

# Clases

Algoritmos y Estructuras de Datos II

# Definiciones vs. Declaraciones

## Declaraciones

Asocian un **nombre** a un **tipo**.

Se pueden repetir.

Ejemplos:

```
class Persona;  
bool foo(int x);  
external int x;
```

## Definiciones

Son declaraciones que además le otorgan un **valor** al nombre.

No se pueden repetir.

Ejemplos:

```
int x = 5;  
int y;  
bool foo(int x) {  
    return x == 10;  
};
```

```
class Persona {  
    string nombre;  
    string apellido;  
};
```

# Scopes

- ▶ El **scope** es el alcance de una declaración, es decir, el fragmento del programa en el que dicho nombre es visible.
- ▶ En C++ los scopes se delimitan por llaves `{ . . . }`.
- ▶ Los scopes con nombre se pueden acceder mediante `::`

```
namespace NS {
    int ns1 = 1;
    int ns2 = ns1 + 1;
}

int global1 = 2;
// int global2 = ns1 + 1;
int global2 = NS::ns2;

int foo() {
    int foo1 = global1 + 1;
    {
        int foo2 = 2;
    }
    // int foo3 = foo2 + 1;  error: 'foo2' was not declared in this scope
    int x = 4;
    // int x = 5; redeclaration of 'int x'
    for (int i = 0; i < 5; i++) {
    }
    int i = 10;
}
```

# Clases

Las clases en C++ son herramientas para:  
abstraer comportamiento,  
encapsular información.

## Ejemplo: Horario

Se quiere diseñar un módulo para manipular horarios. La hora debe ser un número entre 0 y 23, y los minutos y segundos son números entre 0 y 59.

¿Qué es un horario?

## Ejemplo: Horario

Se quiere diseñar un módulo para manipular horarios. La hora debe ser un número entre 0 y 23, y los minutos y segundos son números entre 0 y 59.

¿Qué es un horario?

- ▶ **Caracterización por su estructura (representación).**

Un horario está dado por un número de horas, minutos y segundos:

```
int hora; int min; int seg;
```

- ▶ **Caracterización por su comportamiento (interfaz).**

Un horario es una entidad que permite ver cuál es su número de horas, minutos y segundos (y quizá otras operaciones):

```
int Horario::dame_hora();
```

```
int Horario::dame_min();
```

```
int Horario::dame_seg();
```

```
void Horario::restar_seg();
```

# Clases

¿Cómo representamos un horario en C++?



# Clases

¿Cómo representamos un horario en C++?

```
// pre: 0 <= h1 < 24 es 0 <= m1 < 60 es 0 <= s1 < 60 es
```

```
//      0 <= h2 < 24 es 0 <= m2 < 60 es 0 <= s2 < 60
```

```
int segsEntre(int h1, int m1, int s1,  
              int h2, int m2, int s2);
```

```
// pre: |horario1| == 3 es |horario2| == 3 es ...
```

```
int segsEntre(std::vector<int> horario1,  
              std::vector<int> horario2);
```

# Clases

¿Cómo representamos un horario en C++?

```
// pre: 0 <= h1 < 24 es 0 <= m1 < 60 es 0 <= s1 < 60 es  
//      0 <= h2 < 24 es 0 <= m2 < 60 es 0 <= s2 < 60
```

```
int segsEntre(int h1, int m1, int s1,  
              int h2, int m2, int s2);
```

```
// pre: |horario1| == 3 es |horario2| == 3 es ...
```

```
int segsEntre(std::vector<int> horario1,  
              std::vector<int> horario2);
```

```
// typedef <tipo_original> <renombre>  
typedef tuple<int, int, int> Horario;
```

```
int segsEntre(Horario h1, Horario h2);
```

# Clases

```
struct Horario {  
    int hora;  
    int min;  
    int seg;  
};  
  
int segsEntre(Horario h1,  
              Horario h2);  
  
int main() {  
    Horario h1;  
    h1.hora = 10;  
    h1.min = 14;  
    h1.seg = 32;  
  
    Horario h2;  
    h2.hora = 15;  
    h2.min = 0;  
    h2.seg = 22;  
  
    segsEntre(h1, h2);  
}  
  
Horario masTarde(std::vector<Horario> horarios);
```

## Clases: Inicialización

```
int main() {  
    Horario h1;  
    h1.hora = 15;  
    h1.min = 30;  
    h1.seg = 27;  
    cout << h1.hora << endl; // 15  
  
    Horario h2 = h1;  
    cout << h2.min << endl; // 30  
  
    Horario h3;  
    cout << h3.min << endl; // ??  
}
```

## Clases: Inicialización

```
Horario init_horario(int hora, int min, int seg) {  
    Horario h;  
    h.hora = hora;  
    h.min = min;  
    h.seg = seg;  
    return h;  
}
```

```
int main() {  
    Horario h1 = init_horario(15, 30, 27);  
    cout << h1.hora << endl; // 15  
  
    Horario h3;  
    cout << h3.min << endl; // ??  
}
```

# Clases: Constructor

- ▶ Los constructores son funciones especiales para inicializar una nueva instancia de un tipo.
- ▶ Se escriben con el nombre del tipo.
- ▶ No tienen tipo de retorno (está implícito).
- ▶ Tres formas de constructores:
  - ▶ Por defecto: `Horario();`
  - ▶ Por copia: `Horario(Horario otro_horario);`<sup>1</sup>
  - ▶ Con parámetros (el resto):  
`Horario(int _hora, int _min, int _seg);`  
`Horario(int _hora);`
- ▶ Cuando se define un constructor desaparece el constructor sin parámetros implícito (se lo puede definir explícitamente).

---

<sup>1</sup>La aridad es levemente distinta, ver después...

## Constructores: ejemplos

```
struct Horario {  
    Horario (int h, int m, int s);  
    Horario (int h);  
  
    int hora, min, segs;  
}  
  
Horario::Horario(int h, int m, int s)  
    : hora(h % 24), min(m % 60), seg(s % 60) {  
}  
  
Horario::Horario(int h)  
    : hora(h % 24), min(0), segs(0) {  
}
```

# Lista de inicialización

Lista de inicialización

```
Horarios(int h, int m, int s) : hora(h % 24), min(m % 60), seg(s) {  
    seg = s % 60; }  
Horarios(int h) : hora(_hora % 24), min(0), seg(0) {}
```

Más código de inicialización

Lista de inicialización

- ▶ Definen el valor inicial de cada miembro de la estructura llamando a un constructor para su tipo.
- ▶ Son opcionales.



## Constructor: usos

```
int main() {  
    Horario h1 = Horario(15, 30, 27);  
    cout << h1.hora << endl; // 15  
  
    Horario h2(16, 110, 7);  
    cout << h2.min << endl; // 50  
  
    Horario h3(15);  
    cout << h3.min << endl; // 0  
  
    // Horario h4;  
    // error: no matching function  
    // for call to 'Horario::Horario()'  
    // cout << h4.min << endl; // ??  
}
```

## Clases: Visibilidad

```
int main() {  
    Horario h1 = Horario(30, 30, 27);  
    cout << h1.hora << endl; // 6  
  
    h1.hora = 30;  
    cout << h1.hora << endl; // 30  
}
```

# Clases: Visibilidad

Podemos definir qué partes de un tipo son visibles para el usuario usando `private` y `public`.

En tipos declarados con `struct`, se asume por defecto `public`.

En tipos declarados con `class`, se asume por defecto `private`.

En la materia vamos a usar `class` típicamente.

## Clases: Visibilidad

```
class Horario {
private:
    int _hora, _min, _seg;

public:
    Horario(int hora, int min, int seg);

};

Horario::Horario(int hora, int min, int seg)
    : _hora(hora % 24), _min(min % 60), _seg(seg % 60) {
}

int main() {
    Horario h1 = Horario(30, 30, 27);

    // h1._hora = 30;
    // error: 'int Horario::_hora' is private
    // cout << h1._hora << endl;
}
```

# Clases: Métodos

Las clases proveen comportamiento de manera controlada a través de una **interfaz** compuesta por métodos.

La **interfaz** es la capa que media entre el usuario de la clase y el programador de la clase:

- ▶ Usuario: invoca métodos sin conocer la representación interna.
- ▶ Programador: define cómo se implementa cada método usando la representación interna.

## Usuario



## Interfaz



## Implementación



```
class Horario {
public:
    int dame_hora();
    int dame_min();
    int dame_seg();

    Horario(int hora, int min, int seg);
    Horario(int hora);
private:
    int _hora, _min, _seg;
};

int Horario::dame_min() {
    return _min;
}

int Horario::dame_hora() {
    return _hora;
}

int Horario::dame_seg() {
    return _seg;
}
```

## Clases: Métodos

```
int main() {  
    Horario h1 = Horario(30, 30, 27);  
    Horario h2 = Horario(22);  
  
    cout << h1.dame_seg() << endl; // 27  
  
    cout << h2.dame_hora() << endl; // 22  
}
```



## Métodos: contexto de la instancia

```
int x = 10;

int main() {
    Horario h1 = Horario(30, 30, 27); // <<
    Horario h2 = Horario(22);

    cout << h1.dame_seg() + x << endl;

    cout << h2.dame_hora() - x << endl;
}
```

### Contexto

```
int | x | 10
```

## Métodos: contexto de la instancia

```
int x = 10;

int main() {
    Horario h1 = Horario(30, 30, 27);
    Horario h2 = Horario(22); // <<

    cout << h1.dame_seg() + x << endl;

    cout << h2.dame_hora() - x << endl;
}
```

### Contexto

int	x	10
Horario	h1	...

## Métodos: contexto de la instancia

```
int x = 10;

int main() {
    Horario h1 = Horario(30, 30, 27);
    Horario h2 = Horario(22);

    cout << h1.dame_seg() + x << endl; // <<

    cout << h2.dame_hora() - x << endl;
}
```

### Contexto

int	x	10
Horario	h1	...
Horario	h2	...

## Métodos: contexto de la instancia

```
int Horario::dame_seg() {  
    return _seg;  
}
```

### Contexto

int	x	10
int	_hora	30
int	_min	30
int	_seg	27

## Métodos: contexto de la instancia

```
int x = 10;

int main() {
    Horario h1 = Horario(30, 30, 27);
    Horario h2 = Horario(22);

    cout << h1.dame_seg() + x << endl; // 37
    // <<
    cout << h2.dame_hora() - x << endl;
}
```

### Contexto

int	x	10
Horario	h1	...
Horario	h2	...

## Métodos: contexto de la instancia

```
int x = 10;

int main() {
    Horario h1 = Horario(30, 30, 27);
    Horario h2 = Horario(22);

    cout << h1.dame_seg() + x << endl; // 37

    cout << h2.dame_hora() - x << endl; // <<
}
```

### Contexto

int	x	10
Horario	h1	...
Horario	h2	...

## Métodos: contexto de la instancia

```
int Horario::dame_hora() {  
    return _hora;  
}
```

### Contexto

int	x	10
int	_hora	22
int	_min	0
int	_seg	0

## Métodos: contexto de la instancia

```
int x = 10;

int main() {
    Horario h1 = Horario(30, 30, 27);
    Horario h2 = Horario(22);

    cout << h1.dame_seg() + x << endl; // 37

    cout << h2.dame_hora() - x << endl; // 12
}
```

### Contexto

int	x	10
Horario	h1	...
Horario	h2	...



## Clases: Métodos

Los métodos también pueden modificar el estado interno de una instancia.

Supongamos que queremos agregar una operación para decrementar el horario en un segundo.

## Clases: Métodos

```
class Horario {  
    public:  
        Horario(int hora, int min, int seg);  
        int dame_hora();  
        int dame_min();  
        int dame_seg();  
  
        void restar_seg();  
    private:  
        int _hora, _min, _seg;  
};
```

## Clases: Métodos

```
void Horario::restar_seg() {  
    if (_seg > 0) {  
        _seg--;  
    } else if (_seg == 0 and (_min > 0 or _hora > 0)) {  
        _seg = 59;  
        if (_min == 0 and _hora > 0) {  
            _min = 59;  
            _hora--;  
        } else {  
            _min--;  
        }  
    }  
}
```

## Clases: Métodos

```
int main() {  
    Horario h1(0, 0, 30);  
    Horario h2(0, 1, 50);  
  
    cout << h1.dame_seg() << endl; // 30  
    cout << h2.dame_seg() << endl; // 50  
    h1.restar_seg();  
    cout << h1.dame_seg() << endl; // 29  
    h1.restar_seg();  
    cout << h1.dame_seg() << endl; // 28  
  
    h2.restar_seg();  
    cout << h2.dame_seg() << endl; // 49  
    cout << h1.dame_seg() << endl; // 28  
}
```

## Clases dentro de clases

Vamos a crear una clase Recordatorio que represente un recordatorio para el día de hoy.

Debe sonar en un cierto horario y debe tener un mensaje.

```

class Recordatorio {
public:
    Recordatorio(string mensaje, Horario horario);

    Horario horario();
    string mensaje();
private:
    string _mensaje;
    Horario _horario;
}

Recordatorio::Recordatorio(string mensaje, Horario horario)
    : _mensaje(mensaje), _horario(horario) {
}

Horario Recordatorio::horario() {
    return _horario;
}

string Recordatorio::mensaje() {
    return _mensaje;
}

```

Si se encuentran con ...

```
error: no matching function for call to 'Horario::Horario()'
in Recordatorio::Recordatorio(string mensaje, Horario horario)
```

Seguramente no hicieron esto ...

```
Recordatorio::Recordatorio(string mensaje, Horario horario)
    : _mensaje(mensaje), _horario(horario) {
}
```

# Const

Si T es un tipo, `const T` es el tipo de las expresiones **no modificables** de tipo T. Por ejemplo:

```
int main() {  
    const int x = 5;  
    int y = x + 3;  
    y--;  
    // x--; // error: decrement of read-only variable 'x'  
}
```



# Const

La palabra clave `const` tiene varios usos (que ya veremos).  
Otro uso es al final de una declaración de un método de una clase.  
Indica que el método no puede modificar la representación interna.

```
class Horario {  
    public:  
        int dame_hora() const;  
        ...  
  
        void restar_seg() const;  
  
    private:  
        ...  
};
```

```
int Horario::dame_hora() const {  
    return hora;  
}
```

```
/* error: decrement of member 'Horario::_seg' in read-only object  
void Horario::restar_seg() const {  
    _seg--;  
}  
*/
```

# Const

**Nota:** la declaración y la definición del método deben coincidir en la presencia/ausencia del modificador `const`.

```
class Horario {  
    public:  
        int dame_hora() const;  
        int dame_min();  
        ...  
  
    private:  
        ...  
};
```

*// error: prototype for 'int Horario::dame\_hora()' does not match any in  
↪ class 'Horario'*

```
int Horario::dame_hora() {  
    return hora;  
}
```

*// error: prototype for 'int Horario::dame\_min() const' does not match  
↪ any in class 'Horario'*

```
int Horario::dame_min() const {  
    return min;  
}
```

# Const

De hecho, a veces puede ser necesario contar con dos versiones del método (una con y otra sin el modificador `const`). Más adelante vamos a usar esto. Por ejemplo:

```
class Horario {  
    public:  
        Horario restar_seg() const;  
        void restar_seg();  
        ...  
  
    private:  
        ...  
};  
  
void Horario::restar_seg() {  
    ...  
}  
  
Horario Horario::restar_seg() const {  
    ...  
}
```

# Sobrecarga de Operadores

C++ permite *sobrecargar* operadores (ej. -, +, <, >, >=, <=).  
Por ejemplo, para definir operaciones entre horarios:

```
int main() {  
    Horario h1(2, 15, 26);  
    Horario h2(1, 25, 20);  
  
    // Saber si dos horarios son el mismo  
    h1 == h2;  
  
    // Saber si una hora pasa antes que la otra  
    h1 < h2;  
  
    // Conocer el intervalo entre dos horarios  
    Horario d = h1 - h2;  
}
```

## Sobrecarga de Operadores

```
class Horario {  
    public:  
        Horario(int hora, int min, int seg);  
  
        ...  
  
        bool operator<(Horario o) const;  
        bool operator==(Horario o) const;  
  
    private:  
        ...  
};
```

## Sobrecarga de Operadores

```
bool Horario::operator==(Horario o) const {  
    return hora == o.dame_hora()  
        && min == o.dame_min()  
        && segs == o.dame_seg();  
}  
  
bool Horario::operator<(Horario o) const { ... }  
  
Horario operator-(Horario h1, Horario h2) {  
    ...  
}
```

# Referencias

- ▶ Stroustrup, B. (2000). The C++ programming language. Pearson Education India.
- ▶ <https://en.cppreference.com/w/cpp/language/classes>
- ▶ [https://en.cppreference.com/w/cpp/language/initializer\\_list](https://en.cppreference.com/w/cpp/language/initializer_list)
- ▶ [https://en.cppreference.com/w/cpp/language/default\\_constructor](https://en.cppreference.com/w/cpp/language/default_constructor)
- ▶ [https://en.cppreference.com/w/cpp/language/copy\\_constructor](https://en.cppreference.com/w/cpp/language/copy_constructor)
- ▶ <https://en.cppreference.com/w/cpp/language/operators>