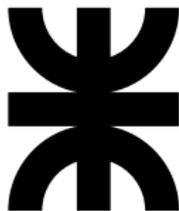


Informática II

Clases en C++

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2019 –

Declaración de clase en C++

```
class Hora {  
public:  
    // Constructor  
    Hora();  
    // Funciones miembros  
    void estableceHora(int, int, int);  
    void imprimeMilitar();  
    void imprimeEstandar();  
  
private:  
    // Miembros datos 'private'  
    int hora;  
    int minuto;  
    int segundo;  
}; // fin de la clase Hora
```

Declaración de clase en C++

```
class Hora {  
public:  
    // Constructor  
    Hora();  
    // Funciones miembros  
    void estableceHora(int, int, int);  
    void imprimeMilitar();  
    void imprimeEstandar();  
  
private:  
    // Miembros datos 'private'  
    int hora;  
    int minuto;  
    int segundo;  
}; // fin de la clase Hora
```

- ▶ Se declara con la palabra reservada `class` (bloque entre `{` y `}`), la declaración termina con punto y coma

Declaración de clase en C++

```
class Hora {  
    public:  
        // Constructor  
        Hora();  
        // Funciones miembros  
        void estableceHora(int, int, int);  
        void imprimeMilitar();  
        void imprimeEstandar();  
  
    private:  
        // Miembros datos 'private'  
        int hora;  
        int minuto;  
        int segundo;  
}; // fin de la clase Hora
```

- ▶ Se declara con la palabra reservada `class` (bloque entre `{` y `}`), la declaración termina con punto y coma
- ▶ Identificador `Hora` como *nuevo tipo*

Declaración de clase en C++

```
class Hora {
public:
    // Constructor
    Hora();
    // Funciones miembros
    void estableceHora(int, int, int);
    void imprimeMilitar();
    void imprimeEstandar();

private:
    // Miembros datos 'private'
    int hora;
    int minuto;
    int segundo;
}; // fin de la clase Hora

Hora atardecer, // objeto de tipo Hora
arregloDeHoras[5], // arreglo
*apuntadorAHora, // puntero
&horaCena = atardecer // referencia
```

- ▶ Se declara con la palabra reservada `class` (bloque entre `{` y `}`), la declaración termina con punto y coma
- ▶ Identificador `Hora` como *nuevo tipo*
- ▶ La declaración no reserva espacio en memoria, sino que crea un nuevo tipo de dato

Declaración de clase en C++

```
class Hora {
public:
    // Constructor
    Hora();
    // Funciones miembros
    void estableceHora(int, int, int);
    void imprimeMilitar();
    void imprimeEstandar();

private:
    // Miembros datos 'private'
    int hora;
    int minuto;
    int segundo;
}; // fin de la clase Hora

Hora atardecer, // objeto de tipo Hora
arregloDeHoras[5], // arreglo
*apuntadorAHora, // puntero
&horaCena = atardecer // referencia
```

- ▶ Se declara con la palabra reservada `class` (bloque entre `{` y `}`), la declaración termina con punto y coma
- ▶ Identificador `Hora` como *nuevo tipo*
- ▶ La declaración no reserva espacio en memoria, sino que crea un nuevo tipo de dato
- ▶ Especificadores de acceso a miembro, etiquetas `public` y `private`

Declaración de clase en C++

```
class Hora {
public:
    // Constructor
    Hora();
    // Funciones miembros
    void estableceHora(int, int, int);
    void imprimeMilitar();
    void imprimeEstandar();

private:
    // Miembros datos 'private'
    int hora;
    int minuto;
    int segundo;
}; // fin de la clase Hora

Hora atardecer, // objeto de tipo Hora
arregloDeHoras[5], // arreglo
*apuntadorAHora, // puntero
&horaCena = atardecer // referencia
```

- ▶ Se declara con la palabra reservada `class` (bloque entre `{` y `}`), la declaración termina con punto y coma
- ▶ Identificador `Hora` como *nuevo tipo*
- ▶ La declaración no reserva espacio en memoria, sino que crea un nuevo tipo de dato
- ▶ Especificadores de acceso a miembro, etiquetas `public` y `private`
- ▶ El modo de acceso predeterminado es `private` (`public` en `struct`)

Declaración de clase en C++

```
class Hora {
public:
    // Constructor
    Hora();
    // Funciones miembros
    void estableceHora(int, int, int);
    void imprimeMilitar();
    void imprimeEstandar();

private:
    // Miembros datos 'private'
    int hora;
    int minuto;
    int segundo;
}; // fin de la clase Hora

Hora atardecer, // objeto de tipo Hora
arregloDeHoras[5], // arreglo
*apuntadorAHora, // puntero
&horaCena = atardecer // referencia
```

- ▶ Se declara con la palabra reservada `class` (bloque entre `{` y `}`), la declaración termina con punto y coma
- ▶ Identificador `Hora` como *nuevo tipo*
- ▶ La declaración no reserva espacio en memoria, sino que crea un nuevo tipo de dato
- ▶ Especificadores de acceso a miembro, etiquetas `public` y `private`
- ▶ El modo de acceso predeterminado es `private` (`public` en `struct`)
- ▶ *Constructor*: función miembro de igual nombre que la clase

Declaración de clase en C++

```
class Hora {
public:
    // Constructor
    Hora();
    // Funciones miembros
    void estableceHora(int, int, int);
    void imprimeMilitar();
    void imprimeEstandar();

private:
    // Miembros datos 'private'
    int hora;
    int minuto;
    int segundo;
}; // fin de la clase Hora

Hora atardecer, // objeto de tipo Hora
arregloDeHoras[5], // arreglo
*apuntadorAHora, // puntero
&horaCena = atardecer // referencia
```

- ▶ Se declara con la palabra reservada `class` (bloque entre `{` y `}`), la declaración termina con punto y coma
- ▶ Identificador `Hora` como *nuevo tipo*
- ▶ La declaración no reserva espacio en memoria, sino que crea un nuevo tipo de dato
- ▶ Especificadores de acceso a miembro, etiquetas `public` y `private`
- ▶ El modo de acceso predeterminado es `private` (`public` en `struct`)
- ▶ *Constructor*: función miembro de igual nombre que la clase
- ▶ El constructor no regresa valor

Declaración de clase en C++

```
class Hora {  
public:  
    // Constructor  
    Hora();  
    // Funciones miembros  
    void estableceHora(int, int, int);  
    void imprimeMilitar();  
    void imprimeEstandar();  
  
private:  
    // Miembros datos 'private'  
    int hora;  
    int minuto;  
    int segundo;  
}; // fin de la clase Hora  
  
Hora atardecer, // objeto de tipo Hora  
    arregloDeHoras[5], // arreglo  
    *apuntadorAHora, // puntero  
    &horaCena = atardecer // referencia
```

- ▶ Se declara con la palabra reservada `class` (bloque entre `{` y `}`), la declaración termina con punto y coma
- ▶ Identificador `Hora` como *nuevo tipo*
- ▶ La declaración no reserva espacio en memoria, sino que crea un nuevo tipo de dato
- ▶ Especificadores de acceso a miembro, etiquetas `public` y `private`
- ▶ El modo de acceso predeterminado es `private` (`public` en `struct`)
- ▶ *Constructor*: función miembro de igual nombre que la clase
- ▶ El constructor no regresa valor
- ▶ Funciones miembros (`set` y `get`) son la *interfaz de la clase*

Declaración de clase en C++

Inicialización de datos miembros:

- ▶ No se puede inicializar en el lugar donde se declaran (en el cuerpo de la clase)

Declaración de clase en C++

Inicialización de datos miembros:

- ▶ No se puede inicializar en el lugar donde se declaran (en el cuerpo de la clase)
- ▶ Se inicializan mediante el constructor de la clase o pueden ser valores asignados mediante funciones `set` (establecer/fijar)

Declaración de clase en C++

Inicialización de datos miembros:

- ▶ No se puede inicializar en el lugar donde se declaran (en el cuerpo de la clase)
- ▶ Se inicializan mediante el constructor de la clase o pueden ser valores asignados mediante funciones `set` (establecer/fijar)

Alcance de una clase

- ▶ Dentro del alcance de una clase se puede acceder a los miembros de esa clase desde todas las funciones miembros de la misma

Declaración de clase en C++

Inicialización de datos miembros:

- ▶ No se puede inicializar en el lugar donde se declaran (en el cuerpo de la clase)
- ▶ Se inicializan mediante el constructor de la clase o pueden ser valores asignados mediante funciones `set` (establecer/fijar)

Alcance de una clase

- ▶ Dentro del alcance de una clase se puede acceder a los miembros de esa clase desde todas las funciones miembros de la misma
- ▶ Fuera del alcance de una clase, se hace referencia a los miembros de la clase a través de uno de los manipuladores de objeto: el nombre de un objeto, una referencia o un apuntador a un objeto

Declaración de clase en C++

Inicialización de datos miembros:

- ▶ No se puede inicializar en el lugar donde se declaran (en el cuerpo de la clase)
- ▶ Se inicializan mediante el constructor de la clase o pueden ser valores asignados mediante funciones `set` (establecer/fijar)

Alcance de una clase

- ▶ Dentro del alcance de una clase se puede acceder a los miembros de esa clase desde todas las funciones miembros de la misma
- ▶ Fuera del alcance de una clase, se hace referencia a los miembros de la clase a través de uno de los manipuladores de objeto: el nombre de un objeto, una referencia o un apuntador a un objeto

Los clientes/usuarios tienen acceso a la interfaz de la clase, pero no deben tener acceso a la implementación de la clase

Separación de interfaz e implementación

Archivo 'hora1.h' (D&D 4° ed.)

```
1 // Declaración de la clase Hora.
2 // Las funciones miembro están definidas en hora1.cpp
3
4 // Evita inclusiones múltiples del archivo encabezado
5 #ifndef HORA1_H
6 #define HORA1_H
7
8 // Declaración del tipo de dato abstracto Hora
9 class Hora {
10     public:
11         Hora(); // constructor
12         void estableceHora(int, int, int); // establece hora, minuto, segundo
13         void imprimeMilitar(); // imprime la hora en formato militar
14         void imprimeEstandar(); // imprime la hora en formato estándar
15
16     private:
17         int hora; // 0 - 23
18         int minuto; // 0 - 59
19         int segundo; // 0 - 59
20 }; // fin de la clase Hora
21
22 #endif
```

Separación de interfaz e implementación

Archivo 'hora1.cpp' (D&D 4° ed.)

```
1 // Definiciones de las funciones miembro de la clase Hora
2 #include <iostream>
3
4 using std::cout;
5 #include "hora1.h"
6
7 // El constructor Hora inicializa en cero a cada dato miembro.
8 Hora::Hora() {
9     hora = minuto = segundo = 0;
10 }
11
12 // Establece un nuevo valor de Hora por medio de la hora militar.
13 void Hora::estableceHora(int h, int m, int s) { // Implementación
14 }
15
16 // Imprime Hora en formato militar
17 void Hora::imprimeMilitar() { // Implementación
18 }
19
20 // Imprime Hora en formato estándar
21 void Hora::imprimeEstandar() { // Implementación
22 }
```

Separación de interfaz e implementación

- ▶ Se compila la clase
 > `g++ -c hora1.cpp`

Separación de interfaz e implementación

- ▶ Se compila la clase
> `g++ -c hora1.cpp`
- ▶ Se compila la aplicación
> `g++ -c fig16_04.cpp`

Separación de interfaz e implementación

- ▶ Se compila la clase
> `g++ -c hora1.cpp`
- ▶ Se compila la aplicación
> `g++ -c fig16_04.cpp`
- ▶ Se enlaza y se genera el binario
> `g++ hora1.o fig16_04.o`

Separación de interfaz e implementación

- ▶ Se compila la clase
> `g++ -c hora1.cpp`
- ▶ Se compila la aplicación
> `g++ -c fig16_04.cpp`
- ▶ Se enlaza y se genera el binario
> `g++ hora1.o fig16_04.o`

O bien, se compila todo junto

> `g++ hora1.cpp fig16_04.cpp`

Separación de interfaz e implementación

- ▶ Se compila la clase
> g++ -c hora1.cpp
- ▶ Se compila la aplicación
> g++ -c fig16_04.cpp
- ▶ Se enlaza y se genera el binario
> g++ hora1.o fig16_04.o

O bien, se compila todo junto

> g++ hora1.cpp fig16_04.cpp

Ejecución:

```
> ./a.out
La hora en militar inicial es 00:00:00
La hora estándar inicial es 12:00:00 AM
La hora militar después de estableceHora es 13:27:06
La hora estándar después de estableceHora es 1:27:06 PM
Después de intentar establecer valores inválidos:
Hora militar: 00:00:00
Hora estándar: 12:00:00 AM
```

Separación de interfaz e implementación

- ▶ Se compila la clase
> g++ -c hora1.cpp
- ▶ Se compila la aplicación
> g++ -c fig16_04.cpp
- ▶ Se enlaza y se genera el binario
> g++ hora1.o fig16_04.o

O bien, se compila todo junto

> g++ hora1.cpp fig16_04.cpp

Ejecución:

```
> ./a.out
La hora en militar inicial es 00:00:00
La hora estándar inicial es 12:00:00 AM
La hora militar después de estableceHora es 13:27:06
La hora estándar después de estableceHora es 1:27:06 PM
Después de intentar establecer valores inválidos:
Hora militar: 00:00:00
Hora estándar: 12:00:00 AM
```

¿Qué dificultades ve en la separación de la interfaz e implementación?

Separación de interfaz e implementación

- ▶ Se compila la clase
> `g++ -c hora1.cpp`
- ▶ Se compila la aplicación
> `g++ -c fig16_04.cpp`
- ▶ Se enlaza y se genera el binario
> `g++ hora1.o fig16_04.o`

O bien, se compila todo junto

> `g++ hora1.cpp fig16_04.cpp`

Ejecución:

```
> ./a.out
La hora en militar inicial es 00:00:00
La hora estándar inicial es 12:00:00 AM
La hora militar después de estableceHora es 13:27:06
La hora estándar después de estableceHora es 1:27:06 PM
Después de intentar establecer valores inválidos:
Hora militar: 00:00:00
Hora estándar: 12:00:00 AM
```

¿Qué dificultades ve en la separación de la interfaz e implementación?

- ▶ Funciones `inline` en la declaración de clase

Separación de interfaz e implementación

- ▶ Se compila la clase
> `g++ -c hora1.cpp`
- ▶ Se compila la aplicación
> `g++ -c fig16_04.cpp`
- ▶ Se enlaza y se genera el binario
> `g++ hora1.o fig16_04.o`

O bien, se compila todo junto

> `g++ hora1.cpp fig16_04.cpp`

Ejecución:

```
> ./a.out
La hora en militar inicial es 00:00:00
La hora estándar inicial es 12:00:00 AM
La hora militar después de estableceHora es 13:27:06
La hora estándar después de estableceHora es 1:27:06 PM
Después de intentar establecer valores inválidos:
Hora militar: 00:00:00
Hora estándar: 12:00:00 AM
```

¿Qué dificultades ve en la separación de la interfaz e implementación?

- ▶ Funciones `inline` en la declaración de clase
- ▶ El acceso al archivo de cabecera muestra los miembros `private`

Control de acceso a miembros y funciones miembros

Control de acceso a miembros

- ▶ Los especificadores de acceso a miembros `public` y `private` controlan el acceso a los datos y a las funciones miembros de una clase

Control de acceso a miembros y funciones miembros

Control de acceso a miembros

- ▶ Los especificadores de acceso a miembros `public` y `private` controlan el acceso a los datos y a las funciones miembros de una clase
- ▶ El modo de acceso predeterminado de una clase es `private`

Control de acceso a miembros y funciones miembros

Control de acceso a miembros

- ▶ Los especificadores de acceso a miembros **public** y **private** controlan el acceso a los datos y a las funciones miembros de una clase
- ▶ El modo de acceso predeterminado de una clase es **private**
- ▶ Solo se puede acceder a los miembros privados de una clase mediante funciones miembros

Control de acceso a miembros y funciones miembros

Control de acceso a miembros

- ▶ Los especificadores de acceso a miembros **public** y **private** controlan el acceso a los datos y a las funciones miembros de una clase
- ▶ El modo de acceso predeterminado de una clase es **private**
- ▶ Solo se puede acceder a los miembros privados de una clase mediante funciones miembros
- ▶ El propósito de los miembros públicos es brindar a los usuario de la clase los servicios (comportamientos/funciones) que proporciona la clase

Control de acceso a miembros y funciones miembros

Control de acceso a miembros

- ▶ Los especificadores de acceso a miembros **public** y **private** controlan el acceso a los datos y a las funciones miembros de una clase
- ▶ El modo de acceso predeterminado de una clase es **private**
- ▶ Solo se puede acceder a los miembros privados de una clase mediante funciones miembros
- ▶ El propósito de los miembros públicos es brindar a los usuario de la clase los servicios (comportamientos/funciones) que proporciona la clase
- ▶ Este conjunto de servicios forma la interfaz pública de la clase

Control de acceso a miembros y funciones miembros

Control de acceso a miembros

- ▶ Los especificadores de acceso a miembros **public** y **private** controlan el acceso a los datos y a las funciones miembros de una clase
- ▶ El modo de acceso predeterminado de una clase es **private**
- ▶ Solo se puede acceder a los miembros privados de una clase mediante funciones miembros
- ▶ El propósito de los miembros públicos es brindar a los usuario de la clase los servicios (comportamientos/funciones) que proporciona la clase
- ▶ Este conjunto de servicios forma la interfaz pública de la clase

Funciones de acceso y funciones de utilidad (o de utilidad)

- ▶ No todas las funciones miembros necesitan ser públicas

Control de acceso a miembros y funciones miembros

Control de acceso a miembros

- ▶ Los especificadores de acceso a miembros **public** y **private** controlan el acceso a los datos y a las funciones miembros de una clase
- ▶ El modo de acceso predeterminado de una clase es **private**
- ▶ Solo se puede acceder a los miembros privados de una clase mediante funciones miembros
- ▶ El propósito de los miembros públicos es brindar a los usuario de la clase los servicios (comportamientos/funciones) que proporciona la clase
- ▶ Este conjunto de servicios forma la interfaz pública de la clase

Funciones de acceso y funciones de utilidad (o de utilidad)

- ▶ No todas las funciones miembros necesitan ser públicas
- ▶ Algunas funciones miembro permanecen como privadas y sirven como *funciones de utilidad* para otras funciones de la clase

Funciones miembros `set` y `get`

Las clases generalmente brindan funciones miembros `public` para que los usuarios de la clase puedan *establecer* (o sea, escribir) y *obtener* (o sea, leer) los datos miembros `private` de la clase.

Funciones miembros `set` y `get`

Las clases generalmente brindan funciones miembros `public` para que los usuarios de la clase puedan *establecer* (o sea, escribir) y *obtener* (o sea, leer) los datos miembros `private` de la clase.

```
class Hora {
public:
    Hora(int = 0, int = 0, int = 0); // constructor

    // funciones establecer
    void estableceHora(int, int, int); // establece hora, minuto, segundo
    void estableceHora(int); // establece hora
    void estableceMinuto(int); // establece minuto
    void estableceSegundo(int); // establece segundo

    // funciones obtener
    int obtieneHora(); // devuelve hora
    int obtieneMinuto(); // devuelve minuto
    int obtieneSegundo(); // devuelve segundo

private:
    int hora, minuto, segundo;
}; // fin de la clase Hora
```

Constructores

Los constructores se utilizan para inicializar los datos miembros de una clase

Constructores

Los constructores se utilizan para inicializar los datos miembros de una clase

- ▶ Al crear un objeto de una clase se pueden inicializar sus miembros mediante la función constructor

Constructores

Los constructores se utilizan para inicializar los datos miembros de una clase

- ▶ Al crear un objeto de una clase se pueden inicializar sus miembros mediante la función constructor
- ▶ El constructor es una función miembro especial que tiene el mismo nombre que la clase y no devuelve un tipo de dato

Constructores

Los constructores se utilizan para inicializar los datos miembros de una clase

- ▶ Al crear un objeto de una clase se pueden inicializar sus miembros mediante la función constructor
- ▶ El constructor es una función miembro especial que tiene el mismo nombre que la clase y no devuelve un tipo de dato
- ▶ Los constructores pueden estar sobrecargados para producir distintas maneras de inicializar a los miembros de una clase

Constructores – Argumentos predeterminados

```
// Definición del tipo de dato abstracto Hora
class Hora {
public:
    Hora(int = 0, int = 0, int = 0); // Constructor predeterminado
    // Demás funciones miembros

private:
    // Datos miembros
};
```

Constructores – Argumentos predeterminados

```
// Definición del tipo de dato abstracto Hora
class Hora {
public:
    Hora(int = 0, int = 0, int = 0); // Constructor predeterminado
    // Demás funciones miembros

private:
    // Datos miembros
};
```

- ▶ Los constructores pueden contener argumentos predeterminados

Constructores – Argumentos predeterminados

```
// Definición del tipo de dato abstracto Hora
class Hora {
public:
    Hora(int = 0, int = 0, int = 0); // Constructor predeterminado
    // Demás funciones miembros

private:
    // Datos miembros
};
```

- ▶ Los constructores pueden contener argumentos predeterminados
- ▶ Así se garantiza inicializar los datos miembros a un estado consistente incluso si no se proporcionan valores al constructor

Constructores – Argumentos predeterminados

```
// Definición del tipo de dato abstracto Hora
class Hora {
public:
    Hora(int = 0, int = 0, int = 0); // Constructor predeterminado
    // Demás funciones miembros

private:
    // Datos miembros
};
```

- ▶ Los constructores pueden contener argumentos predeterminados
- ▶ Así se garantiza inicializar los datos miembros a un estado consistente incluso si no se proporcionan valores al constructor
- ▶ Un constructor que tiene todos sus argumentos predeterminados (que no requiere argumentos explícitos) se llama *constructor predeterminado*

Constructores – Argumentos predeterminados

```
// Definición del tipo de dato abstracto Hora
class Hora {
public:
    Hora(int = 0, int = 0, int = 0); // Constructor predeterminado
    // Demás funciones miembros

private:
    // Datos miembros
};
```

- ▶ Los constructores pueden contener argumentos predeterminados
- ▶ Así se garantiza inicializar los datos miembros a un estado consistente incluso si no se proporcionan valores al constructor
- ▶ Un constructor que tiene todos sus argumentos predeterminados (que no requiere argumentos explícitos) se llama *constructor predeterminado*
- ▶ Puede existir un único constructor predeterminado

Constructores – Argumentos predeterminados

```
// Definición del tipo de dato abstracto Hora
class Hora {
public:
    Hora(int = 0, int = 0, int = 0); // Constructor predeterminado
    // Demás funciones miembros

private:
    // Datos miembros
};
```

- ▶ Los constructores pueden contener argumentos predeterminados
- ▶ Así se garantiza inicializar los datos miembros a un estado consistente incluso si no se proporcionan valores al constructor
- ▶ Un constructor que tiene todos sus argumentos predeterminados (que no requiere argumentos explícitos) se llama *constructor predeterminado*
- ▶ Puede existir un único constructor predeterminado
- ▶ Si no se define un constructor el compilador crea un constructor predeterminado, el cual no realiza ninguna inicialización

Destructores

- ▶ El *destructor* es otro tipo de función miembro especial de una clase

Destructores

- ▶ El *destructor* es otro tipo de función miembro especial de una clase
- ▶ El nombre del destructor comienza con `~` seguido por el nombre de la clase

Destructores

- ▶ El *destructor* es otro tipo de función miembro especial de una clase
- ▶ El nombre del destructor comienza con ~ seguido por el nombre de la clase
- ▶ El destructor de una clase se ejecuta cuando se destruye un objeto

Destructores

- ▶ El *destructor* es otro tipo de función miembro especial de una clase
- ▶ El nombre del destructor comienza con ~ seguido por el nombre de la clase
- ▶ El destructor de una clase se ejecuta cuando se destruye un objeto
- ▶ El destructor en realidad no destruye el objeto, en realidad realiza una *limpieza final* antes de recuperar la memoria del objeto

Destructores

- ▶ El *destructor* es otro tipo de función miembro especial de una clase
- ▶ El nombre del destructor comienza con `~` seguido por el nombre de la clase
- ▶ El destructor de una clase se ejecuta cuando se destruye un objeto
- ▶ El destructor en realidad no destruye el objeto, en realidad realiza una *limpieza final* antes de recuperar la memoria del objeto
- ▶ Un destructor no recibe parámetros y no devuelve valor alguno

Destruyores

- ▶ El *destructor* es otro tipo de función miembro especial de una clase
- ▶ El nombre del destructor comienza con ~ seguido por el nombre de la clase
- ▶ El destructor de una clase se ejecuta cuando se destruye un objeto
- ▶ El destructor en realidad no destruye el objeto, en realidad realiza una *limpieza final* antes de recuperar la memoria del objeto
- ▶ Un destructor no recibe parámetros y no devuelve valor alguno
- ▶ La sobrecarga de destructores no está permitida

Destruyores

- ▶ El *destructor* es otro tipo de función miembro especial de una clase
- ▶ El nombre del destructor comienza con ~ seguido por el nombre de la clase
- ▶ El destructor de una clase se ejecuta cuando se destruye un objeto
- ▶ El destructor en realidad no destruye el objeto, en realidad realiza una *limpieza final* antes de recuperar la memoria del objeto
- ▶ Un destructor no recibe parámetros y no devuelve valor alguno
- ▶ La sobrecarga de destructores no está permitida
- ▶ Una clase puede tener solo un destructor

Ejecución de constructores y destructores

- ▶ Tanto los constructores como los destructores se ejecutan automáticamente

Ejecución de constructores y destructores

- ▶ Tanto los constructores como los destructores se ejecutan automáticamente
- ▶ El orden de su ejecución depende del momento donde los objetos entran y salen de alcance

Ejecución de constructores y destructores

- ▶ Tanto los constructores como los destructores se ejecutan automáticamente
- ▶ El orden de su ejecución depende del momento donde los objetos entran y salen de alcance
- ▶ En general se ejecutan los destructores en orden inverso a los constructores

Ejecución de constructores y destructores

- ▶ Tanto los constructores como los destructores se ejecutan automáticamente
- ▶ El orden de su ejecución depende del momento donde los objetos entran y salen de alcance
- ▶ En general se ejecutan los destructores en orden inverso a los constructores

Ver código fuente ejemplo de D&D 4^o ed. – `fig16_08.cpp`, `crea.h`, `crea.cpp`

Reutilización de software

- ▶ Al escribir programas orientados a objetos pone énfasis en la implementación de las clases adecuadas

Reutilización de software

- ▶ Al escribir programas orientados a objetos pone énfasis en la implementación de las clases adecuadas
- ▶ Existe gran variedad de bibliotecas de clases en la comunidad de software libre

Reutilización de software

- ▶ Al escribir programas orientados a objetos pone énfasis en la implementación de las clases adecuadas
- ▶ Existe gran variedad de bibliotecas de clases en la comunidad de software libre
- ▶ En la actualidad el software se contruye a partir de bibliotencas existentes, bien probadas y documentadas, portátiles y de alto rendimiento

Reutilización de software

- ▶ Al escribir programas orientados a objetos pone énfasis en la implementación de las clases adecuadas
- ▶ Existe gran variedad de bibliotecas de clases en la comunidad de software libre
- ▶ En la actualidad el software se contruye a partir de bibliotencas existentes, bien probadas y documentadas, portátiles y de alto rendimiento
- ▶ Esto agiliza el desarrollo de software con altas prestaciones y facilita el *desarrollo rápido de aplicaciones*

Ejercicios (D&D 16.5)

Escribir una clase *Complejo* que permita realizar aritmética básica con números complejos, junto a un programa que verifique el correcto funcionamiento.

Los números complejos son de la forma: `parteReal + parteImaginaria * i` donde $i = \sqrt{-1}$

Ejercicios (D&D 16.5)

Escribir una clase *Complejo* que permita realizar aritmética básica con números complejos, junto a un programa que verifique el correcto funcionamiento.

Los números complejos son de la forma: `parteReal + parteImaginaria * i` donde $i = \sqrt{-1}$

- ▶ Utilizar variables `double` para los datos miembros (`private`) de la clase
- ▶ Codifique un constructor que permita inicializar un objeto de la clase
- ▶ Este constructor debe tener valores predeterminados si no se proporcionan argumentos
- ▶ Codifique funciones miembros `public` para
 1. Sumar dos números complejos
 2. Restar dos números complejos
 3. Imprimir los números complejos de la forma (`parteReal, parteImaginaria`)

Ejercicios (D&D 16.5)

Escribir una clase *Complejo* que permita realizar aritmética básica con números complejos, junto a un programa que verifique el correcto funcionamiento.

Los números complejos son de la forma: `parteReal + parteImaginaria * i` donde $i = \sqrt{-1}$

- ▶ Utilizar variables `double` para los datos miembros (`private`) de la clase
- ▶ Codifique un constructor que permita inicializar un objeto de la clase
- ▶ Este constructor debe tener valores predeterminados si no se proporcionan argumentos
- ▶ Codifique funciones miembros `public` para
 1. Sumar dos números complejos
 2. Restar dos números complejos
 3. Imprimir los números complejos de la forma (`parteReal, parteImaginaria`)

La suma del nro complejo $a + b \cdot i$ y $c + d \cdot i$ es $(a + c) + (b + d) \cdot i$

La resta del nro complejo $a + b \cdot i$ y $c + d \cdot i$ es $(a - c) + (b - d) \cdot i$

