

Union Type

TPV2
Samir Genaim

¿Qué es Union Type?

```
void foo() {  
    int x;  
    double z;
```

A veces, hay variables que no se usan a la vez durante la ejecución, se usa una o otra depende de alguna condición o en bloques de código distintos, etc.

```
// hacer algo con x, sin usar z
```

```
x = 1;
```

```
...
```

Usa sólo x

```
// hacer algo con z, ya no se usa x
```

```
z = 2.4;
```

```
...
```

Usa sólo z

- ♦ `x` y `z` ocupan `sizeof(int)+sizeof(double)` bytes en la memoria
- ♦ Si podemos decir al compilador que vamos a usar sólo una a la vez, lo que puede hacer es usar la misma dirección en la memoria para ambos y así ocupar sólo `MAX(sizeof(int),sizeof(double))` bytes
- ♦ Union Types nos permite transmitir esa información al compilador

Ejemplo

```
void foo() {  
    union {  
        int x;  
        double z;  
    };  
}
```



Las variables x y z ocupan la misma memoria (8 bytes en lugar de 12). Cuando usamos x usamos sólo 4 bytes, y cuando usamos z usamos los 8 bytes

```
// hacer algo con x, sin usar z  
x = 1;  
...
```

```
// hacer algo con z, ya no se usa x  
z = 2.4;  
...
```

```
}
```

La responsabilidad es tuya ...

```
void foo() {  
    union {  
        int x;  
        double z;  
    };  
  
    x = 3;  
    cout << "x = " << x << endl;  
    cout << "z = " << z << endl;  
}
```

La responsabilidad del uso correcto es tuya, si asignas uno y usas el otro puedes tener resultados inesperados

x = 3

z = 2.122e-314

Union dentro de Struct

```
struct SomeType_1 {  
    int id;  
    double x;  
    int z;  
};
```

```
struct SomeType_2 {  
    int id;  
    union {  
        double x;  
        int z;  
    };  
};
```



```
cout << "sizeof(SomeType_1) = " << sizeof(SomeType_1) << endl;  
cout << "sizeof(SomeType_2) = " << sizeof(SomeType_2) << endl;
```

Suponemos que depende del valor de id, se usa x o z.

Dos implementación, sin y con union

```
sizeof(SomeType_1) = 16  
sizeof(SomeType_2) = 12
```

Union con Nombre

```
struct SomeType {  
    int id;  
    double d;  
    union {  
        double x;  
        int z;  
    } t1;   
  
    union {  
        float x;  
        boolean w;  
    } t2;   
};
```

Si tenemos unions en el mismo struct que comparten nombres de variables, tendremos conflicto. Para resolverlo se puede dar nombres distintos a cada union y usarlos para acceder a sus variables. Un union sin nombre es anónimo.

```
SomeType a;  
a.id = ...  
a.d = ...  
a.t1.x = ...  
a.t1.z = ...  
a.t2.x = ...  
a.t2.w = ...
```

Union para Mensajes en el ECS

Se puede usar para definir el tipo **Message** que usamos en ECS. Un mensaje siempre tiene **id** y depende de ese valor usamos **v1**, o **v2**, etc.

```
struct Message {  
    Uint8 id;  
    union {  
        MsgType1 v1;  
        MsgType2 v2;  
        ...  
    };  
};
```

```
struct MsgType1 {  
    Uint8 side_;  
};
```

```
struct MsgType2 {  
    Uint8 winner_;  
};
```

...

Cuidado al incluir algo que no es de "primitive type", en ese caso hay que definir constructoras por defecto y de copia casi en todas partes ...