Compile-Time Counter Using Template and Constexpr Magic

Barbara Geller & Ansel Sermersheim

Lightning Talk - CPPCon 2015

Things to Discuss

- Constexpr
- Templates
- Method Overloading

Our Goal

- cs_register() will do something and then call the "next cs_register" method

```
cs_register(0) {
   cs_register(1);
}

cs_register(1) {
   cs_register(2);
}
```

Implementation

- “zero” and “one” are integer values
- method overloading is based on data types
- create a class template to wrap the int value

```
cs_register(0) {
   cs_register(1);
}
```

Constexpr

- constexpr expressions evaluated at compile time
- foo is initialized to 42 at compile time
- without constexpr the array size would be invalid

```
static constexpr int foo = 30 + 12;
char data[foo];
```

Templates

- Templates allow you to pass a data type as a parameter to a class, method, or function
- Can you pass an int as a template parameter?
- Yes, passing an integer to a class template creates a unique data type (by instantiating the template)

Template Class with an Integer argument

```
template<int N>
class CSInt : public CSInt<N - 1> {
   public:
      static constexpr const int value = N;
};

template<>
class CSInt<0> {
   public:
      static constexpr const int value = 0;
};

// inheritance relationship, “3” inherits from “2”,
“2” inherits from “1”, and “1” inherits from “0”
```

Example Class (after preprocessing)

```
class Ginger : public QObject
{
  public:
    template<int N>
    static void cs_register(CSInt<N>) { }

    static constexpr CSInt<0> cs_counter(CSInt<0>);



// this code is expanded from a macro which is called
// at the beginning of your class
```

Example Class (after preprocessing)

```
// macro expansion from line 42   CS_TOKENPASTE2(value_, __LINE__)
static constexpr const int value_42 =
   decltype(cs_counter(CSInt<255>{}))::value;

static constexpr CSInt<value_42 + 1> cs_counter(CSInt<value_42 + 1>);
// additional code . . .


// macro expansion from line 43   CS_TOKENPASTE2(value_, __LINE__)
static constexpr const int value_43 =
    decltype(cs_counter(CSInt<255>{}))::value;

static constexpr CSInt<value_43 + 1> cs_counter(CSInt<value_43 + 1>);
// additional code . . .


// what is value_42 ?  what is value_43 ?
```

Using the Counter Value

```
// retrieve current counter value of “zero”
static constexpr const int value_42 =
    decltype(cs_counter(CSInt<255>{}))::value;

// increment counter value, declare “cs_counter(1)”
static constexpr CSInt<value_42 + 1> cs_counter(CSInt<value_42 + 1>);

// setup “cs_register(0)”
static void cs_register(CSInt<value_42>)
{
  cs_class::staticMetaObject().register_method( v1, v2,
    QMetaMethod::Slot, v4, QMetaMethod::Public );

  cs_register(CSInt<value_42 + 1>{} );
}

// retrieve current counter value of “one” . . .
```

Using the Counter Value

```
// cs_counter() can only “see” above this point
static constexpr const int value_42 =
    decltype(cs_counter(CSInt<255>{}))::value;

//
static constexpr CSInt<value_42 + 1> cs_counter(CSInt<value_42 + 1>);

// cs_register() can “see” the entire class
static void cs_register(CSInt<value_42>)
{
  cs_class::staticMetaObject().register_method( v1, v2,
    QMetaMethod::Slot, v4, QMetaMethod::Public );

  cs_register(CSInt<value_42 + 1>{} );
}
```

Source, Binaries, Forums

- www.copperspice.com
- download.copperspice.com
- forum.copperspice.com

- ansel@copperspice.com
- barbara@copperspice.com