Scoped Enumerations

In this lesson, we'll discuss how enumerations are used to define custom data-type containing integer constants.

WE'LL COVER THE FOLLOWING

Enumerations allow us to define a custom data-types containing **integer constants**. These integer constants are called **enumerators**. Sadly, classical enums have a few drawbacks.

Drawbacks of enumerations in classical C++

Three of the biggest drawbacks of classical enumerations are listed below:

1. Enumerators implicitly convert to `int`.
2. Enumerators in the enclosing scope.
3. The type of the enumeration cannot be specified.

First, let's focus on 3. Enumerations cannot be forward declared because their type is unknown. There is only one guarantee for enumerations in classical C++: the type must be integral and big enough to hold the enumerator values.

Points 1 and 2 are more surprising. Have a look at the code below:

```
// enumClassic.cpp

#include <iostream>

int main(){

  std::cout << std::endl;

  enum Colour{red= 0,green= 2,blue};
```

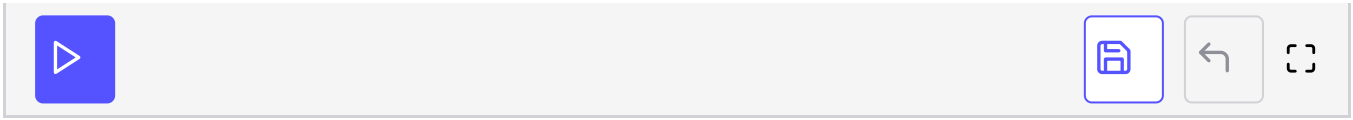
```
  std::cout << "red: " << red << std::endl;
  std::cout << "green: " << green << std::endl;
  std::cout << "blue: " << blue << std::endl;

  int red2= red;

  std::cout << "red2: " << red2 << std::endl;

  // int red= 5;  ERROR

}
```



On the one hand are the enumerators `red`, `green`, and `blue` in the enclosing scope. Therefore, the definition of the variable red in line 19 is not possible. On the other hand, red can be implicitly converted to `int`.

If we don't give a name to an enumeration like `enum{red, green, blue}`, the enumerators will be introduced in the enclosing scope.

But that surprise ends with C++11.

Strongly-typed enumerations

Scoped enumerations are often called strongly-typed enumerations. The strongly-typed enumerations have to follow stronger rules:

1. The enumerators can only be accessed in the scope of the enumeration.
2. The enumerators don't implicitly convert to `int`.
3. The enumerators aren't imported into the enclosing scope.
4. The type of the enumerators is by default `int`. Therefore, we can forward declare the enumeration.

The syntactical difference between classic enumerations and strongly-typed enumerations is minimal. Strongly-typed enumerations additionally use the keywords `class` or `struct`.

If we want to use an enumerator as an `int`, we must explicitly convert it with `static_cast`.

```
// enumCast.cpp
```

```
#include <iostream>

enum OldEnum{
  one= 1,
  ten=10,
  hundred=100,
  thousand= 1000
};

enum struct NewEnum{
  one= 1,
  ten=10,
  hundred=100,
  thousand= 1000
};

int main(){

  std::cout << std::endl;

  std::cout << "C++11= " << 2*thousand + 0*hundred + 1*ten + 1*one << std::endl;
  std::cout << "C++11= " << 2*static_cast<int>(NewEnum::thousand) +
                            0*static_cast<int>(NewEnum::hundred) +
                            1*static_cast<int>(NewEnum::ten) +
                                  1*static_cast<int>(NewEnum::one) << std::endl;

}
```

In order to calculate or print the enumerators, we have to convert them into integral types. Neither the addition nor the output of strongly-typed enumerations are defined.

> In this lesson, we have discussed classical versus strongly-typed enumerations. Commonly, these are known as unscoped and scoped enumerations, respectively.

Let’s look at an example of scoped enumerations in the next lesson.