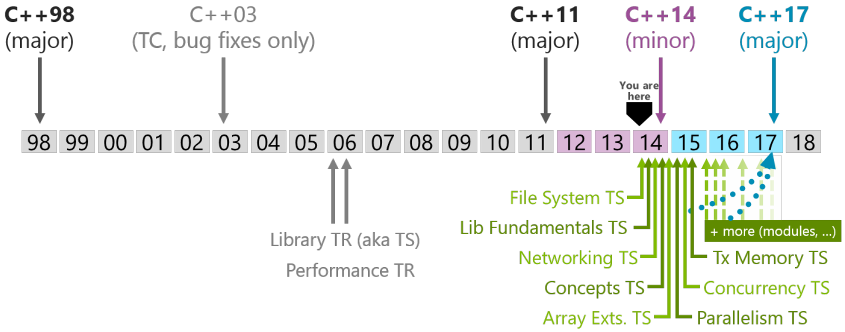
Co nowego w C++14

Piotr Wygocki
paal.mimuw.edu.pl

November 5, 2014

timeline


C++98
(major)
C++03
(TC, bug fixes only)
C++11
(major)
C++14
(minor)
C++17
(major)
You are here
98 99 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18
Library TR (aka TS)
Performance TR
File System TS
Lib Fundamentals TS
Networking TS
Concepts TS
Array Exts. TS
+ more (modules, ...)
Tx Memory TS
Concurrency TS
Parallelism TS

Plan

New language features

- Function return type deduction
- Alternate type deduction on declaration
- Relaxed constexpr restrictions
- Variable templates
- Aggregate member initialization
- Binary literals
- Digit separators
- Generic lambdas
- Lambda captures expressions

Czego NIE bedzie

New standard library features

- Shared mutexes and locking
- Heterogeneous lookup in associative containers
- Standard user-defined literals
- Tuple addressing via type
- Smaller library features

Function return type deduction

```
template <class T, class W>
??? sum(T t, W w) {
    return t + w;
}
```

Function return type deduction, rozwiazanie z C++11

```
template <class T, class W>
auto sum(T t, W w) -> decltype(t + w) {
    return t + w;
}
```

Function return type deduction, rozwiazanie z C++11

```
template <class T, class W>
auto sum(T t, W w) -> decltype(t + w) {
    return t + w;
}
```

Copy paste!

Function return type deduction, rozwiazanie z C++14

```
template <class T, class W>
auto sum(T t, W w) {
    return t + w;
}
```

Dużo lepiej!

Function return type deduction, rozwiazanie z C++14

```
template <class T, class W>
auto sum(T t, W w) {
    return t + w;
}
```

Dużo lepiej! (ale mogło by być jeszcze ładniej)

Function return type deduction, rozwiazanie z C++17

```
auto sum(auto t, auto w) {
    return t + w;
}
```

Zaimplementowane w g++-4.9

```cpp
template <typename T>
auto sum(auto t, auto w, int k, T tt) {
    return t + w;
}
```

Function return type deduction, rozwiazanie z C++48

```
sum(t,  w) {
    return t + w;
}
```

Funkcja z wieloma returnami

Funkcja z wieloma returnami

```
auto Correct(int i) {
    if (i == 1)
    return i; // return type deduced as int
    else
    return Correct(i-1)+i;// ok to call it now
}
```

Funkcja z wieloma returnami

```
auto Correct(int i) {
    if (i == 1)
    return i; // return type deduced as int
    else
    return Correct(i-1)+i;// ok to call it now
}
```

```
auto Wrong(int i) {
    if (i != 1)
    return Wrong(i-1)+i;//Too soon to call this
    // No prior return statement.
    else
    return i;   // return type deduced as int
}
```

Alternate type deduction on declaration

```
std::map<int, std::pair<double, float>> map
    = f(1, 2, "bu bu bu");
```

Alternate type deduction on declaration

```
std::map<int, std::pair<double, float>> map
    = f(1, 2, "bu bu bu");
```

```
auto map = f(1, 2, "bu bu bu");
//Kompilator sam domysli sie typu zmiennej map
```

Alternate type deduction on declaration

```
std::map<int, std::pair<double, float>> map
    = f(1, 2, "bu bu bu");
```

```
auto map = f(1, 2, "bu bu bu");
//Kompilator sam domysli sie typu zmiennej map
```

A co jeśli f zwraca referencje?

Alternate type deduction on declaration

```
decltype(f(1, 2, "bu bu bu")) map
    = f(1, 2, "bu bu bu");
```

Alternate type deduction on declaration, rozwiazanie w C++14

```
decltype(auto) map = f(1, 2, "bu bu bu");
```

Alternate type deduction on declaration in function return type deduction

```
decltype(auto) f()  {
    return some_reference;
}
```

Relaxed constexpr restrictions

```
constexpr int factorial(int n)
{
    return n <= 1 ? 1 : (n * factorial(n-1));
}
```

Relaxed constexpr restrictions - niedozwolone w C++11

```
constexpr int factorial_iterative(int n)
{
  int ret = 1;
  while(n > 0) {
          ret = ret * (n--);
  }
  return ret;
}
```

Relaxed constexpr restrictions

- Any declarations except:
  - static or thread_local variables.
  - variable declarations without initializers.
  - goto statements.
- The conditional branching statements if and switch.
- All looping statements, including range-based for.
- Expressions may change the value of an object

Variable templates

Variable templates

```
template<typename T>
constexpr T pi = T(3.1415926535897932385);
```

Variable templates

```
template<typename T>
constexpr T pi = T(3.1415926535897932385);
```

```
template <typename T>
T area_of_circle_with_radius(T r) {
    return pi<T> * r * r;
}
```

Aggregate member initialization

```
struct S {
    int a;
    const char* b;
    int c;
    int d = b[a];
};
```

Aggregate member initialization

```
struct S {
    int a;
    const char* b;
    int c;
    int d = b[a];
};
```

```
S ss = { 1, "asdf" };
```

Aggregate member initialization

```
struct X { int i, j, k = 42; };
X a[] = { 1, 2, 3, 4, 5, 6 };
X b[2] = { { 1, 2, 3 }, { 4, 5, 6 } };
```

Binary literals

```
int a = 0xf;
```

Binary literals

```
int a = 0xf;
```

```
int b = 0b1111;
```

Digit separators

```
auto integer_literal = 1'000'000;
auto floating_point_literal = 0.000'015'3;
auto binary_literal = 0b0100'1100'0110;
auto silly_example = 1'0'0'000'00;
```

Generic lambdas

```
struct unnamed_lambda
{
    auto operator()(int x, int y) const
        {return x + y;}
};
auto lambda = unnamed_lambda{};
```

```
auto lambda = [](int x, int y) { return x + y; };
```

Generic lambdas

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

```
struct unnamed_lambda
{
   template<typename T, typename U>
   auto operator()(T x, U y) const {return x + y;}
};
auto lambda = unnamed_lambda{};
```

Generic lambdas

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

```
struct unnamed_lambda
{
      auto operator()(auto x, auto y) const
            {return x + y;}
};
auto lambda = unnamed_lambda{};
```

Generic lambdas, przyklad

```
std::vector<int> v = {2, -1, 3};
abs_sort(v);
```

Generic lambdas, przyklad

```
std::vector<int> v = {2, -1, 3};
abs_sort(v);
```

```
void abs_sort(auto & range) {
    std::sort(std::begin(range), std::end(range),
            [](auto x, auto y){
                return std::abs(x) < std::abs(y);
            });
}
```

Generic lambdas, przylad range

```
void abs_sort(auto & range) {
      boost::sort(range,
            [](auto x, auto y){
                return std::abs(x) < std::abs(y);
            });
}
```

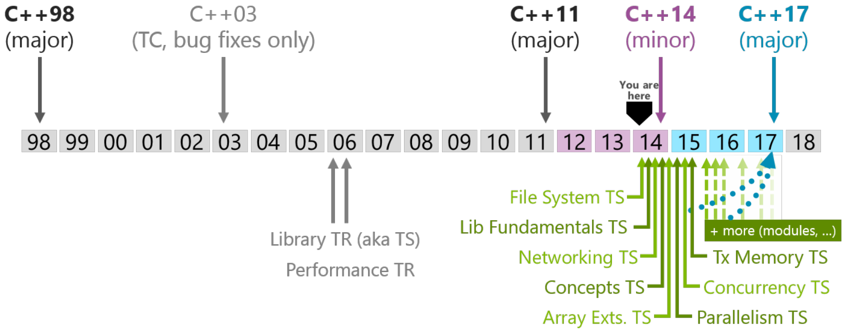
Lambda captures expressions

```
auto lambda = [=] {return value;};
auto lambda = [&] {return value;};
```

Lambda captures expressions

```
auto lambda = [=] {return value;};
auto lambda = [&] {return value;};
```

```
auto lambda = [value = 1] {return value;};
auto lambda = [value = std::move(some_value)] {
                return value;};
```



C++98
(major)
C++03
(TC, bug fixes only)
C++11
(major)
C++14
(minor)
C++17
(major)
You are here
98 99 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18
Library TR (aka TS)
Performance TR
File System TS
Lib Fundamentals TS
Networking TS
Concepts TS
Array Exts. TS
+ more (modules, ...)
Tx Memory TS
Concurrency TS
Parallelism TS

Jak to odpalic?

- g++ –std=c++1y
- clang++ –std=c++1y
- MSVC - nic nie trzeba

Jak to odpalic?

- g++ –std=c++1y
- clang++ –std=c++1y
- MSVC - nic nie trzeba (ale sa do tylu z implementacja)

Referencje

- isocpp.org/wiki/faq/cpp14-language
- en.wikipedia.org/wiki/C%2B%2B14
- opisy funkcjonalnosci np. open-std.org/JTC1/SC22/WG21/docs/papers/2013/n3638.html

Dziekuje za uwage!