Enumeraciones

Construcción de Tipo por Extensión

Esp. Ing. José María Sola, profesor.

Revisión 1.0.0
2018-06-13

Tabla de contenidos

1

Enumeraciones

Una enumeración es un conjunto finito de valores. Los siguientes programas explican el concepto con ejemplos, el último ejemplos utiliza funciones con `enum`s como retornor o argumento.

1.1. Dirección

$$\text{Dirección}=\{\text{Arriba},\text{Abajo},\\ \text{Derecha},\text{Izquierda},\text{Atrás},\text{Adelante}\}$$

```
enum struct Dirección{Arriba, Abajo, Derecha, Izquierda, Atrás,
 Adelante};
Dirección t = Dirección::Arriba;
assert( t == Dirección::Arriba    );
assert( t != Dirección::Abajo     );
assert( t != Dirección::Izquierda );
```

1.2. LuzDeSemáforo

$$\text{LuzDeSemáforo} = \{\text{Rojo}, \text{Amarillo}, \text{Verde}\}$$

```
enum struct LuzDeSemáforo{Rojo, Amarillo, Verde};
assert(0 == static_cast(LuzDeSemáforo::Rojo)    ); ❶ ❷
assert(1 == static_cast(LuzDeSemáforo::Amarillo)); ❸
assert(2 == static_cast(LuzDeSemáforo::Verde)   ); ❹

LuzDeSemáforo unaLuzDeSemáforo, otraLuzDeSemáforo;
```

```
unaLuzDeSemáforo  = LuzDeSemáforo::Verde;
otraLuzDeSemáforo = LuzDeSemáforo::Amarillo;
unaLuzDeSemáforo  = unaLuzDeSemáforo == otraLuzDeSemáforo ?
 LuzDeSemáforo::Rojo : LuzDeSemáforo::Verde;
assert(unaLuzDeSemáforo == LuzDeSemáforo::Verde);
assert(2 == static_cast(unaLuzDeSemáforo)       );
```

❶ Por omisión, el tipo de dato subyacente que el compilador utiliza para representar los valores de una enumeración es `int`.

❷ La expresión del tipo `static_cast<T>(e)` interpreta el valor del la expresión `e` como si fuera del tipo `T`. En este caso interpretamos `Rojo` y `Verde` como `int`s.

❸ El primer valor de la enumeración es cero, y el siguiente uno más que el anterior…

❹ … y así sucesivamente.

1.3. Palo

$$\text{Palo} = \{\text{Copa, Oro, Basto, Espada}\}$$
$$\text{Copa=1, Oro=2, Basto=41, Espada=42}$$

```
enum struct Palo{Copa=1, Oro, Basto=41, Espada}; ❶
assert(  2 == static_cast<int>(Palo::Oro)    );
assert( 42 == static_cast<int>(Palo::Espada) );
```

❶ Es posible explicitar el valor de cada miembro. Esto es útil para abstraernos de una interfaz existente.

1.4. PuntoCardinal

$$\text{PuntoCardinal} = \{\text{Norte, Sur, Este, Oste}\}$$
$$\text{Norte=North, Sur=South, Este=East, Oeste=West}$$

```
enum struct PuntoCardinal{Norte, Sur, Este, Oeste, North=0, South, East,
 West}; ❶
assert( PuntoCardinal::Norte == PuntoCardinal::North            );
assert( PuntoCardinal::Oeste == PuntoCardinal::West             );
assert(              3 == static_cast<int>(PuntoCardinal::Oeste) );
```

```
assert(              3 == static_cast<int>(PuntoCardinal::West)  );
```

❶ Diferentes miembros de una enumeración pueden tener el mismo valor.

1.5. Funciones que Retornan o Reciben Tipos Enum

```
/* EnumFunctions.cpp
JMS
2018 May */
#include <cassert>

enum struct Asignatura{Algoritmos, Sintaxis};
enum struct Turno{Mañana, Tarde, Noche};
enum struct Día{Domingo, Lunes, Martes, Miércoles, Jueves, Viernes,
 Sábado};

Turno TurnoQueCurso(Asignatura);
Día DíaQueCurso(Asignatura);
bool TengoQueCursar(Día, Turno);

int main(){
 assert( Día::Jueves  == DíaQueCurso(Asignatura::Algoritmos));
 assert( Día::Lunes   == DíaQueCurso(Asignatura::Sintaxis)  );

 assert( Turno::Noche == TurnoQueCurso(Asignatura::Sintaxis)  );
 assert( Turno::Noche == TurnoQueCurso(Asignatura::Algoritmos));

 assert(     TengoQueCursar(Día::Lunes,   Turno::Noche));
 assert( not TengoQueCursar(Día::Lunes,   Turno::Tarde));
 assert(     TengoQueCursar(Día::Jueves,  Turno::Noche));
 assert( not TengoQueCursar(Día::Jueves,  Turno::Mañana));
 assert( not TengoQueCursar(Día::Domingo, Turno::Mañana));
}

bool TengoQueCursar(Día d, Turno t){
 return
  DíaQueCurso(Asignatura::Sintaxis) == d and
 TurnoQueCurso(Asignatura::Sintaxis) == t
  or
  DíaQueCurso(Asignatura::Algoritmos) == d and
 TurnoQueCurso(Asignatura::Algoritmos) == t;
}

bool TengoQueCursarBis(Día d, Turno t){
```

```
 return
  d==Día::Lunes  and t==Turno::Noche
  or
  d==Día::Jueves and t==Turno::Noche;
}

Día DíaQueCurso(Asignatura a){return
 a == Asignatura::Algoritmos ? Día::Jueves :
                               Día::Lunes  ;}

Turno TurnoQueCurso(Asignatura a){return Turno::Noche;}
```