

Tuplas & Secuencias y Structs & Arrays

Construcción de Tipo
por Producto Cartesiano

Esp. Ing. José María Sola, profesor.

Revisión 1.0.0-beta.2

2018-06-18

Tabla de contenidos

1. Tuplas & Structs	1
1.1. Punto	1
1.1.1. Matemática	1
1.1.2. C++	1
1.1.3. Prueba	1
1.2. Círculo	2
1.2.1. Matemática	2
1.2.2. C++	2
1.2.3. Prueba	2
1.2.4. Declaración dentro de Declaración	3
1.2.5. Struct Anónimo	3
2. Secuencias & Arrays	5
2.1. Equipo Titular	5
2.2. Struct de Once Miembros unsigned	6
2.2.1. Matemática	6
2.2.2. C++	6
2.2.3. Prueba	7
2.2.4. Declaración Única con Múltiples Declaradores en Diferentes Líneas	7
2.2.5. Declaración Única con Múltiples Declaradores en la Misma Línea	7
2.3. Array de Once Elementos unsigned	7
2.3.1. Matemática	8
2.3.2. C++	8
2.3.3. Prueba	8
2.4. Sinónimo de Array de Once Elementos unsigned	8
2.4.1. Matemática	8
2.4.2. C++	8
2.4.3. Prueba	8
3. Combinaciones	9
3.1. Array de Structs — Array-of-Structs (AoS)	9
3.1.1. Matemática	9
3.1.2. C++	9
3.1.3. Prueba	9

3.2. Struct de Arrays — Struct-of-Arrays (SoA)	10
3.2.1. Matemática	10
3.2.2. C++	10
3.2.3. Prueba	10
3.3. Ejemplo: Héroes	10
3.3.1. Declaración de Tipos	10
3.3.2. Declaración e Inicialización de Variable	11
3.3.3. Prueba	12

Tuplas & Structs

Un *par ordenado* es una *2-upla*. Una *n-upla*, o simplemente *tupla*, es la generalización de las 2-uplas a una cantidad finita no acotada de elementos.

1.1. Punto

1.1.1. Matemática

$$\text{Punto} = \{(x, y) \in \mathbb{R} \times \mathbb{R} = \mathbb{R}^2\}$$

1.1.2. C++

```
struct Punto{ double x, y; }; // <1> Declara el tipo Punto como una
    estructura de dos doubles, llamados x e y.
```

1.1.3. Prueba

```
Punto p{1,2}; // <2> Declara e inicializa p como el Punto que está en la
    coordenada 1 , 2.
assert(1 == p.x); // <3> El operador `.` accede a una miembro de la
    estructura, proyección.
assert(2 == p.y);

Punto origen{0,0};
assert(0 == origen.x);
assert(0 == origen.y);

assert(p.x != origen.x);
```

```
assert(p.x != origen.y);
```

1.2. Círculo

Uno de los elementos de esta tupla es, a su vez, otra tupla.

1.2.1. Matemática

$$\text{Círculo} = \left\{ (\text{radio}, \text{centro}) / \text{radio} \in \mathbb{R} \wedge \text{centro} \in \text{Punto} = \mathbb{R}^2 \right\}$$

$$\text{Punto} = \left\{ (x, y) \in \mathbb{R} \times \mathbb{R} = \mathbb{R}^2 \right\}$$

1.2.2. C++

```
struct Punto{ double x, y; }; // <1> Declaramos el tipo Punto...
struct Círculo{ // <2> y luego el tipo Círculo, ...
    double radio;
    Punto centro; // <2> ... que se basa en el tipo Punto.
};
```

1.2.3. Prueba

```
Círculo c{1,{2,3}};
assert(1 == c.radio );
```

```
assert(2 == c.centro.x);  
assert(3 == c.centro.y);
```

1.2.4. Declaración dentro de Declaración

```
struct Círculo{  
    double radio;  
    struct Punto{ double x, y; } centro; // <1> Declaramos centro y el tipo  
    Punto en la misma declaración.  
};
```

1.2.5. Struct Anónimo

```
struct Círculo{  
    double radio;  
    struct{ double x, y; } centro; // <1> Declaramos centro de tipo struct  
    anónimo.  
};
```

2

Secuencias & Arrays

Una secuencia es un caso particular de tupla. Es el caso dónde todos los elementos tienen el mismo tipo y sólo se diferencian por su posición.

2.1. Equipo Titular

Un equipo titular está formado por 11 jugadores, cada uno identificado por su número de remera, que es un natural en el rango 1 a 23 inclusive.

2.2. Struct de Once Miembros unsigned

2.2.1. Matemática

EquipoTitular = {(jugador1,jugador2,jugador3,jugador4,jug
jugador1,jugador2,jugador3,jugador4,jugador5,jugador6,jug

2.2.2. C++

```
struct EquipoTitular{  
    unsigned jugador1;  
    unsigned jugador2;  
    unsigned jugador3;  
    unsigned jugador4;  
    unsigned jugador5;  
    unsigned jugador6;  
    unsigned jugador7;  
    unsigned jugador8;  
    unsigned jugador9;  
    unsigned jugador10;  
    unsigned jugador11;  
};
```

2.2.3. Prueba

```
EquipoTitular argentina{1,4,6,10,12,15,18,19,20,22,23};
assert( 1 == argentina.jugador1 );
assert( 15 == argentina.jugador6 );
assert( 23 == argentina.jugador11 );
```

2.2.4. Declaración Única con Múltiples Declaradores en Diferentes Líneas

```
struct EquipoTitular{
    unsigned jugador1,
        jugador2,
        jugador3,
        jugador4,
        jugador5,
        jugador6,
        jugador7,
        jugador8,
        jugador9,
        jugador10,
        jugador11;
};
```

2.2.5. Declaración Única con Múltiples Declaradores en la Misma Línea

```
struct EquipoTitular{
    unsigned
    jugador1,jugador2,jugador3,jugador4,jugador5,jugador6,jugador7,jugador8,jugador9,
};
```

2.3. Array de Once Elementos `unsigned`

Las tres versiones anteriores sufren el mismo problema.

2.3.1. Matemática

2.3.2. C++

```
std::array<unsigned,11> argentina{1,4,6,10,12,15,18,19,20,22,23};
```

2.3.3. Prueba

```
assert( 1 == argentina.at(0) ); // <1> La operación `at` retorna el  
    elemento "a" n del la base del arreglo.  
assert( 15 == argentina.at(5) );  
assert( 23 == argentina.at(10) );
```

2.4. Sinónimo de Array de Once Elementos `unsigned`

2.4.1. Matemática

2.4.2. C++

```
using EquipoTitular = std::array<unsigned,11>; // <1> Declara  
    EquipoTitular como sinónimo de array<unsigned,11>
```

La declaración del sinónimo facilita la declaración de nuevas variables.

2.4.3. Prueba

```
EquipoTitular argentina{1,4,6,10,12,15,18,19,20,22,23};  
assert( 1 == argentina.at(0) );  
assert( 15 == argentina.at(5) );  
assert( 23 == argentina.at(10) );  
  
EquipoTitular alemania{1,2,4,5,7,8,9,10,15,17,21};  
assert( alemania.at( 0 ) == argentina.at( 0 ) );  
assert( alemania.at( 7 ) == argentina.at( 3 ) );  
assert( alemania.at(10) != argentina.at(10) );
```

3

Combinaciones

3.1. Array de Structs — Array-of-Structs (AoS)

3.1.1. *Matemática*

$$\text{Punto} = \{(x, y) \in \mathbb{R} \times \mathbb{R} = \mathbb{R}^2\}$$

$$\text{Pentágono} = \text{Punto}^5$$

3.1.2. C++

```
struct Punto{ double x, y; };  
using Pentágono = std::array<Punto,5>;
```

3.1.3. *Prueba*

```
Pentágono p{{ {1,1},{1,4},{2,5},{3,4},{3,1} }};  
assert( 1 == p.at(0).x );  
assert( 1 == p.at(1).x );  
assert( p.at(1).y == p.at(3).y );
```

3.2. Struct de Arrays — Struct-of-Arrays (SoA)

3.2.1. Matemática

$$\text{Punto} = \{(x, y) \in \mathbb{R} \times \mathbb{R} = \mathbb{R}^2\}$$

$$\text{Pentágono} = \{(x, y) / x \in \mathbb{R}^5 \wedge y \in \mathbb{R}^5\}$$

3.2.2. C++

```
struct Pentágono{ std::array<double,5> x, y; }; // <1> Tanto x e y son
arrays de 5 doubles, son arrays paralelos.
```

3.2.3. Prueba

```
Pentágono p{ {1,1,2,3,3}, {1,4,5,4,1} };
assert( 1 == p.x.at(0) );
assert( 1 == p.x.at(1) );
assert( p.y.at(1) == p.y.at(3) );
```

3.3. Ejemplo: Héroes

Ejemplo con varias combinaciones. Hay dos equipos de héroes, cada equipo tiene un nombre y una lista de héroes miembro, cada héroe tiene un alias y un promedio.

3.3.1. Declaración de Tipos

```
struct Héroe{ ❶
    std::string alias;
    double promedio;
};
using Miembros = std::array<Héroe, 3>; ❷

struct Equipo{ ❸
    std::string nombre;
    Miembros miembros;
```

```
};
using Equipos = std::array<Equipo, 2>; ❹
```

- ❶ El tipo héroe es un par formado por el alias y la calificación promedio del héroe.
- ❷ El nombre miembros es un sinónimo del tipo array<Héroes,3>.
- ❸ El tipo Equipo es un par formado por nombre y miembros.
- ❹ El nombre Equipos es un sinónimo del tipo array<Equipo,2>.

3.3.2. Declaración e Inicialización de Variable

```
Equipos equipos{{ ❶❷
{
    "Justice League",
    {{ ❸
        {"Batman",      9.1},
        {"Superman",    9.0},
        {"Wonder woman", 9.9}, ❹
    }}
},
{
    "Avengers",
    {{
        {"Cap",          8.8},
        {"Iron Man",     7.9},
        {"Thor",         8.9},
    }}
}, ❺
}};
```

- ❶ La variable equipos es un array de dos structs, cada struct es un string y un array de 3 structs de string y double.
Anidamiento: equipos = ((nombre, ((alias,promedio).(),())), ())
Jerarquía: equipos→equipo→miembros→héroe→alias
- ❷ Los arrays se inicializan con doble llave.
- ❸ Doble llave para arrays.
- ❹ Las comas finales (*trailing commas*) no dañan, ayudan.
- ❺ Aquí tampoco dañan.

3.3.3. Prueba

```
auto promedioDeSuperman = equipos.at(0).miembros.at(1).promedio;  
auto aliasDelÚltimoAvenger = equipos.at(1).miembros.at(2).alias;  
assert(10 > promedioDeSuperman);  
assert("Thor" == aliasDelÚltimoAvenger);  
assert(equipos.at(0).miembros.at(0).promedio >  
equipos.at(1).miembros.at(1).promedio); ❶
```

❶ *Batman es mejor que Iron Man.*