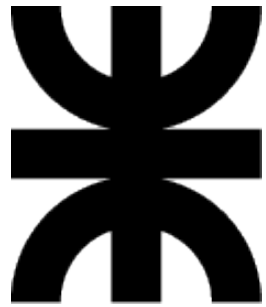
Informática II
Clases en C++

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Declaración de clase en C++

```
class Hora {
  public:
    Hora(); // Constructor
    // Funciones miembros
    void establecer(int, int, int);
    void imprimir();

  private:
    // Miembros datos 'private'
    int hora;
    int minuto;
    int segundo;
};
```

Declaración de clase en C++

```
class Hora {
  public:
    Hora(); // Constructor
    // Funciones miembros
    void establecer(int, int, int);
    void imprimir();

  private:
    // Miembros datos 'private'
    int hora;
    int minuto;
    int segundo;
};
```

- 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, crea un nuevo tipo de dato

Declaración de clase en C++

```
class Hora {
  public:
    Hora(); // Constructor
    // Funciones miembros
    void establecer(int, int, int);
    void imprimir();

  private:
    // Miembros datos 'private'
    int hora;
    int minuto;
    int segundo;
};

Hora atardecer,
  arregloDeHoras[5],
  *apuntadorAHora = nullptr,
  &horaCena = atardecer;
```

- 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, crea un nuevo tipo de dato

Declaración de clase en C++

```
class Hora {
  public:
    Hora(); // Constructor
    // Funciones miembros
    void establecer(int, int, int);
    void imprimir();

  private:
    // Miembros datos 'private'
    int hora;
    int minuto;
    int segundo;
};

Hora atardecer,
  arregloDeHoras[5],
  *apuntadorAHora = nullptr,
  &horaCena = atardecer;
```

- 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, 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:
    Hora(); // Constructor
    // Funciones miembros
    void establecer(int, int, int);
    void imprimir();

  private:
    // Miembros datos 'private'
    int hora;
    int minuto;
    int segundo;
};

Hora atardecer,
  arregloDeHoras[5],
  *apuntadorAHora = nullptr,
  &horaCena = atardecer;
```

- 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, 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 (cuerpo de la clase)

Declaración de clase en C++

Inicialización de datos miembros:

- No se puede inicializar en el lugar donde se declaran (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 (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 (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 (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 usuarios tienen acceso a la interfaz de la clase, pero no deben tener acceso a la implementación de la clase

Declaración de clase en C++

Inicialización de datos miembros:

- No se puede inicializar en el lugar donde se declaran (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 usuarios tienen acceso a la interfaz de la clase, pero no deben tener acceso a la implementación de la clase

¿Cómo se logra?

Separación de interfaz e implementación

Archivo 'hora.h' (basado en D&D 4° ed.)

```
1  // Declaración de la clase Hora.
2  // Las funciones miembro están definidas en hora.cpp
3
4  // Evita inclusiones múltiples del archivo cabecera
5  #ifndef HORA_H
6  #define HORA_H
7
8  class Hora {
9    public:
10     Hora(); // constructor
11     void establecer(int, int, int); // establece hora, minuto, segundo
12     void imprimir(); // imprime la hora
13
14   private:
15     int hora; // 0 - 23
16     int minuto; // 0 - 59
17     int segundo; // 0 - 59
18 };
19
20 #endif
```

Separación de interfaz e implementación

Archivo 'hora.cpp' (basado en D&D 4° ed.)

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

Separación de interfaz e implementación

- Se compila la clase

```
> g++ -c hora.cpp
```

Separación de interfaz e implementación

- Se compila la clase
  ```
  > g++ -c hora.cpp
  ```
- Se compila la aplicación
  ```
  > g++ -c hora_main.cpp
  ```

Separación de interfaz e implementación

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

Separación de interfaz e implementación

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

O bien, se compila todo junto
`> g++ hora.cpp hora_main.cpp`

Separación de interfaz e implementación

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

O bien, se compila todo junto
`> g++ hora.cpp hora_main.cpp`

Ejecución:

```
> ./a.out
La hora inicial es: 00:00:00
La hora despues de establecer es: 13:27:06
Después de intentar establecer valores inválidos: 00:00:00
```

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:

- 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:

- No todas las funciones miembros necesitan ser públicas
- Algunas funciones miembros 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 , int , int ); // constructor

    // funciones establecer
    void estabecer(int, int, int); // establece hora, minuto, segundo
    void establecerHora(int); // establece hora
    void establecerMinuto(int); // establece minuto
    void establecerSegundo(int); // establece segundo

    // funciones obtener
    int obtenerHora(); // devuelve hora
    int obtienerMinuto(); // devuelve minuto
    int obtienerSegundo(); // devuelve segundo

  private:
    int hora, minuto, segundo;
};
```

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

```
class Hora {
  public:
    Hora(int = 0, int = 0, int = 0); // Constructor predeterminado
    // Demás funciones miembros

  private:
  // Datos miembros
};
```

Constructores – Argumentos predeterminados

```
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

```
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 proporcional valores al constructor

Constructores – Argumentos predeterminados

```
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 proporcional valores al constructor
- Un constructor que tiene todos sus argumentos predeterminados (que no requiere argumentos explícitos) se llama *constructor predeterminado*

Constructores – Argumentos predeterminados

```
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 proporcional 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

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

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
- La sobrecarga de destructores no está permitida

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
- 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° ed. – `fig16_08.cpp`, `crea.h`, `crea.cpp`

Constructor de copia

- Se invoca cuando es necesario copiar un objeto a otro, por ejemplo:

Constructor de copia

- Se invoca cuando es necesario copiar un objeto a otro, por ejemplo:
  1. en llamada a funciones por valor,

Constructor de copia

- Se invoca cuando es necesario copiar un objeto a otro, por ejemplo:
  1. en llamada a funciones por valor,
  2. cuando se devuelve por valor un objeto desde una función, o

Constructor de copia

- Se invoca cuando es necesario copiar un objeto a otro, por ejemplo:
  1. en llamada a funciones por valor,
  2. cuando se devuelve por valor un objeto desde una función, o
  3. cuando se inicializa un objeto que es copia de otro de la misma clase

Constructor de copia

- Se invoca cuando es necesario copiar un objeto a otro, por ejemplo:
  1. en llamada a funciones por valor,
  2. cuando se devuelve por valor un objeto desde una función, o
  3. cuando se inicializa un objeto que es copia de otro de la misma clase
- Se llama en una definición cuando se instancia y se inicializa un objeto

```
Hora hora2(hora1);
```

Constructor de copia

- Se invoca cuando es necesario copiar un objeto a otro, por ejemplo:
  1. en llamada a funciones por valor,
  2. cuando se devuelve por valor un objeto desde una función, o
  3. cuando se inicializa un objeto que es copia de otro de la misma clase
- Se llama en una definición cuando se instancia y se inicializa un objeto

```
Hora hora2(hora1);
```

  o de forma equivalente

```
Hora hora2 = hora1;
```

Constructor de copia

- Se invoca cuando es necesario copiar un objeto a otro, por ejemplo:
  1. en llamada a funciones por valor,
  2. cuando se devuelve por valor un objeto desde una función, o
  3. cuando se inicializa un objeto que es copia de otro de la misma clase
- Se llama en una definición cuando se instancia y se inicializa un objeto

  ```
  Hora hora2(hora1);
  ```

  o de forma equivalente

  ```
  Hora hora2 = hora1;
  ```

- El constructor de copia debe tener como parámetro una referencia al objeto a copiar, no un objeto (llamada por valor).
- De lo contrario, el constructor de copia dará como resultado una *recursividad infinita*, ya que en una llamada por valor se debe pasar una copia del objeto al constructor copia lo que da como resultado que se ejecute el constructor copia de manera recursiva.

Reutilización de software

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

Reutilización de software

- Al escribir programas orientados a objetos se 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 se 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 se 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*.