What’s new in C++ 17 Language and Libraries?

PAVEL YOSIFOVICH

ZODIACON@LIVE.COM

@ZODIACON

About Me

Developer, trainer, author, speaker

Author

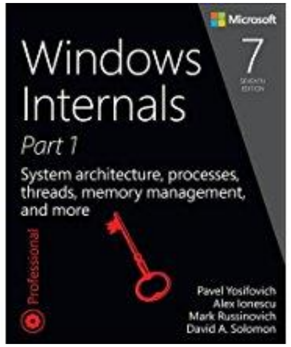
- Windows Internals 7th edition Part 1 (2017)
- WPF 4.5 Cookbook (2012)
- Mastering Windows 8 C++ App Development (2013)

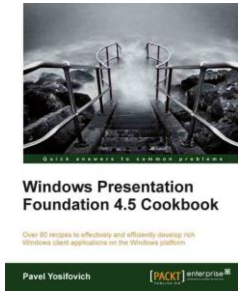
Pluralsight Author (www.pluralsight.com)

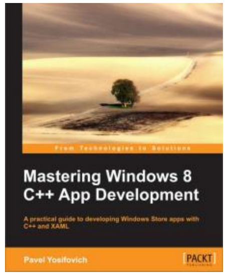
Microsoft MVP

Blog: http://blogs.Microsoft.co.il/pavely

Open source projects on GitHub (http://github.com/zodiacon)







Agenda

Introduction

Language Features

Library Features

Q & A

Introduction

Since 2011, C++ standards have been making steady marches every 3 years

C++ 17 is the latest approved C++ standard

C++ 20 is already in the works

This session is about new features in C++17 language and libraries

Assumption: you feel relatively comfortable with C++ 11/14

Nested Namespace Definitions

Before C++17

```
namespace A {
   namespace B {
      namespace C {
         ...
      }
   }
}
```

C++17

```
namespace A::B::C {
   ...
}
```

Structural Decomposition

```
map<string, string> capitals {
    { "England", "London" },
    { "France", "Paris" },
    { "Israel", "Jerusalem" },
    { "United States", "Washington D.C." },
    { "Spain", "Madrid" }
};
```

Before C++17

```
for (const auto& pair : capitals)
    cout << pair.first.c_str() << ": "
         << pair.second.c_str() << endl;
```

C++17

```
for (const auto& [country, capital] : capitals)
    cout << country.c_str() << ": "
        << capital.c_str() << endl;
```

Structural Decomposition

Works for any POD

- `std::pair`, `std::tuple`
- Any other structure
  - No static data members

All members must be provided

- Must come from the same class (not a base class)

Initialization Statements in `if/switch`

Before C++17

```cpp
auto it = capitals.find("Israel");
if (it != capitals.end()) {
    cout << it->second.c_str() << endl;
}
else {
    ...
}

auto it2 = someMap.find(...);
if (it2 != someMap.end()) {
   ...
}
```

C++17

```cpp
if (auto it = capitals.find("Israel");
    it != capitals.end()) {
    cout << it->second.c_str() << endl;
}
else {
    // “it” is still in scope
}

// but not here

auto it = someMap.find(...);
```

Template Arguments Class Template Deduction

Complements function template argument deduction

Before C++ 17

```
pair<int, string> p1(2, string("hello"));

// or

auto p2 = make_pair(2, "hello");
```

C++ 17

```
pair p1(2, string("hello"));

// or

auto p2 = pair(2, "hello");
```

Supported in Visual Studio 15.7 Preview

Inline Variables

Normally, static member variables must be explicitly defined in a CPP file

- Otherwise “Unresolved external” linker error reported

C++ 17

- Use the `inline` keyword to initialize the static variable in the header file only
- Useful for templated types implemented entirely in the header file

```
struct Dummy {
    static std::string _greeting;
};

inline std::string Dummy::_greeting = "Greetings, earthlings!";
```

Supported in Visual Studio 15.7 Preview

Fold Expressions

Non-recursive way to work with variadic templates

Before C++ 17

```
auto SumCpp11() {
    return 0;
}

template<typename T1, typename... T>
auto SumCpp11(T1 s, T... ts) {
    return s + SumCpp11(ts...);
}
```

C++ 17

```
template<typename... Args>
auto SumCpp17(Args... args) {
    return (args + ... + 0);
}

// or

template<typename... Args>
auto SumCpp17Alt(Args... args) {
    return (args + ...);
}
```

Supported in Visual Studio 15.7 Preview

Lambdas Enhancements

constexpr lambdas

- Compile-time lambdas with constexpr

```
constexpr auto add = [](auto x, auto y) {
    return x + y;
};

static_assert(add(3, 4) == 7);
```

- Lambdas may capture this by value
  - [this] – capture by reference
  - [*this] – capture by value

if constexpr

Conditionally compile code base on constexpr expressions

```
template <typename T>
constexpr bool IsIntegral() {
    if constexpr (std::is_integral<T>()) {
        return true;
    }
    return false;
}
```

```
template <typename T>
constexpr auto GetValue(const T& t) {
    if constexpr (std::is_pointer_v<T>)
        return *t;
    else
        return t;
}
```

```
static_assert(IsIntegral<int>() == true);
static_assert(IsIntegral<char>() == true);
static_assert(IsIntegral<double>() == false);
struct S {};
static_assert(IsIntegral<S>() == false);
```

```
int x = 17;
auto v1 = GetValue(x);
auto v2 = GetValue(&x);
```

New Standard Attributes

`[[fallthrough]]`

- A `case` in a `switch` may fall through

`[[maybe_unused]]`

- Indicates a variable may be unused

`[[nodiscard]]`

- Return value from a function should not be discarded

Library Features

`std::variant`

`std::optional`

`std::any`

`std::string_view`

`std::invoke` and `std::apply`

Splicing for maps and sets

Removed Features

`auto_ptr` removed

- Copy operation would actually do a move
- Possibly confusing
- Replacement is `unique_ptr` (C++ 11)
  - Copy operation does not compile

Trigraphs removed

- What the hell is a trigraph?

std::variant

Type-safe union

- “holds” a single data item at a time

```
#include <variant>
```

```
std::variant<int, double> v { 12 };

auto v1 = std::get<int>(v);  // v1 == 12
auto v2 = std::get<0>(v);    // v2 == 12

v = 12.0;
auto v3 = std::get<double>(v);   // v3 == 12.0
auto v4 = std::get<1>(v);        // v4 == 12.0

auto v5 = std::get<0>(v);        // exception
```

std::optional

Holds a value or no value at all

- Similar to C# nullable types ☺

```
#include <optional>
```

```
std::optional<double> DoCalculation(double value) {
    if (value < 0)
        return {};

    return ::sqrt(value);
}
```

```
double x = -12;
auto result = DoCalculation(x);
if (!result.has_value())
cout << "no value!" << endl;
else
cout << "result: " << result.value();

auto result2 = DoCalculation(x).value_or(-1);
```

std::any

Holder of a single value of any type

```
#include <any>
```

```
any x1{ 5 };
bool h1 = x1.has_value();            // true
cout << any_cast<int>(x1) << endl;
cout << x1.type().name() << endl;    // "int"

any x2{ string("zebra") };
cout << any_cast<string&>(x2).c_str() << endl;// "zebra"
cout << ((string&)x2).c_str() << endl;   // "zebra"
```

std::string_view

Non-owning string reference

- Prevents unnecessary copying
- Cannot modify the string it references
- wstring_view exists as well

```
std::string str{ "   some long string" };
std::string_view sv{ str };
sv.remove_prefix(sv.find_first_not_of(" "));

cout << str.c_str() << endl;     // "   some long string"
cout << sv.data() << endl;       // "some long string"
```

std::invoke and std::apply

std::invoke invokes any callable object

- E.g. std::function, std::bind, functor, function pointer, pointer to member, …
  - In pointer-to-member first argument is the this pointer

std::apply is similar but arguments are provided with std::tuple

```
#include <functional>
```

```
auto add = [](auto x, auto y) {
    return x + y;
};
auto add3 = [](auto x, auto y, auto z) {
    return x + y + z;
};

auto sum1 = std::invoke(add, 3, 4);                 // == 7
auto sum2 = std::apply(add, std::make_tuple(3, 4)); // == 7
auto sum3 = std::invoke(add3, 3, 4, 5.2);           // == 12.2
```

Parallel STL

Parallel execution of STL algorithms based on execution policy

```
#include <execution>
```

```
std::vector<int> vec = { 3, 2, 1, 4, 5, 6, 10, 8, 9, 4 };

std::sort(vec.begin(), vec.end());                            // sequential as ever
std::sort(std::execution::seq, vec.begin(), vec.end());       // sequential
std::sort(std::execution::par, vec.begin(), vec.end());       // parallel
std::sort(std::execution::par_unseq, vec.begin(), vec.end()); // parallel and vectorized
```

Splicing for maps and sets

Moving nodes and merging containers without the overhead of expensive copies or heap allocations/deallocations

```
std::map<int, string> src{ { 1, "one" },{ 2, "two" },{ 3, "zebra" } };
std::map<int, string> dst{ { 3, "three" } };
dst.insert(src.extract(src.find(1)));    // Cheap remove and insert of { 1, "one" }
dst.insert(src.extract(2));          // Cheap remove and insert of { 2, "two" }
```

```
// inserting an entire set

std::set<int> src{ 1, 3, 5 };
std::set<int> dst{ 2, 4, 5 };
dst.merge(src);
```

?
Q
&
A
!

Thank you