```
void DiaAnyo::visualizar() const
{
    cout << "mes = " << mes << " , dia = " << dia << endl;
}
```

Por último, el archivo `DemoFecha.cpp` contiene la función `main()`, crea los objetos y se envian mensajes.

```
#include <iostream>
using namespace std;
#include "DiaAnyo.h"

int main()
{
    DiaAnyo* hoy;
    DiaAnyo* cumpleanyos;
    int d, m;

    cout << "Introduzca fecha de hoy, dia: ";
    cin >> d;
    cout << "Introduzca el número de mes: ";
    cin >> m;
    hoy = new DiaAnyo(d, m);
    cout << "Introduzca su fecha de nacimiento, dia: ";
    cin >> d;
    cout << "Introduzca el número de mes: ";
    cin >> m;
    cumpleanyos = new DiaAnyo(d, m);
    cout << " La fecha de hoy es ";
    hoy -> visualizar();
    cout << " Su fecha de nacimiento es ";
    cumpleanyos -> visualizar();
    if (hoy -> igual(*cumpleanyos))
        cout << "¡Feliz cumpleaños ! " << endl;
    else
        cout << "¡Feliz dia ! " << endl;
    return 0;
}
```


2.2.6. Sobrecarga de funciones miembro

Al igual que sucede con las funciones no miembro de una clase, las funciones miembro de una clase se pueden sobrecargar. Una función miembro se puede sobrecargar pero sólo en su propia clase.

Las mismas reglas utilizadas para sobrecargar funciones ordinarias se aplican a las funciones miembro: dos miembros sobrecargados no puede tener el mismo número y tipo de parámetros. La sobrecarga permite utilizar un mismo nombre para una función y ejecutar la función definida más adecuada a los parámetros pasados durante la ejecución del programa. La ventaja fundamental de trabajar con funciones miembro sobrecargadas es la comodidad que aporta a la programación.

Para ilustrar la sobrecarga, veamos la clase `Vector` donde aparecen diferentes funciones miembro con el mismo nombre y diferentes tipos de parámetros.

```
class Vector
{
public:
    int suma(int m[], int n);    //funcion 1
    int suma(const Vector& v);   //funcion 2
    float suma(float m, float n);      //funcion 3
    int suma();                  //funcion 4
};
```

Normas para la sobrecarga

No pueden existir dos funciones en el mismo ámbito con igual signatura (lista de parámetros).

2.3. CONSTRUCTORES Y DESTRUCTORES

Un *constructor* es una función miembro que se ejecuta automáticamente cuando se crea un objeto de una clase. Sirve para inicializar los miembros de la clase.

El constructor tiene el mismo nombre que la clase. Cuando se define no se puede especificar un valor de retorno, nunca devuelve un valor. Sin embargo, puede tomar cualquier número de argumentos.

Reglas

1. El constructor tiene el mismo nombre que la clase.
2. Puede tener cero, o más argumentos.
3. No tiene tipo de retorno.

EJEMPLO 2.4. La clase `Rectángulo` tiene un constructor con cuatro parámetros.

```
class Rectangulo
{
private:
    int izdo, superior;
    int dcha, inferior;
public:
    Rectangulo(int iz, int sr, int d, int inf) // constructor
    {
        izdo = iz; superior = sr;
        dcha = d; inferior = inf;
    }
    // definiciones de otros métodos miembro
};
```

Al crear un objeto se pasan los valores al constructor, con la misma sintaxis que la de llamada a una función. Por ejemplo:

```
Rectangulo Rect(4, 4, 10, 10);
Rectangulo* ptr = new Rectangulo(25, 25, 75, 75);
```

Se han creado dos instancias de `Rectangulo`, pasando valores concretos al constructor de la clase.

2.3.1. Constructor por defecto

Un constructor que no tiene parámetros, o que se puede llamar sin argumentos, se llama *constructor por defecto*. Un constructor por defecto normalmente inicializa los miembros dato de la clase con valores por defecto.

Regla

C++ crea automáticamente un constr uctor por defecto cuando no e xisten otros constructores. Tal constructor inicializa el objeto a ceros binar ios.

EJEMPLO 2.5. El constructor de la clase `Complejo` tiene dos argumentos con v alores por defecto `0` y `1` respectivamenete, por tanto, puede llamarse sin argumentos .

```
class Complejo
{
private                                :
   double x;
   double y;
public:
   Complejo(double r = 0.0, double i = 1.0)
   {
      x = r;
      y = i;
   }
};
```


Cuando se crea un objeto `Complejo` puede inicializarse a los valores por defecto, o bien a otros valores.

```
Complejo z1;    // z1.x == 0.0, z1.y == 1.0
Complejo* pz = new Complejo(-2, 3);  // pz -> x == -2, pz -> y == 3
```

2.3.2. Constructores sobrecargados

Al igual que se puede sobrecargar un método de una clase, también se puede sobrecargar el constructor de una clase. De hecho los constructores sobrecargados son bastante frecuentes, proporcionan diferentes alternativas de inicializar objetos.

> ***Regla***
>
> Para prevenir a los usuarios de la clase de crear un objeto sin parámetros, se puede: 1) omitir el constructor por defecto; o bien, 2) hacer el constructor privado.

EJEMPLO 2.6. La clase `EquipoSonido` se define con tres constructores; un constructor por defecto, otro con un argumento de tipo cadena y el tercero con tres argumentos.

```
class EquipoSonido
{
private:
    int potencia, voltios;
    string marca;
public:
    EquipoSonido()   // constructor por defecto
    {
        marca = "Sin marca"; potencia = voltios = 0;
    }
    EquipoSonido(string mt)
    {
        marca = mt; potencia = voltios = 0;
    }
    EquipoSonido(string mt, int p, int v)
    {
        marca = mt;
        potencia = p;
        voltios = v;
    }
};
```


La instanciación de un objeto `EquipoSonido` puede hacerse llamando a cualquier constructor. A continuación se crean tres objetos:

```
EquipoSonido rt;          // constructor por defecto
EquipoSonido ft("POLASIT");
EquipoSonido gt("PARTOLA", 35, 220);
```

2.3.3. Array de objetos

Los objetos se pueden estructurar como un array. Cuando se crea un array de objetos éstos se inicializan llamando al constructor sin argumentos. Por consiguiente, siempre que se prevea organizar los objetos en un array, la clase debe tener un constructor que pueda llamarse sin parámetros.

> ***Precaución***
>
> Tenga cuidado con la escritura de una clase con sólo un constructor con argumentos. Si se omite un constructor que pueda llamarse sin argumento no será posible crear un array de objetos.