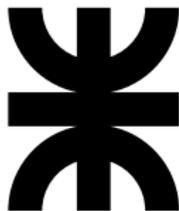


Informática II

Programación orientada a objetos en C++

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2019 –

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

¿Está bien la declaración de variables?

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

► Palabra reservada **struct**

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

¿Está bien la declaración de variables?

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

- ▶ Palabra reservada **struct**
- ▶ Identificador **Hora** como *etiqueta* de la estructura

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

¿Está bien la declaración de variables?

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

- ▶ Palabra reservada **struct**
- ▶ Identificador **Hora** como *etiqueta* de la estructura
- ▶ La etiqueta sirve para declarar variables

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

¿Está bien la declaración de variables?

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

- ▶ Palabra reservada **struct**
- ▶ Identificador **Hora** como *etiqueta* de la estructura
- ▶ La etiqueta sirve para declarar variables
- ▶ El nombre del nuevo tipo es **Hora** a diferencia de C que sería **struct Hora**

¿Está bien la declaración de variables?

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

- ▶ Palabra reservada **struct**
- ▶ Identificador **Hora** como *etiqueta* de la estructura
- ▶ La etiqueta sirve para declarar variables
- ▶ El nombre del nuevo tipo es **Hora** a diferencia de C que sería **struct Hora**
- ▶ Miembros de estructura: **hora**, **minuto** y **segundo**

¿Está bien la declaración de variables?

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

- ▶ Palabra reservada **struct**
- ▶ Identificador **Hora** como *etiqueta* de la estructura
- ▶ La etiqueta sirve para declarar variables
- ▶ El nombre del nuevo tipo es **Hora** a diferencia de C que sería **struct Hora**
- ▶ Miembros de estructura: **hora**, **minuto** y **segundo**
- ▶ Los miembros deben tener nombres únicos

¿Está bien la declaración de variables?

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

- ▶ Palabra reservada **struct**
- ▶ Identificador **Hora** como *etiqueta* de la estructura
- ▶ La etiqueta sirve para declarar variables
- ▶ El nombre del nuevo tipo es **Hora** a diferencia de C que sería **struct Hora**
- ▶ Miembros de estructura: **hora**, **minuto** y **segundo**
- ▶ Los miembros deben tener nombres únicos
- ▶ Diferentes estructuras pueden tener igual nombre de miembro

¿Está bien la declaración de variables?

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

- ▶ Palabra reservada **struct**
- ▶ Identificador **Hora** como *etiqueta* de la estructura
- ▶ La etiqueta sirve para declarar variables
- ▶ El nombre del nuevo tipo es **Hora** a diferencia de C que sería **struct Hora**
- ▶ Miembros de estructura: **hora**, **minuto** y **segundo**
- ▶ Los miembros deben tener nombres únicos
- ▶ Diferentes estructuras pueden tener igual nombre de miembro

- ▶ La definición de estructura termina con punto y coma

¿Está bien la declaración de variables?

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

- ▶ Palabra reservada **struct**
- ▶ Identificador **Hora** como *etiqueta* de la estructura
- ▶ La etiqueta sirve para declarar variables
- ▶ El nombre del nuevo tipo es **Hora** a diferencia de C que sería **struct Hora**
- ▶ Miembros de estructura: **hora**, **minuto** y **segundo**
- ▶ Los miembros deben tener nombres únicos
- ▶ Diferentes estructuras pueden tener igual nombre de miembro

¿Está bien la declaración de variables?

- ▶ La definición de estructura termina con punto y coma
- ▶ La definición no reserva espacio en memoria, sino que crea un nuevo tipo de dato

Ejemplo: Hora (con tipos de datos de C)

Implementar un tipo de dato para almacenar la hora (hora-minuto-segundo)

```
struct Hora {  
    int hora; // 0 - 23  
    int minuto; // 0 - 59  
    int segundo; // 0 - 59  
};
```

```
Hora horaCena, arrayHora[10];  
Hora *ptrHora;
```

- ▶ Palabra reservada **struct**
- ▶ Identificador **Hora** como *etiqueta* de la estructura
- ▶ La etiqueta sirve para declarar variables
- ▶ El nombre del nuevo tipo es **Hora** a diferencia de C que sería **struct Hora**
- ▶ Miembros de estructura: **hora**, **minuto** y **segundo**
- ▶ Los miembros deben tener nombres únicos
- ▶ Diferentes estructuras pueden tener igual nombre de miembro

¿Está bien la declaración de variables?

- ▶ La definición de estructura termina con punto y coma
- ▶ La definición no reserva espacio en memoria, sino que crea un nuevo tipo de dato
- ▶ ¿Cómo sería el prototipo de una función para cargar la hora?

Ejemplo: Hora (con tipos de datos de C)

```
1 horaCena.hora = 18;
2 horaCena.minuto = 30;
3 horaCena.segundo = 0;
4 // Imprimir Hora
5 . . .
6 horaCena.hora = 25;
7 horaCena.minuto = 84;
8 horaCena.segundo = 107;
9 // Imprimir Hora
```

Ejemplo: Hora (con tipos de datos de C)

```
1 horaCena.hora = 18;
2 horaCena.minuto = 30;
3 horaCena.segundo = 0;
4 // Imprimir Hora
5 . . .
6 horaCena.hora = 25;
7 horaCena.minuto = 84;
8 horaCena.segundo = 107;
9 // Imprimir Hora
```

- ▶ Es posible tener datos sin inicializar dado que no es obligatorio

Ejemplo: Hora (con tipos de datos de C)

```
1 horaCena.hora = 18;
2 horaCena.minuto = 30;
3 horaCena.segundo = 0;
4 // Imprimir Hora
5 . . .
6 horaCena.hora = 25;
7 horaCena.minuto = 84;
8 horaCena.segundo = 107;
9 // Imprimir Hora
```

- ▶ Es posible tener datos sin inicializar dado que no es obligatorio
- ▶ Aún si se inicializan pueden estar inicializados de forma incorrecta

Ejemplo: Hora (con tipos de datos de C)

```
1 horaCena.hora = 18;
2 horaCena.minuto = 30;
3 horaCena.segundo = 0;
4 // Imprimir Hora
5 . . .
6 horaCena.hora = 25;
7 horaCena.minuto = 84;
8 horaCena.segundo = 107;
9 // Imprimir Hora
```

- ▶ Es posible tener datos sin inicializar dado que no es obligatorio
- ▶ Aún si se inicializan pueden estar inicializados de forma incorrecta
- ▶ A los miembros de la **struct** se les puede asignar datos inválidos porque el programa tiene acceso a los datos

Ejemplo: Hora (con tipos de datos de C)

```
1 horaCena.hora = 18;
2 horaCena.minuto = 30;
3 horaCena.segundo = 0;
4 // Imprimir Hora
5 . . .
6 horaCena.hora = 25;
7 horaCena.minuto = 84;
8 horaCena.segundo = 107;
9 // Imprimir Hora
```

- ▶ Es posible tener datos sin inicializar dado que no es obligatorio
- ▶ Aún si se inicializan pueden estar inicializados de forma incorrecta
- ▶ A los miembros de la **struct** se les puede asignar datos inválidos porque el programa tiene acceso a los datos
- ▶ Si se modifica la implementación de la **struct** se deberán modificar los programas que la utilizan

Ejemplo: Hora (con tipos de datos de C)

```
1 horaCena.hora = 18;
2 horaCena.minuto = 30;
3 horaCena.segundo = 0;
4 // Imprimir Hora
5 . . .
6 horaCena.hora = 25;
7 horaCena.minuto = 84;
8 horaCena.segundo = 107;
9 // Imprimir Hora
```

- ▶ Es posible tener datos sin inicializar dado que no es obligatorio
- ▶ Aún si se inicializan pueden estar inicializados de forma incorrecta
- ▶ A los miembros de la **struct** se les puede asignar datos inválidos porque el programa tiene acceso a los datos
- ▶ Si se modifica la implementación de la **struct** se deberán modificar los programas que la utilizan
- ▶ Esto se debe a que el programador está manipulando los datos de forma directa

Ejemplo: Hora (con tipos de datos de C)

```
1 horaCena.hora = 18;
2 horaCena.minuto = 30;
3 horaCena.segundo = 0;
4 // Imprimir Hora
5 . . .
6 horaCena.hora = 25;
7 horaCena.minuto = 84;
8 horaCena.segundo = 107;
9 // Imprimir Hora
```

- ▶ Es posible tener datos sin inicializar dado que no es obligatorio
- ▶ Aún si se inicializan pueden estar inicializados de forma incorrecta
- ▶ A los miembros de la **struct** se les puede asignar datos inválidos porque el programa tiene acceso a los datos
- ▶ Si se modifica la implementación de la **struct** se deberán modificar los programas que la utilizan
- ▶ Esto se debe a que el programador está manipulando los datos de forma directa
- ▶ No puede imprimirse la estructura como una unidad

Ejemplo: Hora (con tipos de datos de C)

```
1 horaCena.hora = 18;
2 horaCena.minuto = 30;
3 horaCena.segundo = 0;
4 // Imprimir Hora
5 . . .
6 horaCena.hora = 25;
7 horaCena.minuto = 84;
8 horaCena.segundo = 107;
9 // Imprimir Hora
```

- ▶ Es posible tener datos sin inicializar dado que no es obligatorio
- ▶ Aún si se inicializan pueden estar inicializados de forma incorrecta
- ▶ A los miembros de la **struct** se les puede asignar datos inválidos porque el programa tiene acceso a los datos
- ▶ Si se modifica la implementación de la **struct** se deberán modificar los programas que la utilizan
- ▶ Esto se debe a que el programador está manipulando los datos de forma directa
- ▶ No puede imprimirse la estructura como una unidad
- ▶ No se puede comparar una estructura con otra (sino miembro a miembro)

Ejemplo: Hora (con clase de C++)

```
1 #include <iostream>
2 using namespace std;
3
4 class Hora {
5     int hora; // 0 - 23
6     int minuto; // 0 - 59
7     int segundo; // 0 - 59
8
9     Hora();
10    void imprimir();
11 };
12
13 int main() {
14     // instancia el objeto h de la
15     // clase Hora
16     Hora h;
17
18     cout << "La hora es ";
19     h.imprimir();
20     return 0;
21 }
```

Ejemplo: Hora (con clase de C++)

```
1 #include <iostream>
2 using namespace std;
3
4 class Hora {
5     int hora; // 0 - 23
6     int minuto; // 0 - 59
7     int segundo; // 0 - 59
8
9     Hora();
10    void imprimir();
11 };
12
13 int main() {
14     // instancia el objeto h de la
15     // clase Hora
16     Hora h;
17
18     cout << "La hora es ";
19     h.imprimir();
20     return 0;
21 }
```

```
La hora es 00:00:00
```

Ejemplo: Hora (con clase de C++)

```
1 #include <iostream>
2 using namespace std;
3
4 class Hora {
5     int hora; // 0 - 23
6     int minuto; // 0 - 59
7     int segundo; // 0 - 59
8
9     Hora();
10    void imprimir();
11 };
12
13 int main() {
14     // instancia el objeto h de la
15     // clase Hora
16     Hora h;
17
18     cout << "La hora es ";
19     h.imprimir();
20     return 0;
21 }
```

► ¿Cómo se obtiene el formato de impresión?

```
La hora es 00:00:00
```

Ejemplo: Hora (con clase de C++)

```
1 #include <iostream>
2 using namespace std;
3
4 class Hora {
5     int hora; // 0 - 23
6     int minuto; // 0 - 59
7     int segundo; // 0 - 59
8
9     Hora();
10    void imprimir();
11 };
12
13 int main() {
14     // instancia el objeto h de la
15     // clase Hora
16     Hora h;
17
18     cout << "La hora es ";
19     h.imprimir();
20     return 0;
21 }
```

► ¿Cómo se obtiene el formato de impresión? ...en imprimir()

```
La hora es 00:00:00
```

Ejemplo: Hora (con clase de C++)

```
1 #include <iostream>
2 using namespace std;
3
4 class Hora {
5     int hora; // 0 - 23
6     int minuto; // 0 - 59
7     int segundo; // 0 - 59
8
9     Hora();
10    void imprimir();
11 };
12
13 int main() {
14     // instancia el objeto h de la
15     // clase Hora
16     Hora h;
17
18     cout << "La hora es ";
19     h.imprimir();
20     return 0;
21 }
```

- ▶ ¿Cómo se obtiene el formato de impresión? ...en imprimir()
- ▶ ¿Donde se inicializa?

```
La hora es 00:00:00
```

Ejemplo: Hora (con clase de C++)

```
1 #include <iostream>
2 using namespace std;
3
4 class Hora {
5     int hora; // 0 - 23
6     int minuto; // 0 - 59
7     int segundo; // 0 - 59
8
9     Hora();
10    void imprimir();
11 };
12
13 int main() {
14     // instancia el objeto h de la
15     // clase Hora
16     Hora h;
17
18     cout << "La hora es ";
19     h.imprimir();
20     return 0;
21 }
```

- ▶ ¿Cómo se obtiene el formato de impresión? ...en imprimir()
- ▶ ¿Donde se inicializa? ...en Hora()

```
La hora es 00:00:00
```

Ejemplo: Hora (con clase de C++)

```
1 #include <iostream>
2 using namespace std;
3
4 class Hora {
5     int hora; // 0 - 23
6     int minuto; // 0 - 59
7     int segundo; // 0 - 59
8
9     Hora();
10    void imprimir();
11 };
12
13 int main() {
14     // instancia el objeto h de la
15     // clase Hora
16     Hora h;
17
18     cout << "La hora es ";
19     h.imprimir();
20     return 0;
21 }
```

- ▶ ¿Cómo se obtiene el formato de impresión? ...en imprimir()
- ▶ ¿Donde se inicializa? ...en Hora()
- ▶ ¿Es posible imprimir con cout?...y comparar dos objetos (==)?

```
La hora es 00:00:00
```

Ejemplo: Hora (con clase de C++)

```
1 #include <iostream>
2 using namespace std;
3
4 class Hora {
5     int hora; // 0 - 23
6     int minuto; // 0 - 59
7     int segundo; // 0 - 59
8
9     Hora();
10    void imprimir();
11 };
12
13 int main() {
14     // instancia el objeto h de la
15     // clase Hora
16     Hora h;
17
18     cout << "La hora es ";
19     h.imprimir();
20     return 0;
21 }
```

- ▶ ¿Cómo se obtiene el formato de impresión? ...en imprimir()
- ▶ ¿Donde se inicializa? ...en Hora()
- ▶ ¿Es posible imprimir con cout?...y comparar dos objetos (==)? ...SI!!!

```
La hora es 00:00:00
```

Ejemplo: Hora (con clase de C++)

```
1 #include <iostream>
2 using namespace std;
3
4 class Hora {
5     int hora; // 0 - 23
6     int minuto; // 0 - 59
7     int segundo; // 0 - 59
8
9     Hora();
10    void imprimir();
11 };
12
13 int main() {
14     // instancia el objeto h de la
15     // clase Hora
16     Hora h;
17
18     cout << "La hora es ";
19     h.imprimir();
20     return 0;
21 }
```

- ▶ ¿Cómo se obtiene el formato de impresión? ...en imprimir()
- ▶ ¿Donde se inicializa? ...en Hora()
- ▶ ¿Es posible imprimir con cout?...y comparar dos objetos (==)? ...SI!!!

ERROR al compilar!!!

```
La hora es 00:00:00
```


Definición de clase en C++

```
1 class Hora { // Definición del tipo de dato abstracto (ADT) Hora
2
3     Hora(); // constructor
4
5     void imprimeMilitar(); // imprime la hora en formato militar
6
7
8
9     int hora; // 0 - 23
10    int minuto; // 0 - 59
11    int segundo; // 0 - 59
12 }; // fin de la clase Hora
```

Definición de clase en C++

```
1 class Hora { // Definición del tipo de dato abstracto (ADT) Hora
2   public:
3     Hora(); // constructor
4
5     void imprimeMilitar(); // imprime la hora en formato militar
6
7
8   private:
9     int hora; // 0 - 23
10    int minuto; // 0 - 59
11    int segundo; // 0 - 59
12 }; // fin de la clase Hora
```

Definición de clase en C++

```
1 class Hora { // Definición del tipo de dato abstracto (ADT) Hora
2   public:
3     Hora(); // constructor
4     void estableceHora(int, int, int); // establece hora, minuto, segundo
5     void imprimeMilitar(); // imprime la hora en formato militar
6
7
8   private:
9     int hora; // 0 - 23
10    int minuto; // 0 - 59
11    int segundo; // 0 - 59
12 }; // fin de la clase Hora
```

Definición de clase en C++

```
1 class Hora { // Definición del tipo de dato abstracto (ADT) Hora
2   public:
3     Hora(); // constructor
4     void estableceHora(int, int, int); // establece hora, minuto, segundo
5     void imprimeMilitar(); // imprime la hora en formato militar
6     void imprimeEstandar(); // imprime la hora en formato estándar
7
8   private:
9     int hora; // 0 - 23
10    int minuto; // 0 - 59
11    int segundo; // 0 - 59
12 }; // fin de la clase Hora
```

Definición de clase en C++

```
1 class Hora { // Definición del tipo de dato abstracto (ADT) Hora
2   public:
3     Hora(); // constructor
4     void estableceHora(int, int, int); // establece hora, minuto, segundo
5     void imprimeMilitar(); // imprime la hora en formato militar
6     void imprimeEstandar(); // imprime la hora en formato estándar
7
8   private:
9     int hora; // 0 - 23
10    int minuto; // 0 - 59
11    int segundo; // 0 - 59
12 }; // fin de la clase Hora
```

- ▶ Palabra reservada `class`. Bloque entre `{` y `}`. Punto y coma al final

Definición de clase en C++

```
1 class Hora { // Definición del tipo de dato abstracto (ADT) Hora
2   public:
3     Hora(); // constructor
4     void estableceHora(int, int, int); // establece hora, minuto, segundo
5     void imprimeMilitar(); // imprime la hora en formato militar
6     void imprimeEstandar(); // imprime la hora en formato estándar
7
8   private:
9     int hora; // 0 - 23
10    int minuto; // 0 - 59
11    int segundo; // 0 - 59
12 }; // fin de la clase Hora
```

- ▶ Palabra reservada `class`. Bloque entre `{` y `}`. Punto y coma al final
- ▶ Miembros datos y funciones miembros o *interfaz* de la clase

Definición de clase en C++

```
1 class Hora { // Definición del tipo de dato abstracto (ADT) Hora
2   public:
3     Hora(); // constructor
4     void estableceHora(int, int, int); // establece hora, minuto, segundo
5     void imprimeMilitar(); // imprime la hora en formato militar
6     void imprimeEstandar(); // imprime la hora en formato estándar
7
8   private:
9     int hora; // 0 - 23
10    int minuto; // 0 - 59
11    int segundo; // 0 - 59
12 }; // fin de la clase Hora
```

- ▶ Palabra reservada `class`. Bloque entre `{` y `}`. Punto y coma al final
- ▶ Miembros datos y funciones miembros o *interfaz* de la clase
- ▶ Etiquetas `public:` y `private:` –*especificadores de acceso a miembros*

Definición de clase en C++

```
1 class Hora { // Definición del tipo de dato abstracto (ADT) Hora
2   public:
3     Hora(); // constructor
4     void estableceHora(int, int, int); // establece hora, minuto, segundo
5     void imprimeMilitar(); // imprime la hora en formato militar
6     void imprimeEstandar(); // imprime la hora en formato estándar
7
8   private:
9     int hora; // 0 - 23
10    int minuto; // 0 - 59
11    int segundo; // 0 - 59
12 }; // fin de la clase Hora
```

- ▶ Palabra reservada `class`. Bloque entre `{` y `}`. Punto y coma al final
- ▶ Miembros datos y funciones miembros o *interfaz* de la clase
- ▶ Etiquetas `public:` y `private:` –*especificadores de acceso a miembros*
- ▶ Función miembro con el mismo nombre de la clase: *constructor*

Definición de clase en C++

```
1 class Hora { // Definición del tipo de dato abstracto (ADT) Hora
2   public:
3     Hora(); // constructor
4     void estableceHora(int, int, int); // establece hora, minuto, segundo
5     void imprimeMilitar(); // imprime la hora en formato militar
6     void imprimeEstandar(); // imprime la hora en formato estándar
7
8   private:
9     int hora; // 0 - 23
10    int minuto; // 0 - 59
11    int segundo; // 0 - 59
12 }; // fin de la clase Hora
```

- ▶ Palabra reservada `class`. Bloque entre `{` y `}`. Punto y coma al final
- ▶ Miembros datos y funciones miembros o *interfaz* de la clase
- ▶ Etiquetas `public:` y `private:` –*especificadores de acceso a miembros*
- ▶ Función miembro con el mismo nombre de la clase: *constructor*
- ▶ No se especifica ningún tipo de retorno en el constructor

Definición de clase en C++

```
1 class Hora { // Definición del tipo de dato abstracto (ADT) Hora
2   public:
3     Hora(); // constructor
4     void estableceHora(int, int, int); // establece hora, minuto, segundo
5     void imprimeMilitar(); // imprime la hora en formato militar
6     void imprimeEstandar(); // imprime la hora en formato estándar
7
8   private:
9     int hora; // 0 - 23
10    int minuto; // 0 - 59
11    int segundo; // 0 - 59
12 }; // fin de la clase Hora
```

Una vez declarada la clase, se pueden definir objetos de la misma, arreglos, punteros y referencias

```
Hora atardecer; // Objeto
Hora arregloDeHoras[5]; // Arreglo de objetos
Hora *apuntadorAHora; // Puntero a objeto
Hora &horaCenar = atardecer; // Referencia
```

Ejemplo de aplicación

```
1 #include <iostream>
2 // using namespace, y definición de clase
3
4 int main() {
5     Hora h; // instancia el objeto h de la clase Hora
6
7     cout << "La hora en militar inicial es ";
8     h.imprimeMilitar();
9     cout << "\nLa hora estándar inicial es ";
10    h.imprimeEstandar();
11
12
13
14
15
16
17
18
19
20
21
22    return 0;
23 } // fin de la función main
```

Ejemplo de aplicación

```
1 #include <iostream>
2 // using namespace, y definición de clase
3
4 int main() {
5     Hora h; // instancia el objeto h de la clase Hora
6
7     cout << "La hora en militar inicial es ";
8     h.imprimeMilitar();
9     cout << "\nLa hora estándar inicial es ";
10    h.imprimeEstandar();
11
12    h.estableceHora(13, 27, 6);
13    // Imprimir nuevamente
14
15
16
17
18
19
20
21
22    return 0;
23 } // fin de la función main
```

Ejemplo de aplicación

```
1 #include <iostream>
2 // using namespace, y definición de clase
3
4 int main() {
5     Hora h; // instancia el objeto h de la clase Hora
6
7     cout << "La hora en militar inicial es ";
8     h.imprimeMilitar();
9     cout << "\nLa hora estándar inicial es ";
10    h.imprimeEstandar();
11
12    h.estableceHora(13, 27, 6);
13    // Imprimir nuevamente
14
15    h.estableceHora(99, 99, 99); // intenta establecer valores inválidos
16    cout << "\nDespués de intentar establecer valores inválidos:"
17         << "\nHora militar: ";
18    h.imprimeMilitar();
19    cout << "\nHora estándar: ";
20    h.imprimeEstandar();
21    cout << endl;
22    return 0;
23 } // fin de la función main
```

Ejemplo de aplicación

Salida del programa anterior

```
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
```

Ejemplo de aplicación

Salida del programa anterior

```
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 hacen las funciones miembros?

Ejemplo de aplicación

Salida del programa anterior

```
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 hacen las funciones miembros?
- ▶ ¿Dónde y cómo se define el cuerpo de las funciones miembros?

Ejemplo de aplicación

```
1 // El constructor Hora inicializa en cero a cada dato miembro.
2 // Garantiza que todos los objetos de Hora inicial en un estado
3 // consistente.
4 Hora::Hora()
5 {
6     hora = minuto = segundo = 0;
7 }
8
9
10
11
12
13
14
15
16
17
```

Ejemplo de aplicación

```
1 // El constructor Hora inicializa en cero a cada dato miembro.
2 // Garantiza que todos los objetos de Hora inicial en un estado
3 // consistente.
4 Hora::Hora()
5 {
6     hora = minuto = segundo = 0;
7 }
8
9 // Establece un nuevo valor de Hora por medio de la hora militar.
10 // Realiza verificaciones de validación de los valores de los datos.
11 // Establece en cero a los valores inválidos.
12 void Hora::estableceHora(int h, int m, int s)
13 {
14     hora = (h >= 0 && h < 24) ? h : 0;
15     minuto = (m >= 0 && m < 60) ? m : 0;
16     segundo = (s >= 0 && s < 60) ? s : 0;
17 } // fin de la función estableceHora
```

Ejemplo de aplicación

```
17 // Imprime Hora en formato militar
18 void Hora::imprimeMilitar()
19 {
20     cout << (hora < 10 ? "0" : "") << hora << ":"
21         << (minuto < 10 ? "0" : "") << minuto << ":"
22         << (segundo < 10 ? "0" : "") << segundo;
23
24 } // fin de la función imprimeMilitar
25
26
27
28
29
30
31
32
33
```

Ejemplo de aplicación

```
17 // Imprime Hora en formato militar
18 void Hora::imprimeMilitar()
19 {
20     cout << (hora < 10 ? "0" : "") << hora << ":"
21         << (minuto < 10 ? "0" : "") << minuto << ":"
22         << (segundo < 10 ? "0" : "") << segundo;
23
24 } // fin de la función imprimeMilitar
25
26 // Imprime Hora en formato estándar
27 void Hora::imprimeEstandar()
28 {
29     cout << ( (hora == 0 || hora == 12) ? 12 : hora % 12)
30         << ":" << (minuto < 10 ? "0" : "") << minuto
31         << ":" << (segundo < 10 ? "0" : "") << segundo
32         << (hora < 12 ? " AM" : " PM");
33 } // fin de la función imprimeEstandar
```

Ejemplo de aplicación

```
17 // Imprime Hora en formato militar
18 void Hora::imprimeMilitar()
19 {
20     cout << (hora < 10 ? "0" : "") << hora << ":"
21         << (minuto < 10 ? "0" : "") << minuto << ":"
22         << (segundo < 10 ? "0" : "") << segundo;
23
24 } // fin de la función imprimeMilitar
25
26 // Imprime Hora en formato estándar
27 void Hora::imprimeEstandar()
28 {
29     cout << ( (hora == 0 || hora == 12) ? 12 : hora % 12)
30         << ":" << (minuto < 10 ? "0" : "") << minuto
31         << ":" << (segundo < 10 ? "0" : "") << segundo
32         << (hora < 12 ? " AM" : " PM");
33 } // fin de la función imprimeEstandar
```

Ver código fuente ejemplo fig16_02.cpp del D&D 4º ed.

Programación orientada a objetos en C++

- ▶ La programación orientada a objetos (POO) encapsula datos (atributos) y funciones (comportamiento) en paquetes llamados clases.
- ▶ Los datos y las funciones de una clase están íntimamente ligados entre sí.

Programación orientada a objetos en C++

- ▶ La programación orientada a objetos (POO) encapsula datos (atributos) y funciones (comportamiento) en paquetes llamados clases.
- ▶ Los datos y las funciones de una clase están íntimamente ligados entre sí.

Lenguaje C vs C++:

En C la programación tiende a ser orientada a acciones (procedimientos)

Programación orientada a objetos en C++

- ▶ La programación orientada a objetos (POO) encapsula datos (atributos) y funciones (comportamiento) en paquetes llamados clases.
- ▶ Los datos y las funciones de una clase están íntimamente ligados entre sí.

Lenguaje C vs C++:

En C la programación tiende a ser orientada a acciones (procedimientos)

En C++ lo ideal es programar con orientación a objetos

Programación orientada a objetos en C++

- ▶ La programación orientada a objetos (POO) encapsula datos (atributos) y funciones (comportamiento) en paquetes llamados clases.
- ▶ Los datos y las funciones de una clase están íntimamente ligados entre sí.

Lenguaje C vs C++:

En C la programación tiende a ser orientada a acciones (procedimientos)

En C++ lo ideal es programar con orientación a objetos

En C la unidad de procesamiento es la *función*

Programación orientada a objetos en C++

- ▶ La programación orientada a objetos (POO) encapsula datos (atributos) y funciones (comportamiento) en paquetes llamados clases.
- ▶ Los datos y las funciones de una clase están íntimamente ligados entre sí.

Lenguaje C vs C++:

En C la programación tiende a ser orientada a acciones (procedimientos)

En C++ lo ideal es programar con orientación a objetos

En C la unidad de procesamiento es la *función*

En C++ la unidad de programación es la *clase* (se instancia en objetos)

Programación orientada a objetos en C++

Clases en C++

- ▶ Contienen datos así como un conjunto de funciones que manipulan estos datos

Programación orientada a objetos en C++

Clases en C++

- ▶ Contienen datos así como un conjunto de funciones que manipulan estos datos
- ▶ A los datos que componen una clase se les llaman *datos miembros*

Programación orientada a objetos en C++

Clases en C++

- ▶ Contienen datos así como un conjunto de funciones que manipulan estos datos
- ▶ A los datos que componen una clase se les llaman *datos miembros*
- ▶ A las funciones que componen una clase se les llama *funciones miembro* (o métodos, en otros lenguajes orientados a objetos)

Programación orientada a objetos en C++

Clases en C++

- ▶ Contienen datos así como un conjunto de funciones que manipulan estos datos
- ▶ A los datos que componen una clase se les llaman *datos miembros*
- ▶ A las funciones que componen una clase se les llama *funciones miembro* (o métodos, en otros lenguajes orientados a objetos)
- ▶ Así como a una instancia de un tipo de dato predefinido (p.e. `int`) se le llama variable, a una instancia de un tipo de dato definido por el usuario (instancia de una clase) se le llama *objeto*

Programación orientada a objetos en C++

- ▶ La representación de los datos dentro de la clase no le concierne al usuario

Programación orientada a objetos en C++

- ▶ La representación de los datos dentro de la clase no le concierne al usuario
- ▶ Por ejemplo, la clase `Hora` podría almacenar la hora internamente como la cantidad de segundos desde la media noche

Programación orientada a objetos en C++

- ▶ La representación de los datos dentro de la clase no le concierne al usuario
- ▶ Por ejemplo, la clase `Hora` podría almacenar la hora internamente como la cantidad de segundos desde la media noche
- ▶ Los usuarios de la clase pueden utilizar las funciones miembros públicas y obtener los mismos resultados

Programación orientada a objetos en C++

- ▶ La representación de los datos dentro de la clase no le concierne al usuario
- ▶ Por ejemplo, la clase `Hora` podría almacenar la hora internamente como la cantidad de segundos desde la media noche
- ▶ Los usuarios de la clase pueden utilizar las funciones miembros públicas y obtener los mismos resultados
- ▶ Se dice entonces que la *implementación de la clase* está oculta al usuario

Programación orientada a objetos en C++

- ▶ La representación de los datos dentro de la clase no le concierne al usuario
- ▶ Por ejemplo, la clase `Hora` podría almacenar la hora internamente como la cantidad de segundos desde la media noche
- ▶ Los usuarios de la clase pueden utilizar las funciones miembros públicas y obtener los mismos resultados
- ▶ Se dice entonces que la *implementación de la clase* está oculta al usuario

Los usuarios de una clase tienen acceso a la interfaz de la clase pero no deben tener acceso a la implementación de la clase (encapsulamiento)

Actividad práctica

1. Modificar la *Implementación* de la clase **Hora** del ejemplo anterior de forma tal que internamente la hora este almacenada como la cantidad de segundos desde la media noche. La *Interfaz* de la clase **Hora** debe ser igual al ejemplo.

