

ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

# Los tipos `pair` y `tuple`

Manuel Montenegro Montes

Departamento de Sistemas Informáticos y Computación  
Facultad de Informática – Universidad Complutense de Madrid



# Ejemplo

Esto sirve para hacer que un método pueda devolver más de un resultado. Hay otra alternativa a los parámetros de salida.

- Calcular el elemento mínimo de un array.

```
int min(int *array, int longitud) {  
    int min = std::numeric_limits<int>::max();  
  
    for (int i = 0; i < longitud; i++) {  
        min = std::min(min, array[i]);  
    }  
  
    return min;  
}
```

Coge el máximo valor para no equivocarnos a la hora de coger el valor mínimo.

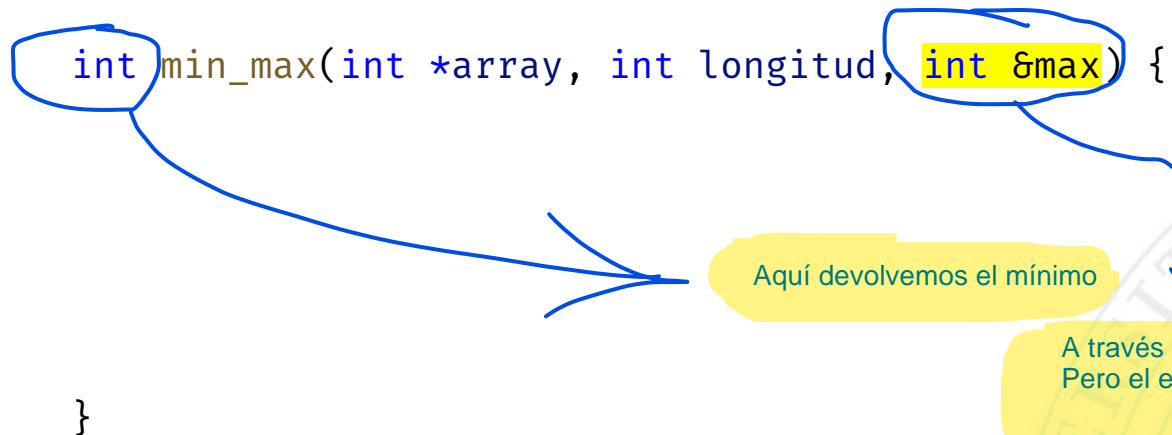
*¿Y si quiero devolver también el máximo?*

Solo devuelve el mínimo, pero...

# Ejemplo

- Calcular el elemento mínimo y máximo de un array.

```
int min_max(int *array, int longitud, int &max) {  
  
}  
}
```



Aquí devolvemos el mínimo

A través de este parámetro pasador por referencia se devuelve el máximo  
Pero el en realidad no quiere que utilicemos parámetros de E/S. ¿Entonces?

Esto queda raro. O bien devolvemos ambos resultados como parámetros de entrada salida o bien como un pair.

# Ejemplo

- Calcular el elemento mínimo y máximo de un array.

Devolvemos a través de parámetros de entrada salida.

```
void min_max(int *array, int longitud, int &min, int &max) {  
    min = std::numeric_limits<int>::max();  
    max = std::numeric_limits<int>::min();  
  
    for (int i = 0; i < longitud; i++) {  
        min = std::min(min, array[i]);  
        max = std::max(max, array[i]);  
    }  
}
```

¿Son parámetros de salida o de E/S?

- Llamadas a función:

```
int min, max;  
min_max(arr, longitud, min, max);
```

Una persona que viene de fuera, que no haya hecho el código y lea esto no va a saber si son parámetros de entrada o de salida

Cuando llamamos al método, esa ambigüedad es mucho mayor porque no sale ni el `umpersant &` ni nada.

# Múltiples resultados

- ¿Cómo podemos especificar varios valores de retorno para una función, sin tener que recurrir a parámetros de salida?

¡CLARO QUE SI, COMPAÑERO!



# Tipo específico

```
struct MinMaxResult {  
    int min;  
    int max;  
};
```

Lo almacenamos a través de un registro (struct) y devolvemos este registro.

```
MinMaxResult min_max(int *array, int longitud) {  
    MinMaxResult res;  
    res.min = std::numeric_limits<int>::max();  
    res.max = std::numeric_limits<int>::min();  
  
    for (int i = 0; i < longitud; i++) {  
        res.min = std::min(res.min, array[i]);  
        res.max = std::max(res.max, array[i]);  
    }  
  
    return res;  
}
```

AQUÍ QUEDARÍA TODO CLARO.

# Tipo específico

- Llamada a la función:

```
MinMaxResult r = min_max(arr, longitud);  
std::cout << "Min = " << r.min << " | Max = " << r.max;
```

- Problema: tener que definir un tipo específico.

Vamos, que solo sirve para ese tipo específico.



Vamos a ver otra alternativa, que es la clase pair, para devolver un par de resultados.

# Pares – `std::pair`



# La clase pair

- Denota un par de elementos  $(x, y)$ , que pueden ser de distinto tipo.
- Definida en el fichero de cabecera `<utility>`

```
template <typename T1, typename T2>
class pair {
public:
    T1 first;
    T2 second;

    pair(const T1 &first, const T2 &second){ ... };
    ...
};
```

está definido en este fichero de cabecera.

# La clase pair

devuelve un par de enteros.

```
std::pair<int, int> min_max(int *array, int longitud) {  
    int min = std::numeric_limits<int>::max();  
    int max = std::numeric_limits<int>::min();  
  
    for (int i = 0; i < longitud; i++) {  
        min = std::min(min, array[i]);  
        max = std::max(max, array[i]);  
    }  
  
    return std::pair<int, int>(min, max);  
}
```

La segunda componente devuelve el máximo.

La primera componente devuelve el mínimo

# La clase pair

```
std::pair<int, int> min_max(int *array, int longitud) {  
    int min = std::numeric_limits<int>::max();  
    int max = std::numeric_limits<int>::min();  
  
    for (int i = 0; i < longitud; i++) {  
        min = std::min(min, array[i]);  
        max = std::max(max, array[i]);  
    }  
  
    return {min, max};  
}
```

Podemos hacer mejor esto que es muchísimo más corto. ¿Por qué se puede hacer esto? Bueno pues porque llama al constructor con parámetros de la clase pair.

# La clase pair

- Llamada a la función:

```
std::pair<int, int> p = min_max(arr, longitud);  
std::cout << "Min = " << p.first << " | Max = " << p.second;
```

- Sintaxis abreviada (*structured binding declaration*) de C++17.

```
auto [min, max] = min_max(arr, longitud);  
std::cout << "Min = " << min << " | Max = " << max << std::endl;
```

- En Visual Studio 2019 es necesario activar la opción `/std:c++17` o `/std:c++latest`.

Me instalo el del 2022 para no tener que acordarme de eso

La primera componente quedaría asignada a la variable min y la segunda a la variable max.

# La clase pair

- Hace explícitos los valores de salida.
- No requiere declarar ninguna clase.
- Pero... conviene documentar el significado de las componentes:

```
// Devuelve un par de enteros.  
// - La primera componente es el valor mínimo del array  
// - La segunda componente es el valor máximo del array
```

IMPORTANTE DECIR A QUÉ HACE REFERENCIA CADA COMPONENTE

```
std::pair<int, int> min_max(int *array, int longitud) {  
    ...  
}
```

*¿Y si la función devuelve más de dos valores?*

Una tupla es un conjunto de valores finito. De forma que podremos devolver el número de valores que nosotros queramos.

## Tuplas – `std::tuple`

# La clase tuple

- Definida en el fichero de cabecera <tuple>

```
// Devuelve una tupla con tres componentes:  
// - La primera componente es el valor mínimo del array  
// - La segunda componente es el valor máximo del array  
// - La tercera componente es la suma de los valores del array
```

En resumen: Además de lo anterior, queremos devolver la suma de los elementos de nuestro array.

```
std::tuple<int, int, int> min_max_sum(int *array, int longitud) {  
    int min = std::numeric_limits<int>::max();  
    int max = std::numeric_limits<int>::min();  
    int sum = 0;  
  
    for (int i = 0; i < longitud; i++) {  
        min = std::min(min, array[i]);  
        max = std::max(max, array[i]);  
        sum += array[i];  
    }  
  
    return {min, max, sum};  
}
```

# La clase tuple

- Llamada:

```
auto [min, max, sum] = min_max_sum(arr, longitud);  
std::cout << "Min = " << min << " | Max = " << max << " | Sum = " << sum;
```

O también por ejemplo podemos utilizar un get. Pero esta sintaxis es bastante más fácil

