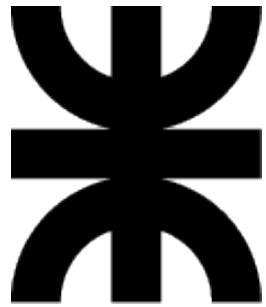
Informática II
Programación orientada a objetos en C++

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

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 definición de las 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 definir variables.
- El nombre del nuevo tipo es `Hora` a diferencia de C que sería `struct Hora`.

¿Está bien la definición de las 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;
```

¿Está bien la definición de las variables?

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

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 definición de las variables?

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

- La declaración de estructura termina con punto y coma.
- La declaración no reserva espacio en memoria, crea un nuevo tipo de dato.

Ejemplo: Hora (con tipos de datos de C)

¿Cómo sería el prototipo de una función para cargar la hora? (modificar un miembro? imprimir?)

```
void cargarHora(Hora & , int , int , int );
void establecerHora(Hora & , int );
void establecerMinuto(Hora & , int );
void imprimirhora(const Hora & );
```

Ejemplo: Hora (con tipos de datos de C)

¿Cómo sería el prototipo de una función para cargar la hora? (modificar un miembro? imprimir?)

```
void cargarHora(Hora & , int , int , int );
void establecerHora(Hora & , int );
void establecerMinuto(Hora & , int );
void imprimirhora(const Hora & );
```

```
horaCena.hora = 18;
horaCena.minuto = 30;
horaCena.segundo = 0;
// Imprimir Hora
. . .
horaCena.hora = 25;
horaCena.minuto = 84;
horaCena.segundo = 107;
// Imprimir Hora
```

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

Ejemplo: Hora (con tipos de datos de C)

¿Cómo sería el prototipo de una función para cargar la hora? (modificar un miembro? imprimir?)

```
void cargarHora(Hora & , int , int , int );
void establecerHora(Hora & , int );
void establecerMinuto(Hora & , int );
void imprimirhora(const Hora & );
```

```
horaCena.hora = 18;
horaCena.minuto = 30;
horaCena.segundo = 0;
// Imprimir Hora
. . .
horaCena.hora = 25;
horaCena.minuto = 84;
horaCena.segundo = 107;
// 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)

¿Cómo sería el prototipo de una función para cargar la hora? (modificar un miembro? imprimir?)

```
void cargarHora(Hora & , int , int , int );
void establecerHora(Hora & , int );
void establecerMinuto(Hora & , int );
void imprimirhora(const Hora & );
```

```
horaCena.hora = 18;
horaCena.minuto = 30;
horaCena.segundo = 0;
// Imprimir Hora
. . .
horaCena.hora = 25;
horaCena.minuto = 84;
horaCena.segundo = 107;
// 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 clase de C++)

```cpp
#include <iostream>

class Hora {
    int hora; // 0 - 23
    int minuto; // 0 - 59
    int segundo; // 0 - 59

    Hora();
    void imprimir();
};

int main() {
  // instancia el objeto h de la
  // clase Hora
  Hora h;

  cout << "La hora es ";
  h.imprimir();
  return 0;
}
```

Ejemplo: Hora (con clase de C++)

```cpp
#include <iostream>

class Hora {
    int hora; // 0 - 23
    int minuto; // 0 - 59
    int segundo; // 0 - 59

    Hora();
    void imprimir();
};

int main() {
  // instancia el objeto h de la
  // clase Hora
  Hora h;

  cout << "La hora es ";
  h.imprimir();
  return 0;
}
```

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

Ejemplo: Hora (con clase de C++)

```cpp
#include <iostream>

class Hora {
    int hora; // 0 - 23
    int minuto; // 0 - 59
    int segundo; // 0 - 59

    Hora();
    void imprimir();
};

int main() {
  // instancia el objeto h de la
  // clase Hora
  Hora h;

  cout << "La hora es ";
  h.imprimir();
  return 0;
}
```

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

- ¿Cómo se obtiene el formato de impresión?

Ejemplo: Hora (con clase de C++)

```cpp
#include <iostream>

class Hora {
    int hora; // 0 - 23
    int minuto; // 0 - 59
    int segundo; // 0 - 59

    Hora();
    void imprimir();
};

int main() {
  // instancia el objeto h de la
  // clase Hora
  Hora h;

  cout << "La hora es ";
  h.imprimir();
  return 0;
}
```

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

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

Ejemplo: Hora (con clase de C++)

```cpp
#include <iostream>

class Hora {
    int hora; // 0 - 23
    int minuto; // 0 - 59
    int segundo; // 0 - 59

    Hora();
    void imprimir();
};

int main() {
  // instancia el objeto h de la
  // clase Hora
  Hora h;

  cout << "La hora es ";
  h.imprimir();
  return 0;
}
```

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

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

Ejemplo: Hora (con clase de C++)

```cpp
#include <iostream>

class Hora {
    int hora; // 0 - 23
    int minuto; // 0 - 59
    int segundo; // 0 - 59

    Hora();
    void imprimir();
};

int main() {
  // instancia el objeto h de la
  // clase Hora
  Hora h;

  cout << "La hora es ";
  h.imprimir();
  return 0;
}
```

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

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

Ejemplo: Hora (con clase de C++)

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

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

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

Ejemplo: Hora (con clase de C++)

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

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

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

Ejemplo: Hora (con clase de C++)

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

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

- ¿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!!!
(programa incompleto)

Declaración de clase en C++

```
1  class Hora {
2
3      Hora(); // constructor
4
5      void imprimir(); // imprime la hora
6
7
8      int hora; // 0 - 23
9      int minuto; // 0 - 59
10     int segundo; // 0 - 59
11 };
```

Declaración de clase en C++

```
1  class Hora {
2    public:
3      Hora(); // constructor
4
5      void imprimir(); // imprime la hora
6
7    private:
8      int hora; // 0 - 23
9      int minuto; // 0 - 59
10     int segundo; // 0 - 59
11 };
```

Declaración de clase en C++

```
1  class Hora {
2    public:
3      Hora(); // constructor
4      void establecer(int, int, int); // establece hora, minuto, segundo
5      void imprimir(); // imprime la hora
6
7    private:
8      int hora; // 0 - 23
9      int minuto; // 0 - 59
10     int segundo; // 0 - 59
11 };
```

Declaración de clase en C++

```
1  class Hora {
2    public:
3      Hora(); // constructor
4      void establecer(int, int, int); // establece hora, minuto, segundo
5      void imprimir(); // imprime la hora
6
7    private:
8      int hora; // 0 - 23
9      int minuto; // 0 - 59
10     int segundo; // 0 - 59
11 };
```

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

Declaración de clase en C++

```
1  class Hora {
2    public:
3      Hora(); // constructor
4      void establecer(int, int, int); // establece hora, minuto, segundo
5      void imprimir(); // imprime la hora
6
7    private:
8      int hora; // 0 - 23
9      int minuto; // 0 - 59
10     int segundo; // 0 - 59
11 };
```

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

Declaración de clase en C++

```
1  class Hora {
2    public:
3      Hora(); // constructor
4      void establecer(int, int, int); // establece hora, minuto, segundo
5      void imprimir(); // imprime la hora
6
7    private:
8      int hora; // 0 - 23
9      int minuto; // 0 - 59
10     int segundo; // 0 - 59
11 };
```

- 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.*

Declaración de clase en C++

```
1  class Hora {
2    public:
3      Hora(); // constructor
4      void establecer(int, int, int); // establece hora, minuto, segundo
5      void imprimir(); // imprime la hora
6
7    private:
8      int hora; // 0 - 23
9      int minuto; // 0 - 59
10     int segundo; // 0 - 59
11 };
```

- 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.*

Declaración de clase en C++

```
1  class Hora {
2    public:
3      Hora(); // constructor
4      void establecer(int, int, int); // establece hora, minuto, segundo
5      void imprimir(); // imprime la hora
6
7    private:
8      int hora; // 0 - 23
9      int minuto; // 0 - 59
10     int segundo; // 0 - 59
11 };
```

- 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.

Declaración de clase en C++

```
1  class Hora {
2    public:
3      Hora(); // constructor
4      void establecer(int, int, int); // establece hora, minuto, segundo
5      void imprimir(); // imprime la hora
6
7    private:
8      int hora; // 0 - 23
9      int minuto; // 0 - 59
10     int segundo; // 0 - 59
11 };
```

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 *ptrAHora = NULL; // Puntero a objeto
Hora &horaCenar = atardecer; // Referencia
```

Ejemplo de aplicación

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

Ejemplo de aplicación

```
1  #include <iostream>
2  // using namespace, y declaración de clase
3
4  int main() {
5    Hora h; // instancia el objeto h de la clase Hora
6
7    cout << "La hora inicial es ";
8    h.imprimir();
9
10   h.establecer(13, 27, 6);
11   cout << "\nLa hora después de establecer es ";
12   h.imprimir();
13
14
15
16
17
18
19
20   return 0;
21 }
```

Ejemplo de aplicación

```cpp
#include <iostream>
// using namespace, y declaración de clase

int main() {
  Hora h; // instancia el objeto h de la clase Hora

  cout << "La hora inicial es ";
  h.imprimir();

  h.establecer(13, 27, 6);
  cout << "\nLa hora después de establecer es ";
  h.imprimir();

  // Intenta establecer valores inválidos
  h.establecer(99, 99, 99);
  cout << "\nDespués de intentar establecer valores inválidos: ";
  h.imprimir();
  cout << endl;

  return 0;
}
```

Ejemplo de aplicación

Salida del programa anterior

```
La hora inicial es 00:00:00
La hora después de establecer es 13:27:06
Después de intentar establecer valores inválidos: 00:00:00
```

Ejemplo de aplicación

Salida del programa anterior

```
La hora inicial es 00:00:00
La hora después de establecer es 13:27:06
Después de intentar establecer valores inválidos: 00:00:00
```

- ¿Qué hacen las funciones miembros?

Ejemplo de aplicación

Salida del programa anterior

```
La hora inicial es 00:00:00
La hora después de establecer es 13:27:06
Después de intentar establecer valores inválidos: 00:00:00
```

- ¿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
10 // Realiza verificaciones de validación de los valores de los datos.
11 // Establece en cero a los valores inválidos.
12 void Hora::establecer(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 }
```

Ejemplo de aplicación

```
18 // Imprime Hora
19 void Hora::imprimir()
20 {
21   cout << (hora < 10 ? "0" : "") << hora << ":"
22     << (minuto < 10 ? "0" : "") << minuto << ":"
23     << (segundo < 10 ? "0" : "") << segundo;
24
25 }
```

Ejemplo de aplicación

```
18 // Imprime Hora
19 void Hora::imprimir()
20 {
21   cout << (hora < 10 ? "0" : "") << hora << ":"
22     << (minuto < 10 ? "0" : "") << minuto << ":"
23     << (segundo < 10 ? "0" : "") << segundo;
24
25 }
```

(el código fuente está basado en el ejemplo `fig16_02.cpp` del D&D 4° ed.)

struct Hora (C) vs. class Hora (C++)

```
horaCena.hora = 18;
horaCena.minuto = 30;
horaCena.segundo = 0;
// Imprimir Hora
. . .
horaCena.hora = 25;
horaCena.minuto = 84;
horaCena.segundo = 107;
// Imprimir Hora
```

```
class Hora {
  public:
    Hora();
    void establecer(int, int, int);
    void imprimimr();

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

`struct Hora` (C) vs. `class Hora` (C++)

```
horaCena.hora = 18;
horaCena.minuto = 30;
horaCena.segundo = 0;
// Imprimir Hora
. . .
horaCena.hora = 25;
horaCena.minuto = 84;
horaCena.segundo = 107;
// Imprimir Hora
```

```
class Hora {
  public:
    Hora();
    void establecer(int, int, int);
    void imprimimr();

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

- Si se modifica la implementación de la `struct` se deberán modificar los programas que la utilizan

`struct Hora` (C) vs. `class Hora` (C++)

```
horaCena.hora = 18;
horaCena.minuto = 30;
horaCena.segundo = 0;
// Imprimir Hora
. . .
horaCena.hora = 25;
horaCena.minuto = 84;
horaCena.segundo = 107;
// Imprimir Hora
```

```
class Hora {
  public:
    Hora();
    void establecer(int, int, int);
    void imprimimr();

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

- 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

`struct Hora` (C) vs. `class Hora` (C++)

```
horaCena.hora = 18;
horaCena.minuto = 30;
horaCena.segundo = 0;
// Imprimir Hora
. . .
horaCena.hora = 25;
horaCena.minuto = 84;
horaCena.segundo = 107;
// Imprimir Hora
```

```
class Hora {
  public:
    Hora();
    void establecer(int, int, int);
    void imprimimr();

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

- 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

`struct Hora` (C) vs. `class Hora` (C++)

```
horaCena.hora = 18;
horaCena.minuto = 30;
horaCena.segundo = 0;
// Imprimir Hora
. . .
horaCena.hora = 25;
horaCena.minuto = 84;
horaCena.segundo = 107;
// Imprimir Hora
```

```
class Hora {
  public:
    Hora();
    void establecer(int, int, int);
    void imprimimr();

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

- 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)

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* (o atributos)

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* (o atributos)
- A las funciones que componen una clase se les llama *funciones miembros* (o métodos)

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* (o atributos)
- A las funciones que componen una clase se les llama *funciones miembros* (o métodos)
- 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)