

UNIDAD 2

INTRODUCCIÓN A LA PROGRAMACION ORIENTADA A OBJETOS

RESPUESTAS A LOS EJERCICIOS CORRESPONDIENTES A LA UNIDAD 2

Ejercicio 2.1

Diseñe una clase Cilindro que modele un cilindro con el objetivo de calcular el volumen del cuerpo conociendo el radio y la altura del mismo.

- a) Cree los métodos AsignarDatos() y CalcularVolumen() para asignar los datos del problema y calcular el volumen del cuerpo.
- b) Escriba un programa cliente que utilice dicha clase. Defina 2 instancias de Cilindro llamadas c1 y c2. El objeto c1 debe utilizar datos ingresados por el usuario, mientras que para c2 utilice 5.3 cm y 10.2 cm para el radio y la altura respectivamente.
- c) Agregue un constructor a la clase Cilindro que reciba 2 parámetros para inicializar el radio y la altura. Luego intente compilar nuevamente el programa.
¿Puede explicar por qué se produce el error?
- d) Elija y realice alguna de las 2 siguientes modificaciones:
 - Cree un constructor que no reciba parámetros e inicialize el radio y la altura en 0.
 - Modifique el constructor de Cilindro para emplear parámetros por defecto.

Intente compilar el programa nuevamente. ¿Se producen errores de compilación esta vez? ¿Por qué?

Respuesta:

Paso 1: Análisis.

Usted habrá reconocido rápidamente los datos y resultado del problema: *radio*, *altura* y *volumen del cilindro*. También en esta etapa encontramos las relaciones entre datos y resultados:

El volumen del cilindro es:

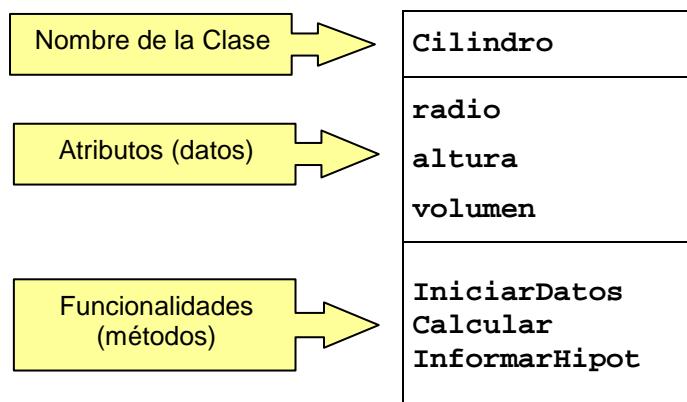
$$\begin{aligned} \text{Volumen Cilindro} &= \text{Superficie de la base (circulo)} * \text{altura} \\ \text{Superficie de la base} &= \pi * \text{radio}^2 \end{aligned}$$

Paso 2: Hallar Clases y responsabilidades

La entidad que realmente hay que modelar es el **cilindro**. Las otras tres entidades (radio, altura y volumen) bien pueden ser atributos del Cilindro y ser modelados simplemente como un número real.

Por otro lado se deberá calcular el volumen a partir del radio y la altura, por lo tanto, la clase Cilindro deberá poseer una funcionalidad (método) para *calcular el volumen*. Finalmente debemos considerar funcionalidades básicas para asignar los atributos e informar el volumen del cilindro.

Resumimos este diseño sencillo mediante un esquema como el siguiente:



CLASE CILINDRO	
Atributos	radio, altura, volumen.
Métodos	asignar_datos() calcular_volumen(); Ver_radio() Ver_altura() Ver_volumen()

Respuesta item a y 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/  
//Guia de estudios N° 2 - Ejercicio N° 1-a-b  
#include <iostream>  
#include <cmath>  
using namespace std;  
class cilindro {//Interface, declaracion de atributos y metodos  
    private:  
        float radio, altura, volumen;  
    public:  
        void asignar_datos(float r1, float a1);  
        void calcular_volumen();  
        float ver_radio(){return radio;} //Al escribir el codigo aqui,  
        float ver_altura(){return altura;} //rompemos el encapsulamiento  
        float ver_volumen(){return volumen;}  
};  
//Implementacion: desarrollo de los metodos.-  
void cilindro::asignar_datos(float r1, float a1){
```

```
radio= r1;
altura= a1;
}
void cilindro::calcular_volumen(){//Superficie de la base por la altura.-
    volumen = M_PI*radio*radio*altura;//M_PI constante con el valor 3.14.....
}

int main(int argc, char *argv[]) {
    cilindro C1,C2;
    C1.asignar_datos(12,21);
    C2.asignar_datos(5.3,10.2);
    C1.calcular_volumen();
    C2.calcular_volumen();
    cout<<"C1= "<<C1.ver_volumen()<<endl;
    cout<<"C2= "<<C2.ver_volumen()<<endl;
    return 0;
}
```

Item c y d

```
//© 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 N° 2 - Ejercicio N° 1-c-d
#include <iostream>
using namespace std;
class cilindro {
private:
    float radio, altura, volumen;
public:
    cilindro(float r1=0, float a1=0);//para que no quede inline lo desarrollamos abajo
    void asignar_datos(float r1, float a1);
    void calcular_volumen();
    float ver_radio(){return radio;};
    float ver_altura(){return altura;};
    float ver_volumen(){return volumen;};
};
void cilindro::asignar_datos(float r1, float a1){
    radio= r1;
    altura= a1;
}
void cilindro::calcular_volumen(){
    volumen = 3.14*radio*radio;
}//Constructor, observe como se deben inicializar los atributos.-
cilindro::cilindro(float r1, float a1):radio(r1),altura(a1){}
```

```
int main(int argc, char *argv[]) {  
    cilindro C1,C2(5.3,10.2);  
    C1.asignar_datos(12,21);  
    //C2.asignar_datos(5.3,10.2);  
    C1.calcular_volumen();  
    C2.calcular_volumen();  
    cout<<"C1= "<<C1.ver_volumen()<<endl;  
    cout<<"C2= "<<C2.ver_volumen()<<endl;  
    return 0;  
}//Prof. G. Sas.
```

Ejercicio 2.2

Cree una clase Rectángulo que posea:

- a) Un constructor que permita crear el objeto a partir de la base y altura del rectángulo.
- b) Otro constructor que permita crear el objeto a partir de las coordenadas (x, y) de sus 4 vértices.
- c) Métodos Area() y Perimetro() que calculen respectivamente el área y perímetro del rectángulo.
- d) Un método EsCuadrado() que permita saber si el rectángulo es cuadrado.
- e) Cree un programa cliente donde utilice varias instancias de la clase Rectángulo. Cree los objetos dinámicamente y no olvide borrarlos al terminar de utilizarlos.

Clase Rectangulo
Atributos
base, altura.
Métodos
rectangulo(a, b);
rectangulo(x1, x2, y1, y2);
Area();
Perimetro();
bool EsCuadrado();

```
//© 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 N° 2 - Ejercicio N° 2
#include <iostream>
using namespace std;
class rectangulo{
private:
    float base, altura;
public:
    rectangulo(float b, float a);
    rectangulo(float xx1,float xx2,float yy1,float yy2);
    float Area();
    float Perimetro();
    bool EsCuadrado();
};
rectangulo::rectangulo(float b, float a){
base= b; altura= a;
}
rectangulo::rectangulo(float xx1,float xx2,float yy1,float yy2){
    base= xx2 - xx1;
    altura = yy2 - yy1;
}
float rectangulo::Area(){
    return base*altura;
}
float rectangulo::Perimetro(){
    return 2*base+2*altura;
}
bool rectangulo::EsCuadrado(){
    if (base==altura)
        return true;
    else
        return false;
}
int main(int argc, char *argv[]) {
    rectangulo *R1= new rectangulo(3,4);
    rectangulo *R2= new rectangulo(1,2,3,4);
    cout<<"R1 Area: "<<R1->Area()<<endl;
    cout<<"R1 Perim:"<<R1->Perimetro()<<endl;
    if (R1->EsCuadrado())
        cout<<"Cuadrado"<<endl;
    else
        cout<<"Rectangulo"<<endl;
```

```
cout<<"R2 Area: "<<R2->Area()<<endl;
cout<<"R2 Perim:"<<R2->Perimetro()<<endl;
if (R2->EsCuadrado())
    cout<<"Cuadrado"<<endl;
else
    cout<<"Rectangulo"<<endl;
delete R1,R2;
return 0;
}//Prof. G. Sas.-
```

Ejercicio 2.3

Proponga una clase EcuaciónCuadratica para modelar ecuaciones cuadráticas de la forma $ax^2 + bx + c = 0$. La clase debe incluir:

- a) Un constructor que reciba los valores de los coeficientes a, b y c.
- b) Una función bool ObtenerRaices(float &x1, float &x2) que devuelva verdadero o falso según las raíces de la ecuación sean reales o no y además devuelva por referencia en x1 y x2 los valores de las raíces reales o los coeficientes de las raíces complejas.
- c) Una función con el prototipo bool TieneRaicesReales(void) que devolverá verdadero o falso según las raíces de la ecuación sean reales o no. Dicha función no estará disponible para ser utilizada directamente por el usuario (es decir, será privada) y será utilizada por la función ObtenerRaices() para saber el tipo de raíces.
- d) Cree un programa cliente que utilice objetos de la clase EcuaciónCuadratica e intente llamar a la función TieneRaicesReales() a partir de uno de los objetos creados. ¿Qué es lo que sucede?

CLASS ECUACIONCUADRATICA

Datos

a, b ,c.

TieneRaicesReales()

Funciones

EcuacionCuadratica(a, b, c);

ObtenerRaices(&x1, &x2);

```
//© 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 N° 2 - Ejercicio N° 3
#include <iostream>
#include <cmath>
using namespace std;
class EcuacionCuadratica{
private:
    float a, b ,c;
    bool TieneRaicesReales(); //Metodo Privado
public:
    EcuacionCuadratica(float a1, float b1, float c1);
    bool ObtenerRaices(float &x1, float &x2);
};
EcuacionCuadratica::EcuacionCuadratica(float a1, float b1, float c1){
    a=a1; b= b1; c= c1; //Constructor que inicializa los atributos
}
bool EcuacionCuadratica::TieneRaicesReales(){ //Metodo Privado
    if ((pow(b,2)-4*a*c)>=0)
        return true;
    else
        return false;
}
bool EcuacionCuadratica::ObtenerRaices(float &x1, float &x2){
    if (TieneRaicesReales()){ //Llamada al metodo privado
        x1=(-b+sqrt(pow(b,2)-4*a*c))/(2*a));
        x2=(-b-sqrt(pow(b,2)-4*a*c))/(2*a));
        return true;}
    else //complejo
        {x1= -b/(2*a);
         x2= sqrt(abs(pow(b,2)-4*a*c))/(2*a);
         return false;};
}
void mostrar(EcuacionCuadratica EC){
    float R1,R2;
    if (EC.ObtenerRaices(R1,R2)){
        cout<<"Reales"<<endl;
        cout<<"X1= "<<R1<<endl;
        cout<<"X2= "<<R2<<endl;}
```

```

else{
    cout<<"Complejas"<<endl;
    cout<<R1<<" + "<<abs(R2)<<" i "<<endl;
    cout<<R1<<" - "<<abs(R2)<<" i "<<endl;
}
// cuerpo del programa.-

int main(int argc, char *argv[]) {
    EcuacionCuadratica E1(1,8,1), E2(6,1,8);
    mostrar(E1);
    mostrar(E2);
    return 0;
}//Prof. G. Sas.

```

Ejercicio 2.4

Escriba una clase llamada Fecha con atributos para definir una fecha (día, mes, año). Implemente los siguientes métodos:

- Un constructor que reciba el día, mes y año y los inicialice.
- Métodos para devolver el día, mes y año de la fecha.
- Una función DiferenciaEnAnios() que reciba otro objeto de tipo fecha y devuelva la diferencia en años entre ambas.
- (OPCIONAL) Investigue como crear un constructor que no reciba parámetros e inicialize la fecha con los datos del reloj de la CPU.
- (OPCIONAL) Cree una función llamada SignoZodiacal() que devuelva una cadena de texto con el nombre del signo del zodiaco al que pertenece la fecha.

Implemente un programa cliente que utilice dicha clase.

CLASE FECHA	
Datos:	Día, mes, año.
Funciones:	Fecha(dia, mes, anio) Fecha() Ver_fecha(&dia, &mes, &anio) DiferenciaEnAnios() SignoZodiacal()

```

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

class fecha{
    private:

```

```
int dia, mes, anio;
public:
    //Item a)
    fecha( int d,int m, int a);
    //Item d)//Opcional
    fecha();
    //Item b)
    void ver_fecha(int &d,int &m,int &a);
    //Item c)
    int diferenciaEnAnios(fecha f);
    //Item d) //Opcional
    char *signoZodiacal(int, int);
};

fecha::fecha(int d, int m, int a){
    dia= d;
    mes= m;
    anio= a;
}
fecha::fecha(){//opcional
    time_t actual = time(NULL);
    struct tm fechaHoy = *(localtime(&actual));
    dia= fechaHoy.tm_mday;
    mes= fechaHoy.tm_mon;
    anio=fechaHoy.tm_year;
}
void fecha::ver_fecha(int &d,int &m,int &a){
d= dia;
m= mes;
a= anio;
}
int fecha::diferenciaEnAnios(fecha f){
    int f1= anio*10000+mes*100+dia;
    int d2,m2,a2;
    f.ver_fecha(d2,m2,a2);
    int f2= a2*10000+m2*100+d2;
    int dif= (f2 - f1)/10000;
    return dif;
}

char * fecha::signoZodiacal(int m, int d){
    switch (m){
        case 1:{ if (d<=20)
                    return "CAPRICORNIO.\n\n";
            else
                return "ACUARIO.\n\n";
            break;
    }
}
```

```
    }
    case 2:{if (d<=19)
        return "ACUARIO.\n\n";
    else
        return "PISCIS.\n\n";
    break;
}
case 3:{if(d<=20)
    return "PISCIS.\n\n";
else
    return "ARIES.\n\n";
break;
}
case 4:{if (d<=20)
    return "ARIES.\n\n";
else
    return "TAURO.\n\n";
break;
}
case 5:{if (d<=21)
    return "TAURO.\n\n";
else
    return "GEMINIS.\n\n";
break;
}
case 6:{if(d<=21)
    return "GEMINIS.\n\n";
else
    return "CANCER.\n\n";
break;
}
case 7:{if(d<=23)
    return "CANCER.\n\n";
else
    return "LEO.\n\n";
break;
}
case 8:{if(d<=23)
    return "LEO.\n\n";
else
    return "VIRGO.\n\n";
break;
}
case 9:{if (d<=23)
    return "LEO.\n\n";
else
    return "LIBRA.\n\n";
```

```

        break;
    }
    case 10:{if(d<=23)
        return "LIBRA.\n\n";
    else
        return "ESCORPION.\n\n";
    break;
}
case 11:{if(d<=22)
    return "ESCORPIO.\n\n";
else
    return "SAGITARIO.\n\n";
break;
}
case 12:{if(d<=21)
    return "SAGITARIO.\n\n";
else
    return "CAPRICORNIO.\n\n";
break;
}
}//del switch
}//de la funcion

int main(int argc, char *argv[]) {
    fecha fe1(1,1,2000), fe2(30,1,2011),fe3;
    int d0,d1,d2,m0,m1,m2,a0,a1,a2;
    fe1.ver_fecha(d1,m1,a1);
    fe2.ver_fecha(d2,m2,a2);
    fe3.ver_fecha(d0,m0,a0);
    cout<<"Fecha de hoy : "<<d0<<"/"<<m0+1<<"/"<<a0+1900<<endl;
    cout<<"Fecha 1: "<<d1<<"/"<<m1<<"/"<<a1<<endl;
    cout<<"Fecha 2: "<<d2<<"/"<<m2<<"/"<<a2<<endl;
    cout<<"Diferencia en anios: "<<fe1.diferenciaEnAnios(fe2)<<endl<<endl;
    cout<<"Fecha 1 - Signo Zodiacial: "<<fe1.signoZodiacial(m1, d1)<<endl;
    cout<<"Fecha 2 - Signo Zodiacial: "<<fe2.signoZodiacial(m2, d2)<<endl;
    cout<<"Fecha Hoy - Signo Zodiacial: "<<fe3.signoZodiacial(m0, d0)<<endl;
    return 0;
}

```

Ejercicio 2.5

Implemente una clase VectorDinamico que posea como atributo un puntero a entero que apunte a la memoria donde se almacenan los datos. Dicha clase debe poseer:

- a) Un constructor que reciba el tamaño inicial del vector, reserve la memoria necesaria e inicialize los valores del vector de manera aleatoria.

- b) Un destructor que se encargue de liberar la memoria reservada.
- c) Un método Duplicar() que duplique la cantidad de memoria reservada manteniendo los datos que ya estaban en el vector e inicializando al azar los nuevos valores.
- d) Un método VerValor() que reciba el número de elemento y devuelva su valor.

Cree un programa cliente que muestre la utilización de todas las funciones implementadas.

Clase vectorDinamico
Atributos
*p, tam
Métodos
vectorDinamico(t)
~vectorDinamico()
duplicar();
ver_valor(x)
ver_tam()

```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
```

```
//Autor: Prof. Gerardo Sas.
```

```
//sitio web: http://e-fich.unl.edu.ar/moodle/
```

```
//U1 Ejercicio N° 5
```

```
#include <iostream>
```

```
#include <cstdlib>
```

```
using namespace std;
```

```
class vectorDinamico{
```

```
private:
```

```
    int *p, tam;
```

```
public:
```

```
    vectorDinamico(int t); //a
```

```
    ~vectorDinamico(); // b
```

```
    void duplicar(); //c
```

```
    int ver_valor(int x){return *(p+x);}; //d
```

```
    int ver_tam(){return tam;};
```

```
};
```

```
vectorDinamico::vectorDinamico(int lar){
```

```
    tam= lar;
```

```
    p= new int [tam];
```

```
    for(int x=0; x < tam; x++)
```

```
        *(p+x)= rand()% 100;
```

```
}
```

```
void vectorDinamico::duplicar(){
```

```
    tam *= 2;
```

```
int *const aux= new int [tam];
for(int x=0; x < (tam/2); x++)
    *(aux+x)= *(p+x);
for(int x=(tam/2); x < tam; x++)
    *(aux+x)= rand()% 100;
delete []p;
p= aux;
}
vectorDinamico::~vectorDinamico(){
    delete []p;
}

int main(int argc, char *argv[]) {
vectorDinamico v(10);
for(int k= 0; k < v.ver_tam(); k++)
    cout<<v.ver_valor(k)<<", ";
cout<<"\nDuplicado: "<<endl;
v.duplicar();
for(int k= 0; k < v.ver_tam(); k++)
    cout<<v.ver_valor(k)<<", ";
return 0;
}
```

Ejercicio 2.6

Escriba una clase POOString que permita modelar una cadena de texto. La clase debe manejar dinámicamente la memoria para guardar la cadena de texto y para ello poseerá como atributo un puntero de tipo char que apuntará a la memoria donde se almacena la cadena. La clase debe tener:

- Un constructor que reciba como parámetro una cadena para inicializar el objeto. El constructor deberá reservar la cantidad de memoria necesaria para la cadena (tener en cuenta el carácter '\0' a la hora de reservar memoria) y guardar una copia de la misma.
- Un destructor que libere la memoria reservada por el constructor.
- Una función `char *GetString(void)` que devuelva un puntero a la cadena para poder mostrarla.
- (OPCIONAL) Una función `void Copy(char *c)` que copie la cadena c en la memoria del objeto. Para esto deberá descartar la memoria reservada anteriormente y reservar una nueva porción de memoria con el tamaño adecuado para la nueva cadena.
- (OPCIONAL) Una función `void Cat(char *c)` que concatene la cadena c a la cadena existente en la memoria del objeto. Para esto deberá descartar la memoria reservada anteriormente y reservar una nueva porción de memoria con el tamaño adecuado para contener la concatenación de ambas cadenas.
- Utilize la clase POOString desde un programa cliente.
- Intente compilar el siguiente código y luego responda:

```
...
int main(int argc, char *argv[]) {
    POOString a("Esta es una POOString");
```

```
POOString b(a);

cout<<a.GetString()<<endl;
cout<<b.GetString()<<endl;
return 0;
}

...
```

- ¿A qué se denomina un constructor de copia?

Un constructor de este tipo crea un objeto a partir de otro objeto existente. Estos constructores sólo tienen un argumento, que es una referencia a un objeto de su misma clase.

- ¿Por qué la compilación es correcta a pesar de no haber definido ningún constructor de ese tipo?

Porque en C++, si no se especifica ningún constructor copia, el compilador crea uno por defecto.

¿El constructor utilizado para el objeto b hace lo que realmente se desea?
SI

```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.
```

```
//Autor: Prof. Gerardo Sas.
```

```
//sitio web: http://e-fich.unl.edu.ar/moodle/
```

```
//U1 Ejercicio N° 6
```

```
#include <iostream>
```

```
#include <cstring>
```

```
#include <cstdio>
```

```
using namespace std;
```

```
class POOString{
```

```
private:
```

```
    char *cadena;
```

```
    int largo;
```

```
public:
```

```
    POOString(char *texto);// a)
```

```
    ~POOString();// b)
```

```
    char *GetString(void){return cadena;}// c)
```

```
    void copy(char *c);// d)
```

```
    void Cat(char *c);// e)
```

```
};
```

```
POOString::POOString(char *texto){
```

```
    largo= strlen(texto);
```

```
    cadena= new char [largo+1];
```

```
    for(int k=0; k < largo; k++)
```

```
        *(cadena+k) = *(texto+k);
```

```
    *(cadena+largo)="\0";
```

```
}
```

```
POOString::~POOString(){
    delete []cadena;
}
void POOString::copy(char *c){
    largo= strlen(c);
    delete []cadena;
    cadena= new char [largo+1];
    for(int k=0; k < largo; k++)
        *(cadena+k) = *(c+k);
    *(cadena+largo)='\\0';
}
void POOString::Cat(char *c){
    largo+= strlen(c);
    char *aux= new char [largo+1];
    for(int k=0; k < (largo-strlen(c)); k++)
        *(aux+k) = *(cadena+k);
    for(int k=(largo-strlen(c)),i= 0; k < (largo); k++, i++)
        *(aux+k) = *(c+i);
    *(aux+largo)='\\0';
    delete []cadena;
    cadena= aux;
}

int main(int argc, char *argv[]) {
    char palabras[50];
    cout<<"Ingrese una frase: ";
    gets(palabras);
    POOString S(palabras);
    cout<<endl<<"Se creo el objeto: "<<S.GetString()<<endl;
    cout<<"Ingrese otra frase para copiar: ";
    gets(palabras);
    S.copy(palabras);
    cout<<endl<<"Se copio en el objeto: "<<S.GetString()<<endl;
    cout<<"Ingrese una frase para concatenar: ";
    gets(palabras);
    S.Cat(palabras);
    cout<<endl<<"Se contateno en el objeto: "<<S.GetString()<<endl;
    POOString a("Esta es una POOString - constructor de copia");
    POOString b(a);
    cout<<"Direccion a: "<<&a<<endl;
    cout<<"Largo de a: "<<strlen(a.GetString())<<" cadena: "<<a.GetString()<<endl;
    cout<<"Direccion b: "<<&b<<endl;
    cout<<"Largo de b: "<<strlen(b.GetString())<<" cadena: "<<b.GetString()<<endl;
    return 0;
}
```

Ejercicio 2.7

Se dispone del siguiente tipo de dato:

```
struct Alumno {  
    char nombre[30];  
    float nota;  
};
```

En base al mismo se desea crear una clase Curso para modelar el cursado de diversas materias. La clase deberá contener el nombre de la materia y la cantidad de alumnos en el curso (dicha cantidad nunca superara los 40 alumnos) junto con una lista de los mismos. Proponga los siguientes métodos:

- Constructores y destructores según lo considere conveniente.
- Un método que permita agregar un alumno especificando su nombre y su calificación.
- Un método que determine el promedio del curso.
- Un método que devuelva la calificación mas alta y el nombre del alumno que la obtuvo.

```
//© Programación Orientada a Objetos- F.I.C.H. - U.N.L.  
//Autor: Prof. Gerardo Sas.  
//sitio web: http://e-fich.unl.edu.ar/moodle/  
//U1 Ejercicio N° 7  
#include <iostream>  
#include <cstring>  
using namespace std;  
typedef struct {  
    char nombre[30];  
    float nota;  
} Alumno;  
  
class Curso{  
private:  
    char nombre[30];//nombre del Curso  
    int cantidad;//cantidad de alumnos cargados  
    Alumno *clase;//Arreglo de struct de Alumnos  
public:  
    Curso(char *nom);//Crea una instancia de 40 Alumnos vacia  
    ~Curso();//Elimina la estructura creada  
    void cargar(Alumno al);//Añade el struct al arreglo y cuenta 1 en cantidad  
    Alumno ver_arreglo(int pos);//Devuelve el alumno en la posicion  
    int ver_cantidad(){return cantidad;};//Devuelve cantidad cargada  
    float promedio();//Devuelve el promedio del Curso  
    Alumno max();//Devuelve el alumno con mas nota  
};
```

```
Curso::Curso(char *nom){  
    strcpy(nombre, nom);//Copia el nombre del Curso  
    cantidad=0;//Inicializo la cantidad de alumnos  
    clase= new Alumno[40];  
    for(int l=0; l<40; l++)  
        (clase+l)->nota= 0;  
}  
Curso::~Curso(){//Elimina la estructura creada  
    delete []clase;  
}  
void Curso::cargar(Alumno al){//Añade el struct al arreglo y cuenta 1 en cantidad  
    *(clase+cantidad)=al;  
    cantidad++;  
}  
Alumno Curso::ver_arreglo(int pos){//Devuelve el alumno en la posicion  
    return *(clase+pos);  
}  
float Curso::promedio(){//Devuelve el promedio del Curso  
    int x;  
    float prom=0;  
    for(x=0; x < cantidad; x++){  
        prom+= (clase+x)->nota;  
    }  
    prom/=cantidad;  
    return prom;  
}  
Alumno Curso::max(){//Retorna el struct con la mayor nota  
    Alumno *max= clase;  
    int x;  
    for(x=0; x < cantidad; x++){  
        if(((clase+x)->nota)>(max->nota))  
            max= (clase+x);  
    }  
    return *max;  
}  
void mostrar(Curso D){  
    Alumno aux1;  
    for(int x=0; x<D.ver_cantidad(); x++){  
        aux1= D.ver_arreglo(x);  
        cout<<aux1.nombre<<" - "<<aux1.nota<<endl;  
    }  
}  
int main(int argc, char *argv[]){  
    Curso C("UNL-FICH-POO");  
    Alumno aux;  
    strcpy(aux.nombre,"Jose");  
    aux.nota= 9;
```

```
C.cargar(aux);
strcpy(aux.nombre,"Carlos");
aux.nota= 6;
C.cargar(aux);
strcpy(aux.nombre,"Marcelo");
aux.nota= 10;
C.cargar(aux);
mostrar(C);
cout<<"Promedio del curso: "<<C.promedio()<<endl;
aux= C.max();
cout<<"Mejor Promedio: "<<aux.nombre<<" - "<<aux.nota<<endl;
return 0;
}
```

Ejercicio 2.8 (OPCIONAL)

Consulte la bibliografía y averigüe el significado de la palabra reservada static y su utilización en las clases. Proponga una clase CuentaObjetos que permita llevar cuenta de los objetos que son creados. Para ello defina un constructor y un destructor que notifiquen por pantalla que el objeto fue creado/destruido y además muestre la cantidad de objetos que aun se encuentran en memoria. Utilice un programa cliente para probar la clase y realiza la creación/destrucción de los objetos de forma dinámica.

```
//© 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 N° 2 - Ejercicio N° 6
```

```
#include <iostream>
```

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

```
using namespace std;
```

```
// miembros estáticos en clases
```

```
class CuentaObjetos {
```

```
public:
```

```
    static int n;//Es publica y estatica
```

```
    CuentaObjetos () { cout<<n<<" suma 1\n";n++; };
```

```
    ~CuentaObjetos () { cout<<n<<" resta 1\n";n--; };
```

```
};
```

/*Observa que es necesario declarar e inicializar los miembros static de la clase, esto es por dos motivos. El primero es que los miembros static deben existir aunque no exista ningún objeto de la clase, declarar la clase no crea

los datos miembro estáticos, es necesario hacerlo explícitamente. El segundo es

porque no lo hiciéramos, al declarar objetos de esa clase los valores de los

miembros estáticos estarían indefinidos, y los resultados no serían los esperados.

```
/*
int CuentaObjetos::n=0;//Así se inicializa el miembro static n de la clase CuentaObjetos

int main () {
    CuentaObjetos (*b)= new CuentaObjetos [5];
    cout <<"Cantidad actual de elementos: "<< CuentaObjetos::n << endl;
    delete []b;
    cout <<"Cantidad actual de elementos: "<< CuentaObjetos::n << endl;
}
```

Ejercicio 2.9 (OPCIONAL)

Consulte la bibliografía y averigüe el significado de la palabra reservada static y su utilización en las clases. Proponga una clase llamada MathUtils que posea una función que calcule la distancia entre 2 puntos recibiendo sus coordenadas. Cree un programa cliente que llame a dicha función sin crear ningún objeto.

```
//© 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 N° 2 - Ejercicio N° 9

#include <iostream>
#include <cmath>
using namespace std;
class MathUtils{
private:
    static float distancia;
public:
    static float mostrar(){return distancia;};
    static void calcular(float x1,float x2,float y1,float y2)
        {distancia= sqrt(pow(x2-x1,2)+pow(y2-y1,2));};
};

float MathUtils::distancia= 0;//Inicializo el miembro static de la clase.-

int main(int argc, char *argv[]) {
    //MathUtils X;
    MathUtils::calcular(8,6,4,2);
    cout<<"Distancia "<<MathUtils::mostrar()<<endl;
    return 0;
}

/*Observa que es necesario declarar e inicializar los miembros static de la clase,
esto es por dos motivos. El primero es que los miembros static deben existir aunque
no exista ningún objeto de la clase, declarar la clase no crea los datos miembro
estáticos, es necesario hacerlo explícitamente. El segundo es porque no lo hiciéramos,
al declarar objetos de esa clase los valores de los miembros estáticos estarían
indefinidos, y los resultados no serían los esperados.

En el caso de las funciones miembro static la utilidad es menos evidente. Estas
funciones no pueden acceder a los miembros de los objetos, sólo pueden acceder a
```

los datos miembro de la clase que sean static. Esto significa que no tienen acceso al puntero this, y además suelen ser usadas con su nombre completo, incluyendo el nombre de la clase y el operador de ámbito (::).

*/

Ejercicio 2.10 (OPCIONAL)

a) Intente responder a la siguiente pregunta antes de continuar leyendo este enunciado: ¿Tiene sentido colocar un constructor en la parte privada de una clase?

NO

b) Analice el siguiente código y determine qué particularidad tendrá esta clase:

```
class Singleton {  
    Singleton() { /*unico constructor*/ };  
    static Singleton *instancia;  
public:  
    static Singleton *ObtenerInstancia() {  
        if(!instancia) instancia=new Singleton();  
        return instancia;  
    };  
    // ...interfaz de la clase...  
};  
Singleton *Singleton::instancia=NULL;
```

Cuestionario

1. ¿Qué es un objeto?

Es una instancia de una clase.

¿Qué es una clase?

Una clase es un tipo genérico (una abstracción) que representa a un conjunto de objetos de similares características.

¿Qué es una instancia?

Luego de haber definido una clase, es posible obtener tantos objetos (instancias) de dicha clase como sea necesario.

2. ¿Qué es un constructor?

Es un método del objeto.

3. ¿Cuándo se invoca al constructor de un objeto? ¿Cuándo al destructor?

Este método constructor se ejecutara automáticamente cuando se instancia la clase.

Y el destructor al liberar la memoria que ocupa el objeto de esa clase.

4. ¿Qué significa constructor por defecto?

Toda clase tiene predefinido el método constructor / destructor por defecto.

¿Y constructor de copia?

Un constructor de este tipo crea un objeto a partir de otro objeto existente. Estos constructores sólo tienen un argumento, que es una referencia a un objeto de su misma clase.

5. ¿Qué significa encapsulamiento?

Cuando declaramos una clase cuyos atributos no pueden ser accedidos desde el exterior decimos que la clase encapsula estos atributos. El concepto de encapsulamiento es fundamental en programación orientada a objetos y tiene por finalidad facilitar la reusabilidad y depuración del código.

Cuando utilizamos una clase, sabemos que esta tiene determinadas responsabilidades y nos interesa que las cumpla sin importar cómo lo hace

6. ¿Cómo se consigue el ocultamiento de datos en C++?

Para esto se pueden declarar partes privadas que no pueden ser accedidas excepto por métodos propias de la clase. (a no ser que declare amistad con alguien)

7. ¿Para qué sirven las etiquetas private y public?

Etiquetas de permisos: pueden ser cualquiera de las 3 palabras reservadas: **private:** , **public:** o **protected:**. Estas hacen referencias a los permisos que los miembros podrán tener:

- **private:** las funciones miembro de la clase son accesibles sólo por otros miembros de la misma clase o de sus clases amigas friend
- **protected:** los miembros son accesibles, además de por los miembros de la misma clase y de sus clases friend, por los miembros de las clases derivadas.
- **public:** son accesibles por cualquiera para el cual la clase es visible.

Si se declaran los miembros de una clase antes de incluir cualquier etiqueta, se consideran privados, por lo que se indica que los permisos por defecto para los miembros es *private*.

8. ¿Cuál es la diferencia, en C++, entre una clase y una estructura?

Las clases son una manera lógica de organizar datos y funciones en una estructura única. Se declaran utilizando la palabra clave class, cuya funcionalidad es similar a la de la palabra reservada de C++ struct, pero con la posibilidad de incluir funciones como miembros, en vez de solamente datos. En la sentencia struct de C los miembros por defecto son públicos (**public**) en vez de privados (**private**) como en class.

9. ¿Para qué sirve el puntero this?

this representa dentro de una clase la dirección en memoria del objeto de esa clase que está siendo ejecutado. Es un puntero cuyo valor es siempre la dirección de memoria del objeto. Puede ser usado para chequear si un parámetro pasado a una función de un objeto es el objeto en sí.

10. ¿Es posible usar el mismo nombre de función para una función miembro de una clase y una función externa? Si es posible, indique como distinguirlas.

En caso contrario, indique las razones.

Una función miembro siempre forma parte de una clase y para invocarla lo debe hacer el objeto a la que pertenece, en cambio una externa no.-

11. ¿Qué implica que un atributo se declare con el modificador static?¿Que pasa cuando un método es declarado con dicho modificador?

Ciertos miembros de una clase pueden ser declarados como static. Los miembros static tienen algunas propiedades especiales.

En el caso de los datos miembro static sólo existirá una copia que compartirán todos los objetos de la misma clase. Si consultamos el valor de ese dato desde cualquier objeto de esa

clase obtendremos siempre el mismo resultado, y si lo modificamos, lo modificaremos para todos los objetos. es necesario declarar e inicializar los miembros static de la clase, esto es por dos motivos. El primero es que los miembros static deben existir aunque no exista ningún objeto de la clase, declarar la clase no crea los datos miembro estáticos, es necesario hacerlo explícitamente. El segundo es porque no lo hiciéramos, al declarar objetos de esa clase los valores de los miembros estáticos estarían indefinidos, y los resultados no serían los esperados.

En el caso de las funciones miembro static la utilidad es menos evidente. Estas funciones no pueden acceder a los miembros de los objetos, sólo pueden acceder a los datos miembro de la clase que sean static. Esto significa que no tienen acceso al puntero this, y además suelen ser usadas con su nombre completo, incluyendo el nombre de la clase y el operador de ámbito (::).