Intermediate Level C++

C++11: New Features Part 2

Yong Zhang

Objectives

In this chapter you will learn:

- Range-based for Loops
- Null Pointers
- Eums
- Raw Literals
- Initializing Non-static Member Variables
- Object construction
- Explicit Conversion Operator

Range-based For Loop

- **Range-based loop for STL containers**

```
vector<int> v { 1, 2, 3, 4, 5 };
for (auto e : v)
{
    cout << e << endl;
}
```

```
for (auto& e : v)
{
    e += 5;
    cout << e << endl;
}
```

- **Range-based loop for initializer list**

```
auto list = { 100, 200, 300, 400, 500 };
for (auto e : list)
{
    cout << e << endl;
}
```

Range-based For Loop (continued)

- **Range-based loop for customized classes**

```
class MyContainer
{
private:
    list<int> myvalues;
public:
    MyContainer() : myvalues({ 111, 222, 333 }) {}
    friend list<int>::iterator begin(MyContainer& c);
    friend list<int>::iterator end(MyContainer& c);

};

list<int>::iterator begin(MyContainer& c)
{
    return c.myvalues.begin();
}

list<int>::iterator end(MyContainer& c)
{
    return c.myvalues.end();
}
```

- **Demo: RangeBasedForLoopDemo**

Range-based For Loop (continued)

- **Range-based loop and its equivalence**

```
for (elem_decl : seq)
    statement;
```

Member-version begin and end

```
for (auto iter = seq.begin(), seq_end = seq.end(); iter != seq_end; ++iter)
{
    elem_decl = *iter;
    statement;
}
```

Non-Member-version begin and end

```
for (auto iter = begin(seq), seq_end = end(seq); iter != seq_end; ++iter)
{
    elem_decl = *iter;
    statement;
}
```

nullptr

- **void pointer: better to use function overloading**

```
void* vptr = &i;
int *anotherp = reinterpret_cast<int*>(vptr);
```

- **Pointer Constants**

```
int intarray[5] = { 1, 2, 3, 4, 5 };
cout << *(intarray++) << endl; // this will not work
```

- **Null pointer: help to avoid confusion between integers and null pointers.**

```
int* ptr3 = nullptr;
ASSERT(ptr3 == ptr1);
ASSERT(ptr3 == ptr2);
int *ptr4 {};
```

- **Demo: NullptrDemo**

enum

- **Scoped enum: enum class**
- **Specify the type for enums**
- **Forwarding declaration**
  - has to include the type implicitly or explicitly
- **Demo: EnumDemo**

Literals

- **Literals != constants**
  - Constants = literals or named literals
- **C++11 support new features of literals:**
  - Unicode literals (no)
  - Raw literals (yes)
  - Customized literals (no)

Raw Literals

- **To include a special character, you need to escape it.**
  - A double quote in a string literal: \"
  - A backslash in a string literal: \\
- **Raw literals cannot contain any special characters.**
  - **R"(anything \ you " want to include but parentheses)"**
  - **R"(anything \ you) want to include)"**
  - **R"***(anything \ you)" want to include)***"**
- **This allows to have things what would normally be escaped.**
- **Demo: LiteralsDemo**

Initializing Non-static Member Variables

- **Demo: InitialNonStaticMemberDemo**

Object Construction

- **If you do not write constructors, the compiler writes the copy, move, and default constructor for you.**
- **If you write one, the compiler will not provide any more, then you need to write a lot of constructors**
- **Constructors often share similar code for initializing member variables or for acquiring resources.**
- **C++11 New Features for constructors:**
  - Call one constructor from another (delegating constructor)
  - Let compiler write some for us (default constructor)
  - Take constructor away (deleted constructor)
- **Demo: ConstructorDemo**

Explicit Conversion Operator

- **explicit keyword on a constructor prevents it being used unless you ask for it.**
- **explicit keyword can now be used on operators to prevent it being called without your consent.**
- **Demo: ExplicitConversionOperatorDemo**

New Features Supported in Microsoft Visual Studio 2013

- **See Document:**
  - Visual Studio Support For C++11 Features (Modern C++).pdf