C++ 11

ENUM FORWARD DECLARATIONS

&

STRONG ENUMS
MITK C++ 11/14 STATUS

http://mitk.org/wiki/MITK_C++_11/14_Status
PROBLEMS WITH TRADITIONAL Enums

• **name clashes**, if two enumerators in different enums declared in the same scope have the same name;

```cpp
enum Animals {Bear, Cat, Chicken};
enum Birds {Eagle, Duck, Chicken}; // error! Chicken has already been declared!
```

• not possible to use an enumerator with a **fully qualified name**

```cpp
Animals animal = Animals::Cat; //error or warning Animals
animal = Cat; // Ok!
```

• enumerations are **not type-safe**
UNSCOPED ENUMERATION

enum name { enumerator = constexpr, enumerator = constexpr, ...

• declares an unscoped enumeration type whose underlying type is not fixed

Since C++ 11:
enum name : type { enumerator = constexpr, enumerator = constexpr, ...

• declares an unscoped enumeration type whose underlying type is fixed
EXAMPLE
TRADITIONAL Enums

```c
enum Color { red, yellow, green=20, blue }

Color col = red;
int n = blue; // n == 21
```

```c
struct Foo {
    enum Direction { left='l', right='r' }
};
```

```c
Foo x;
int a = Foo::left;
int b = x.left;
int c = Foo::Direction::left; // allowed only in C++11 and later
```
SCOPED ENUMERATIONS

```cpp
enum struct class name { enumerator = constexpr, enumerator = constexpr, ... }
```

- declares a scoped enumeration type whose **underlying type** is **int**

```cpp
enum struct class name : type { enumerator = constexpr, enumerator = constexpr, ... }
```

- declares a scoped enumeration type whose **underlying type** is **type**
EXAMPLE
IMPLICIT CONVERSION

```cpp
enum class Color { RED, GREEN=20, BLUE };  

Color r = Color::BLUE;  

switch(r) {  
    case Color::RED : std::cout << "red" "\n"; 
        break;  
    case Color::GREEN : std::cout << "green" "\n"; 
        break;  
    case Color::BLUE : std::cout << "blue" "\n"; 
        break;  
}

int n = r; // error: no scoped enum to int conversion  
int n = static_cast<int>(r); // OK, n = 21
```
EXAMPLE
ENUM FORWARD DECLARATION

• underlying type can be specified, so forward declaration is possible

```cpp
class Selection : unsigned char {
    None,
    Single,
    Multiple,
};
```
CONCLUSION

What's new:

• New type of enums: strongly-typed enums
• Solve the known problems with the traditional enums
  • the scope of the enumerators
  • possibility to specify the underlying type (also traditional)
• type-safe
• Forward-declaring
REFERENCES