacado(this->danioVida, this->danioMana);
}
void Personaje::atacado(int dv, int dm){
        vida -= dv;
        mana -= dm;
}
int Personaje::estaVivo(){
return vida;
}
char *Personaje::tipo(){
        return clase;
}
int Personaje::ver_mana(){
        return mana;
}

///---- Implementacion de Juego ----------------------------
Juego::Juego(int caJ){//constructor, inicializa el arreglo de jugadores
        for(int x=0; x< caJ; x++){
                switch (x%3){
                case 0: {J[x].asignarTipo("Caballero",6,0);
                        break;};
                case 1: {J[x].asignarTipo("Arquero",2,4);
                        break;};
                case 2: {J[x].asignarTipo("Mago",3,3);
                        break;};
                }
                //cout<<x<<") "<<J[x].tipo()<<" : Vida= "<<J[x].estaVivo()<<" : Mana=
"<<J[x].ver_mana()<<endl;
```

```
        }
}
Personaje *Juego::estado(int x){
        return &J[x];
}
void Juego::jugada(int x, int y){
        J[x].atacar(J[y]);
}

void Juego::mostrar(int caJ){
        //Muestro los jugadores
        for(int m=0; m<caJ; m++){
                cout<<m<<")  "<<"Jugador:  "<<J[m].tipo()<<"  Vida:  "<<J[m].esta-
Vivo()<<endl;
        }
}

int main(int argc, char *argv[]) {
        int caJ=5;//Cantidad de Jugadores
        do{
        cout<<"Ingrese la cantidad de Jugadores (<30): ";cin>>caJ;
        }while(caJ>30);
        Personaje *Jugador1, *Jugador2;//Punteros a jugadores
        Juego Game(caJ);//Inicializo el objeto Game
        Game.mostrar(caJ);//muestro el objeto Game
        //*******************************************************/*/*/
        bool final= false;
        int pos, enemig, contVivos, ganador;
        while(!final){
                do{//Selecciono aleatoriamente el atacante
                        pos= rand()%caJ;
                        Jugador1= Game.estado(pos);
                }while(Jugador1->estaVivo()== 0);//Si esta muerto elige otro vivo
                do{//genero enemigo al azar distinto de posicion del atacante
                        enemig= rand()%caJ;
                }while(pos==enemig);
                cout<<pos<<" - "<<enemig<<endl;
                Jugador2= Game.estado(enemig);
                cout<<pos<<") Jugador: "<<Jugador1->tipo()<<" Vida: "<<Jugador1-
>estaVivo()<<"  ";
                cout<<enemig<<") Enemigo: "<<Jugador2->tipo()<<" Vida: "<<Juga-
dor2->estaVivo()<<endl;
                if ((Jugador1->estaVivo()) != 0){
                        Jugador1->atacar(*Jugador2);
                        cout<<"ATAQUE!!!  "<<enemig<<")  Enemigo:  "<<Jugador2-
>tipo()<<" Vida: "<<Jugador2->estaVivo()<<endl;
                        contVivos=0;
                        for (int k=0; k< caJ; k++){
                                Jugador2=Game.estado(k);
                                if ((Jugador2->estaVivo())>0){
                                        contVivos++;
```

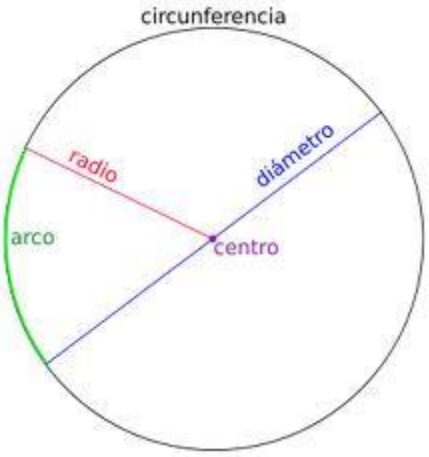
```
                                        ganador= k;
                                }
                        }
                        cout<<"Estan vivos: "<<contVivos<<endl;
                        if (contVivos == 1)
                                final= true;
                }
                else
                        pos++;

        }//Si sale es porque queda un solo jugador vivo
        Jugador2=Game.estado(ganador);
        cout<<"GANADOR "<<ganador<<") "<<Jugador2->tipo()<<": vida= "<<Juga-
dor2->estaVivo()<<endl;
        return 0;
}
```

Continua con Ejercicios adicionales en la próxima página.-

EJERCICIOS ADICIONALES

Ejercicio 3a.1

Defina la clase *Circulo* con atributos y métodos que permitan obtener el área y el perímetro de esta figura. Proponga los constructores y métodos que considere necesarios. Luego, proponga una función amiga externa a la clase Circulo, llamada reducir(), que reciba un objeto de tipo Circulo como parámetro y devuelva otro Circulo con la mitad del diámetro del parámetro, accediendo a los atributos privados o protegidos del parámetro directamente. Utilice todo en un programa C++.
Perímetro = ∏ * 2 * radio - Superficie = ∏ * radio^2

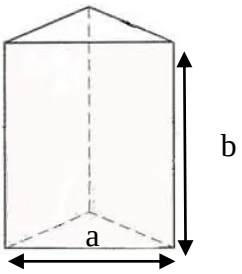



```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//U3 Ejercicio Nº 1

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

class Circulo {
private:
        float radio, perimetro, superficie;
public:
        void asignar_valores(float r1);
        void calcular(void);
        float ver_radio() const {return radio;};
        float ver_perimetro() const {return perimetro;};
        float ver_superficie() const {return superficie;};
        friend Circulo reducir(Circulo);
};
//*** CLASE CIRCULO ***
void Circulo::asignar_valores(float r1)
{ radio = r1;}

void Circulo::calcular(void)
{ perimetro = 2 * radio * M_PI;
superficie= M_PI * pow(radio, 2);
}
//Funcion Amiga
Circulo reducir(Circulo c1){
        Circulo aux;
        aux.asignar_valores(c1.radio/2);
        aux.calcular();
        return aux;
}

//PROGRAMA PRINCIPAL
```

```
int main()
{ Circulo C1, C2; float r, r2;
cout<<" FICH - P.O.O."<<endl<<endl<<endl;
cout << "DATOS DEL Circulo: "<<endl<< endl;
cout << "Ingrese el radio : "; cin>> r;
cout << endl;
C1.asignar_valores(r);
C1.calcular();
cout << "La superficie es : "<< C1.ver_superficie()<<endl;
cout << "El perimatro es  : "<< C1.ver_perimetro()<<endl;
C2= reducir(C1);
cout<<"\nLuego de llamar a la funcion amiga reducir()"<<endl;
cout << "El radio es      : "<<C2.ver_radio()<<endl;
cout << "La superficie es : "<< C2.ver_superficie()<<endl;
cout << "El perimatro es  : "<< C2.ver_perimetro()<<endl;
return 0;
}
```

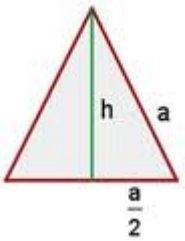
Ejercicio 3a.2

Cree e inicialice dos objetos, uno de tipo Triángulo (equilátero) y otro de Tipo Rectángulo, cada clase consta de atributos y métodos para calcular su superficie. Diseñe un programa que calcule la superficie de un prisma triangular regular, cuyas caras son los objetos anteriores, a través de una función externa amiga acceda a los atributos privados y realice el cálculo, en el programa principal. Ingrese a y b



$h = \sqrt{a^2 - (½\ a)^2}$ SupTriangulo= (½ a*h)/2, SupRectangulo= a*b

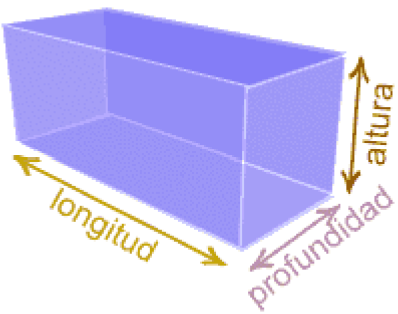
```
/© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//U3 Ejercicio Nº 2
#include <iostream>
#include <cmath>
using namespace std;


class Triangulo;//Debe estar definida para que no de error en la funcion amiga

class Rectangulo{
protected:
        float lado1, lado2, perimetro, superficie;
public:
        void asignar_valores(float l1, float l2);
        void calcular(void);
        float ver_lado1() const {return lado1;};
        float ver_lado2() const {return lado2;};
        float ver_perimetro() const {return perimetro;};
        float ver_superficie() const {return superficie;};
        friend float volumenPrisma(Rectangulo, Triangulo);//Funcion amiga
};

void Rectangulo::asignar_valores(float l1, float l2){
```



```
        lado1= l1; lado2= l2;
}

void Rectangulo::calcular(void){
        perimetro = lado1*2 + lado2*2;
        superficie= lado1 * lado2;
}

class Triangulo{//Es un triangulo equilatero
protected:
        float lado1, altura, superficie, perimetro;
public:
        Triangulo();//Constructor por defecto sin parametros
        //Constructor a partir del lado 1 del Rectangulo
        Triangulo(Rectangulo R){asignar_valores(R.ver_lado1());};
        void asignar_valores(float l1);
        void calcular();
        float ver_lado1() const {return lado1;};
        float ver_altura() const {return altura;};
        float ver_perimetro() const {return perimetro;};
        float ver_superficie() const {return superficie;};
        friend float volumenPrisma(Rectangulo, Triangulo);//Funcion amiga
};

void Triangulo::asignar_valores(float l1){
        lado1 = l1;
        altura= sqrt(pow(lado1,2)-pow((lado1/2),2));
}

void Triangulo::calcular(){
        superficie= (lado1/2)*altura;
        perimetro= lado1* 3;
}

float volumenPrisma(Rectangulo R, Triangulo T){
        R.calcular();T.calcular();
        return R.superfice*3+2 * T.superficie;

}
int main(int argc, char *argv[]) {
        Rectangulo R1;
        R1.asignar_valores(3,4);
        Triangulo T1(R1);
        cout<<" El Volumen del Prisma Triangular es: "<<volumenPrisma(R1,
T1)<<endl;
        return 0;
}
```

Ejercicio 3a.3

La clase PrismaRectangular, consta de tres pares de rectángulos distintos que forman sus caras. Defina atributos y métodos que permitan obtener el área y el volumen de ese cuerpo. Luego otra clase Amiga Cubo que tiene los atributos y métodos para representar un cubo, calcular su área y volumen. Además tiene un método llamado ConvertirEnCubo que recibe un objeto tipo PrismaRectangular y devuelve un Cubo con las dimensiones del primer rectángulo de ese prisma, accediendo a los datos privados o protegidos directamente. Implemente estas clases en un programa C++.




```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//U3 Ejercicio Nº 3

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

//class Cubo;
//*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
class Rectangulo {
protected:
        float lado1, lado2, perimetro, superficie;
public:
        void asignar_valores(float l1, float l2);
        void calcular(void);
        float ver_lado1() {return lado1;};
        float ver_lado2() {return lado2;};
        float ver_perimetro(){return perimetro;};
        float ver_superficie(){return superficie;};
        friend class Cubo;//clase amiga
};
//*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
class PrismaRectangular{
private:
        Rectangulo R1, R2, R3;
        float superficie, volumen;
public:
        PrismaRectangular(float altura, float profundidad, float longitud);
        float ver_superficie(){return superficie;};
        float ver_volumen(){return volumen;};
        friend class Cubo;//clase amiga
};
//*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
class Cubo{
        float arista, superficie, volumen;
public:
```

```
        Cubo(float a){arista= a; superficie= pow(arista,2)*6; volumen= pow(arista,3);};
        float ver_superficie(){return superficie;};
        float ver_volumen(){return volumen;};
        Cubo convertirEnCubo(PrismaRectangular PReg);

};
//*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
void Rectangulo::asignar_valores(float l1, float l2){
        lado1= l1; lado2= l2;
        calcular();
}

void Rectangulo::calcular(void){
        perimetro = lado1*2 + lado2*2;
        superficie= lado1 * lado2;
}


PrismaRectangular::PrismaRectangular(float altura, float profundidad, float longitud){
        R1.asignar_valores(altura, profundidad);
        R2.asignar_valores(longitud, altura);
        R3.asignar_valores(longitud, profundidad);
        superficie = R1.ver_superficie() * 2 + R2.ver_superficie()* 2 + R3.ver_superfi-
cie()* 2;
        volumen= R1.ver_superficie()*R2.ver_lado1();
}

Cubo Cubo::convertirEnCubo(PrismaRectangular PReg){
        Cubo aux(PReg.R1.lado1);
        return aux;
}

int main(int argc, char *argv[]) {
        PrismaRectangular PR(3,4,5);
        cout<<" El Superficie del Prisma Rectangular es: "<<PR.ver_superfi-
cie()<<endl;
        cout<<" El Volumen del Prisma Rectangular es: "<<PR.ver_volumen()<<endl;
        Cubo C(0);
        C= C.convertirEnCubo(PR);
        cout<<" El Superficie del Cubo es: "<<C.ver_superficie()<<endl;
        cout<<" El Volumen del Cubo es: "<<C.ver_volumen()<<endl;
        return 0;
}
```

Ejercicio 3a.4

Explique el siguiente código

```
// constructores y clases derivadas
```

```
#include <iostream.h>

class madre {
  public:
    madre ()
      { cout << "madre: sin parámetros\n"; }
    madre (int a)
      { cout << "madre: parámetro int\n"; }
};

class hija : public madre {
  public:
    hija (int a)
      { cout << "hija: parámetro int\n\n"; }
};

class hijo : public madre {
  public:
    hijo (int a) : madre (a)
      { cout << "hijo: parámetro int\n\n"; }
};

int main () {
  hija cynthia (1);
  hijo daniel(1);

  return 0;
}
```

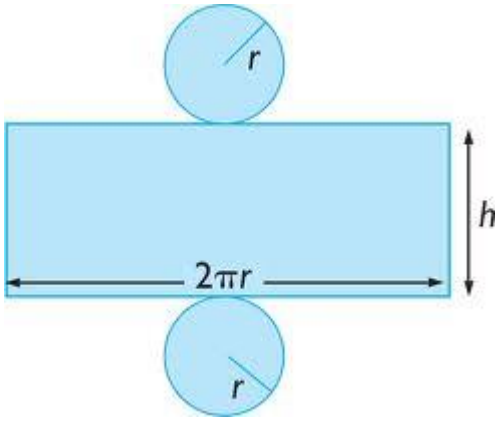
Aunque el constructor y el destructor de la clase base no son heredados, el constructor por defecto (constructor sin parámetros) y el destructor de la clase base son siempre llamados cuando un nuevo objeto de una clase derivada es creado o destruido.

Si la clase base no tiene constructor por defecto o se desea que un constructor sobrecargado sea llamado cuando un nuevo objeto de la clase derivada es creado, se puede especificar en cada definición de un constructor de la clase derivada:

Nombre_clase_derivada (parametros) : nombre_clase_base (parametros) {}

Ejercicio 3a.6

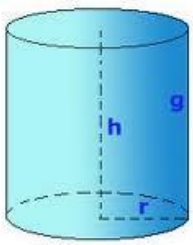
Defina las clases *Circulo* y *Rectangulo* con atributos y métodos que permitan obterner el área de cada una de estas figuras. Los atributos deben inicializarse con el constructor. Proponga los constructores y métodos que considere necesarios. Luego, proponga la clase *Cilindro* reutilizando una o ambas clases anteriores por herencia multiple. Los atributos deben inicializarse con el constructor. La clase *Cilindro* debe poseer un método *ObtenerVolumen()* que permita obtener el volumen del cuerpo. Escriba un programa cliente en C++ que permita ingresar el radio y la altura del de un cilindro y permita obtener su volumen.




```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//U3 Ejercicio Nº 3a.6

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

class Circulo {
private:
        float radio, perimetroC, superficieC;
public:
        Circulo(float r1);
        void calcularC(void);
        float ver_radio() const {return radio;};
        float ver_perimetroC() const {return perimetroC;};
        float ver_superficieC() const {return superficieC;};
};

class Rectangulo{
private:
        float lado1, lado2, perimetroR, superficieR;
public:
        Rectangulo(float l1, float l2);
        void calcularR(void);
        float ver_lado1() const {return lado1;};
        float ver_lado2() const {return lado2;};
        float ver_perimetroR() const {return perimetroR;};
        float ver_superficieR() const {return superficieR;};
};

class Cilindro: public Circulo, Rectangulo{
public:
        Cilindro(float, float);
        float calcularVolumen();
```

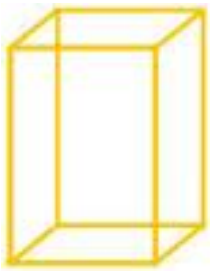
```
};

//*** CLASE CIRCULO ***
Circulo::Circulo(float r1){
        radio = r1;
}
void Circulo::calcularC()
{ perimetroC = 2 * radio * M_PI;
superficieC= M_PI * pow(radio, 2);
}



//*** CLASE TRectanguo ***
Rectangulo::Rectangulo(float l1, float l2)
{       lado1= l1;
lado2= l2;
}
void Rectangulo::calcularR(void)
{       perimetroR= 2*(lado1 + lado2);
superficieR= lado1 * lado2;
}
//***CLASE CILINDRO ***
Cilindro::Cilindro(float r, float a):Circulo(r),Rectangulo(ver_perimetroR(),a)
{
}
float Cilindro::calcularVolumen(){
        calcularC();
        calcularR();
        return (2*ver_superficieC()+ver_superficieR());
}


int main(int argc, char *argv[]) {
        float radio, altura;
        cout<<"Programacion Orientada a Objetos - F.I.C.H. - U.N.L."<<endl<<endl;
        cout<<"Calculo del Volumen del cilindro"<<endl<<endl;
        cout<<"U3 Ejercicio N 10: "<<endl<<endl;
        cout<<"Ingrese el radio:  ";cin>>radio;
        cout<<"Ingrese la altura: ";cin>>altura;
        Cilindro C1(radio, altura);
        cout<<"Volumen del Cilindro: "<<C1.calcularVolumen()<<endl;
        return 0;
}
```

Ejercicio 3a.7



Construya una clase base abstracta **Cuerpo** que tenga los atributos a, b y volumen, además un método virtual puro calcular_volumen(). En las clases hijas **Prismacuadrado y Cilindro** implemente el método virtual calcular_volumen(). En el programa principal cree instancias de **Prisma** y **Cilindro**, cree un puntero de

tipo **Cuerpo**, con este puntero inicialice y calcule el volumen de un **prisma cuadrado**. Con el mismo puntero inicialice y calcule el volumen de un **cilindro**.



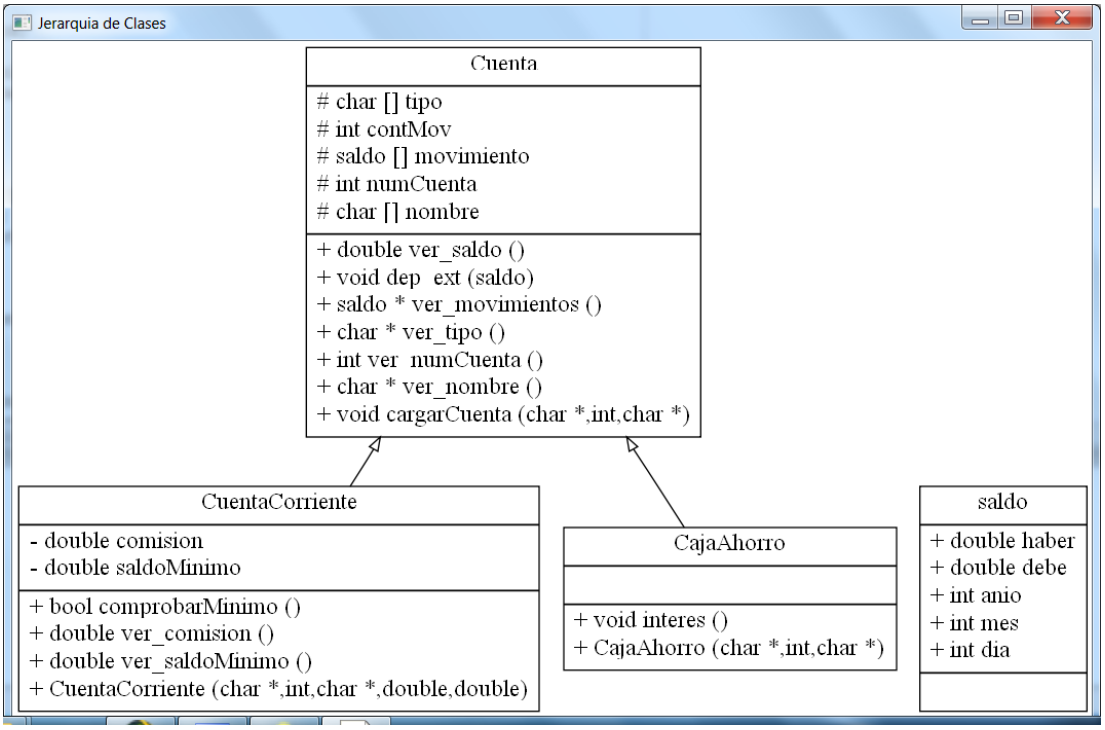

```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//U3 Ejercicio Nº 11

#include <iostream>
#include <cmath>
using namespace std;
//Clase base abstracta, no se instancia
class CCuerpo{
protected:
        float a, b, volumen;
public:
        void cargar_datos(float, float);
        //Método virtual puro
        virtual void calcular_volumen()=0;
        float ver_volumen() {return volumen;};
};
//CPrismaCuadrado herencia simple de CCuerpo
class CPrismaCuadrado: public CCuerpo{
        public: void calcular_volumen(){volumen= a*a*a;};
};
//CCilindro herencia simple de CCuerpo
class CCilindro: public CCuerpo{
        public:    void calcular_volumen() {volumen= M_PI * pow(b,2)*a;};
};

void CCuerpo::cargar_datos(float dato1, float dato2){
        a= dato1;
        b= dato2;
}
```

```
//-------CUERPO PRINCIPAL DEL PROGRAMA------------
        int main()      {
                CPrismaCuadrado PC;
                CCilindro Cil;
                CCuerpo *p; //para aplicar polimorfismo.-
                int a, b;
                cout<<" FICH - P.O.O.- (c) Prof. G. Sas. "<<endl;
                cout<<" "<<endl;
                cout<<" *** C I L I N D R O *** "<<endl;
                cout << "Ingrese el radio en cm: "; cin>> b;
                cout << "Ingrese el alto en cm: "; cin>> a;
                p= &Cil;
                p->cargar_datos(a,b);
                p->calcular_volumen();
                cout<<"Volumen del Cilindro es: "<<p->ver_volumen()<<"
cm2"<<endl;
                cout<<" "<<endl;
                cout<<" *** PRISMA CUADRADO (cubo)*** "<<endl;
                cout << "Ingrese el lado en cm: "; cin>> a;
                p= &PC;
                p->cargar_datos(a,a);
                p->calcular_volumen();
                cout<<"Volumen del Prisma Cuadrado es: "<<p->ver_volumen()<<"
cm2"<<endl;
}
```

Ejercicio 3a.8 (OPCIONAL)

Un banco tiene dos tipos de cuentas para los clientes; unas son cuentas de ahorro y las otras cuentas corrientes. Las cuentas de ahorro tienen un interés compuesto y la posibilidad de reintegro, pero no admiten talonarios de cheques. Las cuentas corrientes ofrecen talonarios de cheques, pero no tienen interés. Los titulares de una cuenta corriente también deben mantener un saldo mínimo; si el saldo desciende por debajo de este nivel, se les cobra una comisión por el servicio.
Cree una clase *Cuenta* que almacene: el nombre del titular, el número de cuenta y el tipo de cuenta. A partir de ésta, derive las clases *CuentaCorriente* y *CuentaAhorro* para adaptarlas a los requerimientos específicos. Incluya las funciones miembro necesarias para realizar las siguientes tareas:

a) Aceptar un ingreso de un titular y actualizar el saldo.
b) Mostrar el saldo.
c) Calcular y abonar los intereses.
d) Permitir un reintegro y actualizar el saldo.
e) Comprobar que el saldo no esté por debajo del mínimo, imponer la sanción si es necesario, y actualizar el saldo.




```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//U3 Ejercicio Nº 8

#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
typedef struct {
        int dia,mes,anio;
        double debe, haber;
        } saldo;

class Cuenta{
protected:
        char nombre[30];
        int numCuenta;
        saldo movimiento[100];
        int contMov;
        char tipo[2];
public:
        void cargarCuenta(char *nom, int numCu, char *tip){strcpy(nombre,nom);

        numCuenta= numCu;

        strcpy(tipo, tip);
```

```
        contMov= 0;};
        char *ver_nombre(){return nombre;};
        int ver_numCuenta(){return numCuenta;};
        char *ver_tipo(){return tipo;};
        saldo *ver_movimientos(){return movimiento;};
        void dep_ext(saldo mov);//Deposito o extraccion de fondos
        double ver_saldo(){
                double sumD=0, sumH=0;
                for(int x=0; x < contMov; x++){
                        sumD += movimiento[x].debe;
                        sumH += movimiento[x].haber;
                }
                return (sumD - sumH);
        }

};
///************************************************
class CajaAhorro: public Cuenta{
public:
        CajaAhorro(char *nom, int numCu, char *tip){
                cargarCuenta(nom, numCu, tip);};
        void interes();
};
///************************************************
class CuentaCorriente: public Cuenta{
private:
        double saldoMinimo, comision;
public:
        CuentaCorriente(char *nom, int numCu, char *tip, double sm=1000, double
com=0.1){
                                                                cargar-
Cuenta(nom, numCu, tip);
                                                                saldoMi-
nimo= sm;
                                                                comision=
com;};
        double ver_saldoMinimo(){return saldoMinimo;};
        double ver_comision(){return comision;};
        bool comprobarMinimo(){if(ver_saldo() > saldoMinimo) return false;};
};
int main(int argc, char *argv[]) {
        char n[30];char t[2];int nu;double deposito;
        cout<<"Ingrese el Nombre del Cliente: ";gets(n);
        cout<<"Ingrese el tipo de cuenta (CC, CA): ";cin>>t;
        cout<<"Ingrese el Numero de Cuenta : ";cin>>nu;
        cout<<"Ingrese el Deposito + o Retiro -: ";cin>>deposito;
        //if (strcmp(t,"CC")==0)
                CuentaCorriente C(n,nu,t,1000,0.1);
        //else
        //      CajaAhorro C(n,nu,t);
```

```
        cout<<"Se creo exitosamente la cuenta "<<C.ver_nombre()<<" N: "
                <<C.ver_numCuenta()<<" Tipo: "<<C.ver_tipo()<<endl;
        return 0;
}
```

Ejercicio 3a.9 (OPCIONAL)

a) Crear una clase llamada **Calcu**, esta debe tener un constructor que reciba dos valores enteros (a y b) como argumentos para realizar operaciones y debe tener funciones públicas para sumar, restar, dividir, multiplicar, elevar a una potencia (de cualquier grado) y para modificar dichos números. La función pública que calcula potencias debe llamar a una función auxiliar privada que le permita multiplicar "n" veces un número pasado como argumento.

b) Derivar de **Calcu** una clase **Calcu_cientifica** que contenga métodos públicos para calcular las funciones trascendentes: seno, coseno y tangente.
NOTA: considere que el lenguaje de programación a emplear carece de estas funciones trigonométricas por lo que el seno debe calcularse aproximadamente con las expresiones:

seno(x) = $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots\ldots\ldots\ldots$; (*tomar 5 términos*)

cos(x)= 1- seno(x);
tan(x)= seno(x)/cos(x);

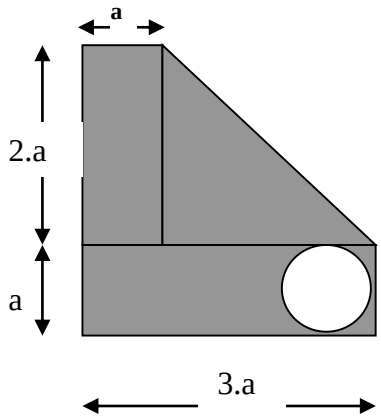
```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//U3 Ejercicio A Nº 9

#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

class TCalcu { //Clase Base
private:
        double a, b;
        int pot(float base, int exponente);
public:
        TCalcu(double x, double y): a(x), b(y){};//Constructor con 2 parametros
        TCalcu() {};//Constructor por defecto
        ~TCalcu(){};//Destructor por defecto
        void asignar_valores(int a1, int b1);
        int suma() {return(a+b);};
        int resta() {return(a-b);};
        int multiplicacion() {return(a*b);};
        float division() {return(a/b);};
        int potencia(float x, int n) {return pot(x,n);};
};

class TCalcu_cientifica: public TCalcu {
private:
```

```
        float alfa;
        float factorial(int num);
public:
        void asignar_angulo(float alfa2);
        float seno();
        float coseno();
        float tangente();
};
//*** CLASE TCalcu ****
void TCalcu::asignar_valores(int a1, int b1)
{ a= a1; b= b1;
}
int TCalcu::pot(float base, int exponente)
{int sum=1;
for (int k=0; k < exponente; k++){
        sum *= base;};
return sum;
}
//**** CLASE TCalcu_cientifica ****
void TCalcu_cientifica::asignar_angulo(float alfa2)
{ alfa= alfa2;
}
float TCalcu_cientifica::factorial(int num)
{int f=1;
for (int x=1; x< num; x++)
        f= f* x;
return f;
}
float TCalcu_cientifica::seno()
{int n, sen= alfa;
for (n=3; n<11; n+=2)
        sen += ((potencia(alfa,n)/factorial(n)))*(-1);
return sen;
}
float TCalcu_cientifica::coseno()
{return(1 - seno());
}
float TCalcu_cientifica::tangente()
{ return(seno()/coseno());
}

//***** CUERPO PRINCIPAL DEL PROGRAMA ***
int main()
{TCalcu_cientifica Cientifica;
int aa, bb;
float alfa1;
cout<<" FICH - P.O.O.- J.T.P. PROF. GERARDO SAS"<<endl;
cout<<"                                        "<<endl;
cout<<"                                        "<<endl;
cout << "DATOS DE LOS NUMEROS PARA HACER LOS CALCULOS."<<endl<< endl;
cout << "Ingrese A: "; cin>> aa;
cout << "Ingrese B: "; cin>> bb;
cout << "Ingrese el angulo Alfa: "; cin>> alfa1;
Cientifica.asignar_valores(aa, bb);
Cientifica.asignar_angulo(alfa1);
cout << "SUMA     (A + B)= "<< Cientifica.suma()<<endl;
```

```
cout << "RESTA    (A - B)= "<< Cientifica.resta()<<endl;
cout << "PRODUCTO (A * B)= "<< Cientifica.multiplicacion()<<endl;
cout << "DIVISION (A / B)= "<< setprecision(5)<<setw(8)<< Cientifica.division()<<endl;
cout << "           B" << endl;
cout << "POTENCIA (A)    = "<< Cientifica.potencia(aa,bb)<<endl<<endl;

cout << "Seno(Alfa)= "<< Cientifica.seno() <<endl;
cout << "Coseno(Alfa)= "<< Cientifica.coseno() <<endl;
cout << "Tangente(Alfa)= "<< Cientifica.tangente() <<endl;

}
```

Ejercicio 3a.10 **(OPCIONAL)**

Reutilice las clases Rectangulo y Circulo para componer una clase Figura que represente a la imagen de la derecha.
La clase debe tener –entre otros- un método que obtenga el área sombreada de la figura.
Pruebe la clase en un programa C++ cliente.
La solución se puede pensar con 2 o 3 rectángulos y 1 circulo.




```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//U3 Ejercicio Nº 10

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

class TCirculo {
private:
        float radio, perimetro, superficie;
public:
        void asignar_valores(float r1);
        void calcular(void);
        float ver_radio() const {return radio;};
        float ver_perimetro() const {return perimetro;};
        float ver_superficie() const {return superficie;};
};
```

```
class TRectangulo{
private:
        float lado1, lado2, perimetro, superficie;
public:
        void asignar_valores(float l1, float l2);
        void calcular(void);
        float ver_lado1() const {return lado1;};
        float ver_lado2() const {return lado2;};
        float ver_perimetro() const {return perimetro;};
        float ver_superficie() const {return superficie;};
};

class TFigura{
private:
        TCirculo C4;
        TRectangulo R1, R2, R3;
public:
        void asignar_valores(float);
        float calcular_superficie();
};

//*** CLASE TCIRCULO ***
void TCirculo::asignar_valores(float r1)
{ radio = r1;}
void TCirculo::calcular(void)
{ perimetro = 2 * radio * M_PI;
superficie= M_PI * pow(radio, 2);
}



//*** CLASE TRectanguo ***
void TRectangulo::asignar_valores(float l1, float l2)
{ lado1= l1;
lado2= l2;
}
void TRectangulo::calcular(void)
{ perimetro= 2*(lado1 + lado2);
superficie= lado1 * lado2;
}

void TFigura::asignar_valores(float a1)
{R1.asignar_valores(2,(2*a1));
R1.calcular();
R2.asignar_valores(a1,(3*a1));
R2.calcular();
R3.asignar_valores((2*a1),(2*a1));
R3.calcular();
C4.asignar_valores(a1);
C4.calcular();
}
```

```
float TFigura::calcular_superficie()
{ float superficie=
        R1.ver_superficie()+R2.ver_superficie()+
        R3.ver_superficie()/2- C4.ver_superficie();
return superficie;
}

//**************** Progra principal ****************
int main(int argc, char* argv[])
{ TFigura F;
int a2=0;
cout<<" FICH - P.O.O.- "<<endl;
cout<<"                                    "<<endl;
cout<<"                                    "<<endl;
cout << "DATOS DE LA FIGURA"<<endl<< endl;
cout << "Ingrese el lado a en cm: "; cin>> a2;
cout << endl << endl << endl;
F.asignar_valores(a2);
F.calcular_superficie();
cout << "La superficie es: "<< F.calcular_superficie()<<" cm2"<<endl;

return 0;
}
//(c) J.T.P. Prof. Gerardo Sas
```

Ejercicio 3a.11 **(OPCIONAL)**

a) ¿Cuál es la salida del siguiente programa? Se ejecuta el método del hijo
b) ¿Qué sucede si quita la palabra virtual? Se ejecuta el método del padre
c) Analice el resultado obtenido en cada caso y comente.

```
class B{//clase padre
public:
        virtual void m() {cout<<"B::m()"<<endl;};
};

class D : public B {//clase hija
public:
        void m() {cout<<"D::m()"<<endl;};
};

int main() {
        D d1;
        B * p=&d1;
        p->m();
}
```

a) D::m()

b) B::m()

Si se antepone virtual al metodo del padre cuando se ejecuta desde un objeto hijo, es el metodo del hijo el que se ejecuta, sino se antepone virtual, será el método del padre el que se ejecute.-

Ejercicio 3a.12 **(OPCIONAL)**
Aplicando el concepto de herencia y polimorfismo: a) Proponga la clase CPoligono y como clases derivadas CRectangulo y CTriangulo. Estas clases son empleadas en el programa del recuadro.
a) Proponga una función virtual pura para el método area() de la clase base CPoligono.
b) Proponga un método virtual area() de las clases derivadas de CPoligono.
c) Cuál es la salida del programa si no se emplea la palabra virtual en el método area()?.

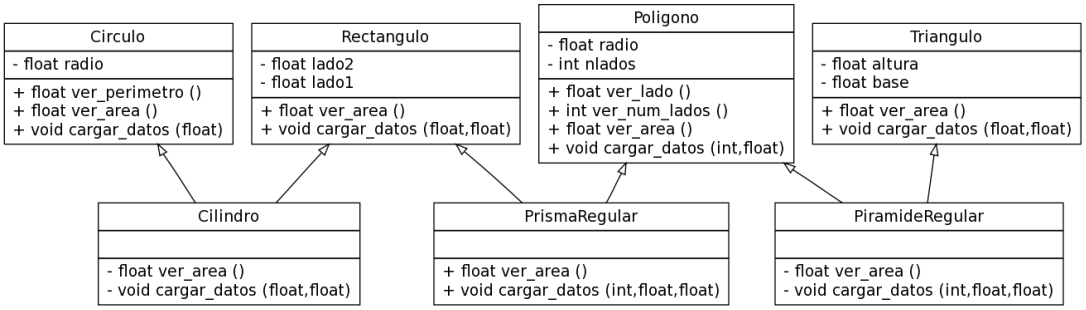
```
int main () {
  CRectangulo rect;
  CTriangulo trgl;
  CPoligono * p1 = &rect;
  CPoligono * p2 = &trgl;
  p1->fijar_valores (3,5);
  p2->fijar_valores (4,6);
  cout << p1->area() << endl;
  cout << p2->area() << endl;
  return 0;
}
```

```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//U3 Ejercicio Nº 12

#include <iostream>
using namespace std;

class CPoligono {// clase base
protected:
        int ancho, largo;
public:
        void fijar_valores (int a, int b)
        { ancho=a; largo=b;};
        virtual int area (void) =0;//virtual pura
};
class CRectangulo: public CPoligono {
public:
        int area (void)
        { return (ancho * largo); }
};
class CTriangulo: public CPoligono {
```

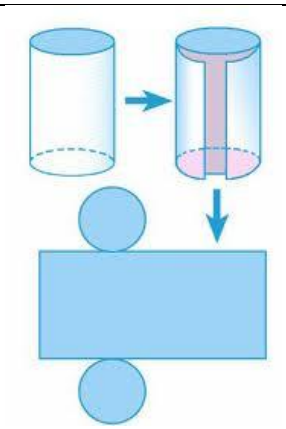
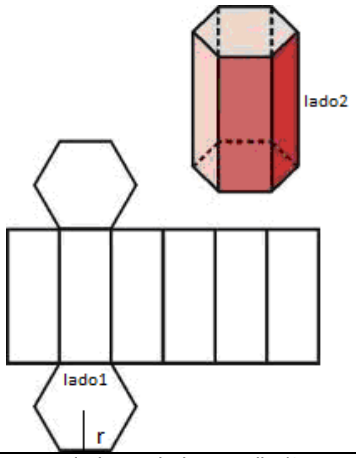
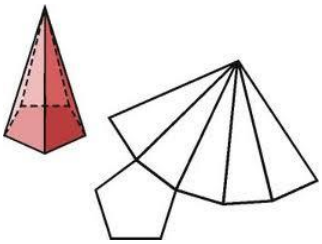
```
public:
        int area (void)
        { return (ancho * largo / 2); }; };

//Programa Principal
int main (void) {
        CRectangulo rect;
        CTriangulo trgl;
        CPoligono * p1 = &rect;
        CPoligono * p2 = &trgl;
        p1->fijar_valores (3,5);
        p2->fijar_valores (4,6);
        cout << rect.area() << endl;
        cout << trgl.area() << endl;
        cout << p1->area() << endl;
        cout << p2->area() << endl;
        //(c)Prof. Gerardo Sas.
}

//Si no se hubiera definido area() como virtual el resultado seria 0.
```

Ejercicio 3a.13 **(OPCIONAL)**

Considere una jerarquía de clases con herencia múltiple como la de la figura. Implemente dicha jerarquía y un programa C++ cliente para probarla construyendo objetos de tipo Cilindro, PrismaRegular y PirámideRegular.



Las formula para el área de un polígono regular es: long_lado*aux/2*nlados, donde aux=radio*cos(2*M_PI/nlados/2), y long_arista=2*radio*sin(2*M_PI/nlados/2).

CILINDRO	PRISMA REGULAR	PIRAMIDE REGULAR
		
Datos: radio, lado1 AreaCirculo= PI * (radio)^2 AreaRect= lado1*(PI*2*radio) AreaCil= 2*AreaCirculo+Area-Rect	Base= lado1 * nlados * radio /2 SupCara= lado1 * lado2; SupPrism= SupCara * nlados + 2 * base.-	SupBase= lado1 * nlados * radio /2 SupCara= lado1 * altura / 2 SupPiram= SupCara * nlados + supBase.-

```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
//Autor: Prof. Gerardo Sas.
//sitio web: http://e-fich.unl.edu.ar/moodle/
//Guia de estudios Nº 3 - Ejercicio Nº 13
#include<iostream>
#include<math.h>
#include <cmath>
using namespace std;

class TCirculo {
protected:
        float radio;
public:
        void cargar_datos(float r1);
        float ver_area();
        float ver_perimetro();
};
class TRectangulo{
protected:
        float lado1, lado2;
public:
        float ver_area();
        void cargar_datos(float l1, float l2);
};
class TPoligono{
protected:
        float radio;
        int nlados;
public:
        void cargar_datos(float r, int nla);
        float ver_radio();
```

```
        int ver_nlados();
        float ver_area();
};
class TTriangulo {
protected:
        float base, altura;
public:
        void cargar_datos(float,float);
        float ver_area();
};
class TCilindro: public TCirculo, public TRectangulo {
public:
        void cargar_datos(float a1,float r1);
        float ver_area();
};
class TPrismaRegular: public TRectangulo, public TPoligono{
public:
        void cargar_datos(int,float,float);
        float ver_area();
};
class TPiramideRegular: public TPoligono, public TTriangulo{
public:
        void cargar_datos(int,float,float);
        float ver_area();
};
//Fin de la Interfase
//Comienzo de la Implementacion
//*** CLASE TCIRCULO ***
void TCirculo::cargar_datos(float r1){radio = r1;}
float TCirculo::ver_area(){return M_PI*pow(radio,2);}
float TCirculo::ver_perimetro(){return M_PI*2*radio;}
//*** CLASE TRectangulo ***
float TRectangulo::ver_area(){return lado1*lado2;}
void TRectangulo::cargar_datos(float l1, float l2){lado1=l1;lado2=l2;}
//*** CLASE TPoligono ***
void TPoligono::cargar_datos(float r, int nla){radio=r;nlados=nla;}
float TPoligono::ver_radio(){return radio;}
int TPoligono::ver_nlados(){return nlados;}
float TPoligono::ver_area(){float aux=radio*cos(2*M_PI/nlados/2);
                                                        float        long_arista=2*ra-
dio*sin(2*M_PI/nlados/2);
                                                        return      (long_arista*aux/2*nla-
dos);}
//CLASE TTriangulo
void TTriangulo::cargar_datos(float b,float a){base = b; altura= a;}
float TTriangulo::ver_area(){return (base*altura/2);}
//*** CLASE TCilindro ***
void TCilindro::cargar_datos(float a1,float r1){//Altura y radio
        TCirculo::cargar_datos(r1);
        TRectangulo::cargar_datos(a1,ver_perimetro());
}
```

```
float TCilindro::ver_area(){return TCirculo::ver_area()*lado1;}
//CLASE TPrismaRegular: Hereda de TRectangulo y TPoligono
void TPrismaRegular::cargar_datos(int nlad,float rad,float alt){//Num Lados, radio, al-
tura
        float long_arista=2*rad*sin(2*M_PI/nlad/2);
        TRectangulo::cargar_datos(long_arista,alt);
        TPoligono::cargar_datos(rad, nlad);}
float     TPrismaRegular::ver_area(){return     (2*TPoligono::ver_area()+TRectan-
gulo::ver_area()*nlados);}
//CLASE TPiramideRegular
void TPiramideRegular::cargar_datos(int nlad,float rad,float alt){//Num Lados, radio,
altura
        float long_arista=2*rad*sin(2*M_PI/nlad/2);
        TTriangulo::cargar_datos(long_arista,alt);
        TPoligono::cargar_datos(rad, nlad);}
float     TPiramideRegular::ver_area(){return     (TPoligono::ver_area()+TTrian-
gulo::ver_area()*nlados);}


//PROGRAMA PRINCIPAL
int main()
{ float r, a;
int nl;
TCilindro Cil;
cout<<" FICH - P.O.O."<<endl<<endl<<endl;
cout << "DATOS DEL C I L I N D R O."<<endl;
cout << "Ingrese el radio : "; cin>> r;
cout << "Ingrese la altura: "; cin>> a;
Cil.cargar_datos(a,r);
cout<<"Volumen del Cilindro= "<<Cil.ver_area()<<endl;
TPrismaRegular P;
cout << endl << endl;
cout << "DATOS DEL PRISMA REGULAR."<<endl;
do{
        cout << "Ingrese Num Lados ( mayor que 2) : "; cin>> nl;}
while (nl<3);
cout << "Ingrese el radio : "; cin>> r;
cout << "Ingrese la altura: "; cin>> a;
cout << endl << endl;
P.cargar_datos(nl,r,a);
cout<<"Area del Cilindro= "<<P.ver_area()<<endl;
TPiramideRegular PR;
cout << endl << endl;
cout << "DATOS DE LA PIRAMIDE REGULAR."<<endl;
do{
        cout << "Ingrese Num Lados ( mayor que 2) : "; cin>> nl;}
while (nl<3);
cout << "Ingrese el radio : "; cin>> r;
cout << "Ingrese la altura: "; cin>> a;
cout << endl << endl;
PR.cargar_datos(nl,r,a);
```

```
cout<<"Area de la Piramide= "<<PR.ver_area()<<endl;
return 0;
}
```

Cuestionario

1. ¿Qué significa herencia?
   La herencia en C++ es un mecanismo de abstracción creado para poder facilitar y mejorar el diseño de las clases de un programa. Con ella se pueden crear nuevas clases a partir de clases ya hechas, siempre y cuando tengan un tipo de relación especial. En la herencia, las clases derivadas "heredan" los datos y las funciones miembro de las clases base, pudiendo las clases derivadas redefinir estos comportamientos (polimorfismo) y añadir comportamientos nuevos propios de las clases derivadas.

2. ¿A qué se denominan clase base y clase heredada?
   C++ permite heredar a las clases características y conductas de una o varias clases denominadas base. Las clases que heredan de clases base se denominan derivadas, estas a su vez pueden ser clases bases para otras clases derivadas. Se establece así una clasificación jerárquica.
3. ¿Cuándo se utiliza la etiqueta *protected* en un miembro de una clase?
   Para no romper el principio de encapsulamiento (ocultar datos cuyo conocimiento no es necesario para el uso de las clases), se proporciona un nuevo modo de visibilidad de los datos/funciones: "protected". Cualquier cosa que tenga visibilidad protected se comportará como pública en la clase Base y en las que componen la jerarquía de herencia, y como privada en las clases que NO sean de la jerarquía de la herencia.
4. ¿Qué es herencia múltiple?
   La herencia múltiple es el mecanismo que permite al programador hacer clases derivadas a partir, no de una sola clase base, sino de varias.
5. ¿Pueden crearse instancias de una clase base?
   SI
6. ¿Para que sirve la palabra reservada *virtual*?
   Una función virtual o método virtual es una función cuyo comportamiento, al ser declarado "virtual", es determinado por la definición de una función con la misma cabecera en alguna de sus subclases.
7. ¿Qué es una clase abstracta?
   Una clase abstracta, o clase base abstracta, es una que está diseñada sólo como clase padre de las cuales se deben derivar clases hijas. Una clase abstracta se usa para representar aquellas entidades o métodos que después se implementarán en las clases derivadas, pero la clase abstracta en sí no contiene ninguna implementación, solamente representa los métodos (al menos uno virtual puro) que se deben implementar. Por ello, no es posible instanciar una clase abstracta, pero sí una clase concreta que implemente los métodos definidos en ella.
8. ¿Qué es un método virtual?¿Y un método virtual puro?
   Ver respuesta 6, sintaxis virtual tipo nombre_metodo() = 0; Debe notarse que el = 0 es la notación que emplea C++ para definir funciones virtuales puras.
9. ¿Qué significa agregación o inclusión?
   - una clase contiene como atributo una o más instancias de otra.
   - se dice que una clase está compuesta por otras
   - se identifica cuando una clase es parte de otra
10. ¿En qué se diferencian agregación y herencia?

Estos son los mecanismos esenciales para evitar la repetición de código y permitir la reusabilidad, y por tanto la reutilización de software, que es una de las características más importantes de la programación orientada a Objetos. En C++, el efecto de la reutilización de código son similares tanto en la composición como en la herencia (lo cual tiene sentido, pues ambas son dos formas de crear nuevos tipos utilizando tipos ya existentes). Sin embargo **su uso, funcionamiento, declaración e implementación son diferentes**. Ver respuestas 1, 9 y 13.-

11. Un método abstracto, ¿es siempre virtual?. ¿Y uno virtual siempre abstracto?
    Métodos abstractos: llamamos así a las funciones virtuales puras en C++.
12. ¿Qué significa polimorfismo?¿Y qué es invocación polimórfica en C++?
    C++ nos permite acceder a objetos de una clase derivada usando un puntero a la clase base. En esa capacidad es posible el polimorfismo. Por supuesto, sólo podremos acceder a datos y funciones que existan en la clase base, los datos y funciones propias de los objetos de clases derivadas serán inaccesibles.
13. Antes de comenzar a codificar, ¿cómo reconoce que dos clases conforman una relación de herencia?¿Cómo reconoce que dos clases pueden componerse mediante una relación de agregación?
    Cuando necesito operar con varios objetos de la clase base en la clase derivada, entonces utilizo agregación, de lo contradio herencia.-