Practical Convenient C++17 Language Improvements

Giovanni Dicanio
AUTHOR, SOFTWARE ENGINEER

https://blogs.msmvps.com/gdicanio

Overview

Nested namespaces

Variable declarations in *if* and *switch*

if constexpr

Structured bindings

Namespaces

Nested Namespaces

```
namespace PluralsightEngine {




} // PluralsightEngine
```

Engine classes

Nested Namespaces

```
namespace PluralsightEngine {
  namespace Graphics {



  } // Graphics
} // PluralsightEngine
```

Graphics classes

Nested Namespaces

```
namespace PluralsightEngine {
  namespace Graphics {
    namespace Rendering {



    } // Rendering
  } // Graphics
} // PluralsightEngine
```

Rendering classes

Nested Namespaces

```
namespace PluralsightEngine {
  namespace Graphics {
    namespace Rendering {
      class OpenGLRender
      …
    } // Rendering
  } // Graphics
} // PluralsightEngine
```

```
namespace PluralsightEngine::Graphics::Rendering {
  class OpenGLRender
  …
}
```

Much simpler!!

C++17 Nested Namespaces

Find and Replace Strings

```
vector<string> names{ /* Some names … */ };


// Find and replace "Connie" with "***"

const auto it = find(begin(names), end(names), "Connie");
```

String to search

«C++11 from Scratch»
Iterators and Sorting

bit.ly/VecSort

Find and Replace Strings

```
vector<string> names{ /* Some names … */ };


// Find and replace "Connie" with "***"
const auto it = find(begin(names), end(names), "Connie");
```

String to search

String *not* found

Find and Replace Strings

```cpp
vector<string> names{ /* Some names … */ };


// Find and replace "Connie" with "***"
const auto it = find(begin(names), end(names), "Connie");
if (it != end(names)) {
   *it = "***";
}
```

String to search

Replace the string

Find and Replace Strings

```
// Find and replace "Connie" with "***"
const auto it = find(begin(names), end(names), "Connie");
if (it != end(names)) {
  *it = "***";
}
// Find and replace "C64" with "**"
const auto it = find(begin(names), end(names), "C64");
if (it != end(names)) {
  *it = "**";
}
```

Find and Replace Strings

```
// Find and replace "Connie" with "***"
const auto it = find(begin(names), end(names), "Connie");
if (it != end(names)) {
  *it = "***";
}
// Find and replace "C64" with "**"
const auto it = find(begin(names), end(names), "C64");
if (it != end(names)) {
  *it = "**";
}
```

Two variables with the *same* name

Rename Iterator Variables

```
// Find and replace "Connie" with "***"
const auto it = find(begin(names), end(names), "Connie");
if (it != end(names)) {
  *it
}

// Find and  eplace "C64" with "**"
const auto it = find(begin(names), end(names), "C64");
if (it != end(names)) {
  *it = "**";
}
```

Use a *different* name,
e.g.: it2

Introduce New Scopes

```
{

  // Find and replace "Connie" with "***"
  const auto it = find(begin(names), end(names), "Connie");
  if (it != end(names)) {
     *it = "***";
  }

}
```

```
for (int i = 0; i < n; i++) {
   ...
}
```

i : *local* to the *for* loop

C++17 Variable Declarations in *if* Statements

Analogy with *for* loop index

```
for (int i = 0; i < n; i++) {
   ...
}
for (int i = 0; i < n; i++) {
   ...
}
```

C++17 Variable Declarations in *if* Statements

Analogy with *for* loop index

```cpp
// Find and replace "Connie" with "***"
if (const auto it = find(...); it != end(names)) {
  *it = "***";
}
```

C++17 Variable Declarations in *if* Statements

```cpp
// Find and replace "Connie" with "***"
if (const auto it = find(...); it != end(names)) {
  *it = "***";
}
```

C++17 Variable Declarations in *if* Statements

```
// Find and replace "Connie" with "***"
if (const auto it = find(...); it != end(names)) {
  *it = "***";
}
```

C++17 Variable Declarations in *if* Statements

```
{
  // Find and replace "Connie" with "***"
  const auto it = find(begin(names), end(names), "Connie");
  if (it != end(names)) {
    *it = "***";
  }
}
```

C++17 Variable Declarations in *if* Statements

Equivalent code with new embracing scope

```
if (const auto it = find(...); it != end(names)) {
  *it = "***";
} else {
  // Not found ...
}
```

it available here

C++17 Variable Declarations in *if* Statements

Variable declaration

```
switch (auto val = GetSomeValue(); expression for switch) {
  Various cases…
}
```

C++17 Variable Declarations in *switch*

```
if (condition) {
   // Executed if condition is true
} else {
   // Executed if condition is false
}
```

Evaluated at *run-time*

if reached by control flow

C++17 if constexpr

From ordinary *if*...

```
if constexpr (condition) {
  // Executed if condition is true
} else {
  // Executed if condition is false
}
```

C++17 if constexpr

```
if constexpr (condition) {
   // Executed if condition is true
} else {
   // Executed if condition is false
}
```

Compile-time if
comes in handy
in C++ *template* code

C++17 if constexpr

Compile-time *if*

Using if constexpr with Template Code

```cpp
template <typename T>
auto DoSomething(T const& value) {

        ● ● ●

}
```

Using if constexpr with Template Code

```
template <typename T>
auto DoSomething(T const& value) {
  if constexpr (T is an int) {
    // Do something with integers…
  }



}
```

Using if constexpr with Template Code

```
template <typename T>
auto DoSomething(T const& value) {
  if constexpr (T is an int) {
    // Do something with integers…
  } else {
    // Do something else…
  }
}
```

Using if constexpr with Template Code

```
template <typename T>
auto DoSomething(T const& value) {
  if constexpr (T is an int) {
    // Do something with integers…
  } else {
    // Do something else…
  }
}
```

Condition evaluated at *compile-time*

Using if constexpr with Template Code

```
template <typename T>
auto DoSomething(T const& value) {
  if constexpr (T is an int) {
    // Do something with integers…
  } else {
    // Do something else…
  }
}
```

The «true» block is compiled

Using if constexpr with Template Code

```
template <typename T>
auto DoSomething(T const& value) {
  if constexpr (T is an int) {
    // Do something with integers…
  } else {
    // Do something else…
  }
}
```

C++ compiler
ignores this block

```
auto [var1, var2, …] = GetSomeData();
```

C++17 Structured Bindings

Single-statement multiple-variable-declarations

from pair/tuple/struct

Building an Italian-to-English Dictionary

string string

$\downarrow$ $\downarrow$

std::*map*<Key, Value>

Italian → English

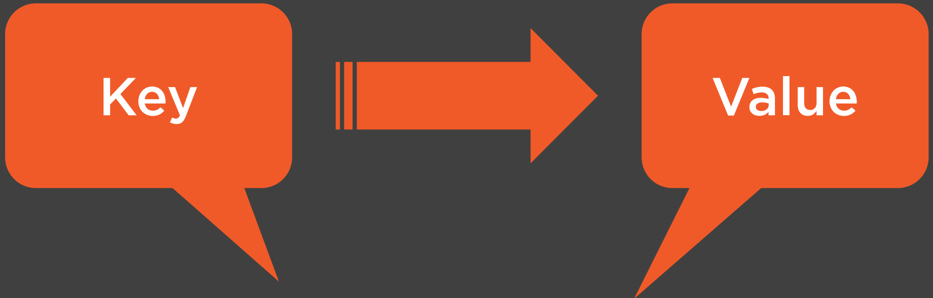
Italian-to-English Dictionary

```
map<string, string> italianDictionary{
   {“casa”,  “home”},
   {“gatto”, “cat”},
   {“pasta”, “pasta”}
   …
};

auto result = italianDictionary.insert({“sedia”, “chair”});
```

Italian-to-English Dictionary

```
map<string, string> italianDictionary{
   {“casa”,  “home”},
   {“gatto”, “cat”},
   {“pasta”, “pasta”}
   …
};
```



```
auto result = italianDictionary.insert({“sedia”, “chair”});
```

Italian-to-English Dictionary

```
map<string, string> italianDictionary{
   {“casa”,  “home”},
   {“gatto”, “cat”},
   {“pasta”, “pasta”}
   …
};

auto result = italianDictionary.insert({“sedia”, “chair”});
```

std::pair

first : Iterator
second : Boolean

Italian-to-English Dictionary

```
map<string, string> italianDictionary{ … };
```

Key *already* in map?

```
auto result = italianDictionary.insert({“sedia”, “chair”});
```

Case 1: Inserting Element with **New** Key

```
map<string, string> italianDictionary{ … };
```

result pair

first: Iterator → new item

second: *true* (insertion OK)

```
auto result = italianDictionary.insert({“sedia”, “chair”});
```

Case 2: Key **Already** in Map

```
map<string, string> italianDictionary{ … };
```

result pair

first: Iterator -> existing key item

second: *false (insertion failed)*

```
auto result = italianDictionary.insert({“sedia”, “chair”});
```

Checking Insertion Result pair

```
map<string, string> italianDictionary{ … };
auto result = italianDictionary.insert({“sedia”, “chair”});

if (result.second == true) {
  // Insertion OK…
} else {
  // Use result.first to locate the existing item…
}
```

Clearer than *first* and *second*

```
auto [position, success] = italianDictionary.insert(
                                {“sedia”, “chair”});
```

Simpler Code with C++17 Structured Bindings

Works also with
tuples and
custom structures

```
auto [position, success] = italianDictionary.insert(
                                      {“sedia”, “chair”});
```

Simpler Code with C++17 Structured Bindings

Summary

Nested namespaces

Variable declarations in *if* and *switch*

if constexpr

Structured bindings

Thank You!