C++14
Prostszy niż kiedykolwiek

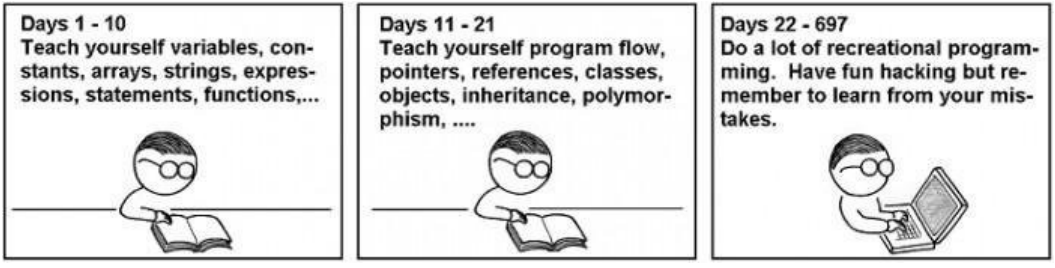
Bartosz ’BaSz’ Szurgot

http://www.baszerr.eu

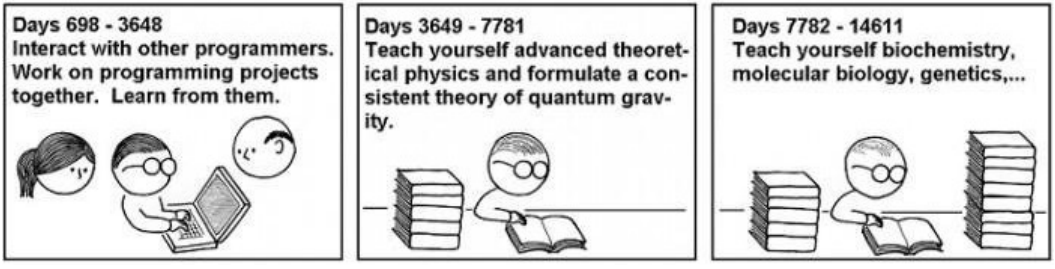
2015-03-25

C++ w 21 dni?

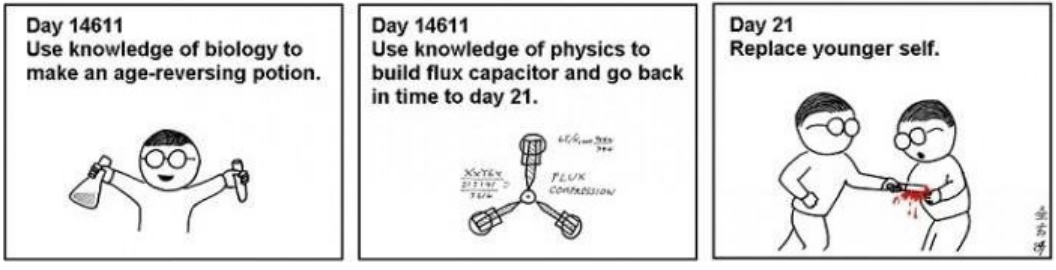
C++ w 21 dni?


Days 1 - 10
Teach yourself variables, constants, arrays, strings, expressions, statements, functions,...
Days 11 - 21
Teach yourself program flow, pointers, references, classes, objects, inheritance, polymorphism, ...
Days 22 - 697
Do a lot of recreational programming. Have fun hacking but remember to learn from your mistakes.

C++ w 21 dni?


Days 698 - 3648
Interact with other programmers. Work on programming projects together. Learn from them.
Days 3649 - 7781
Teach yourself advanced theoretical physics and formulate a consistent theory of quantum gravity.
Days 7782 - 14611
Teach yourself biochemistry, molecular biology, genetics,...

C++ w 21 dni?


Day 14611
Use knowledge of biology to make an age-reversing potion.
Day 14611
Use knowledge of physics to build flux capacitor and go back in time to day 21.
FLUX COMPRESSION
Day 21
Replace younger self.

Poprawny kod?

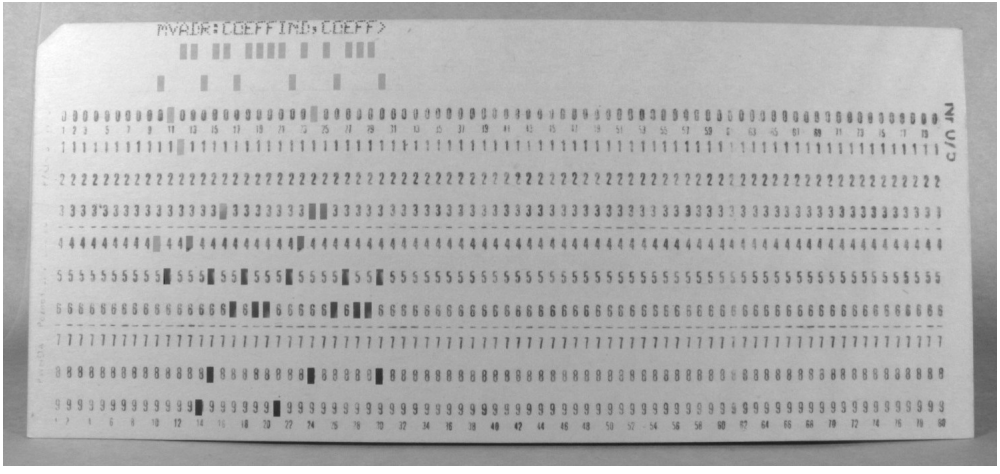
```
1  #include <string>
2  #include <vector>
3  #include <algorithm>
4
5  template<typename T>
6  struct Compare
7  {
8    bool operator()(T const& lhs, T const& rhs)
9    { return lhs.size() < rhs.size(); }
10 };
11 template<typename U, typename T>
12 U convert(T const& t)
13 {
14   U out( begin(t), end(t) );
15   std::sort( begin(out), end(out), Compare<typename T::value_type>() );
16   return out;
17 }
18 // ...
19 std::vector<char const*> in{ /* ... */ };
20 auto out = convert<std::vector<std::string>>(in);
```

Enterprise class template error message

Wniosek

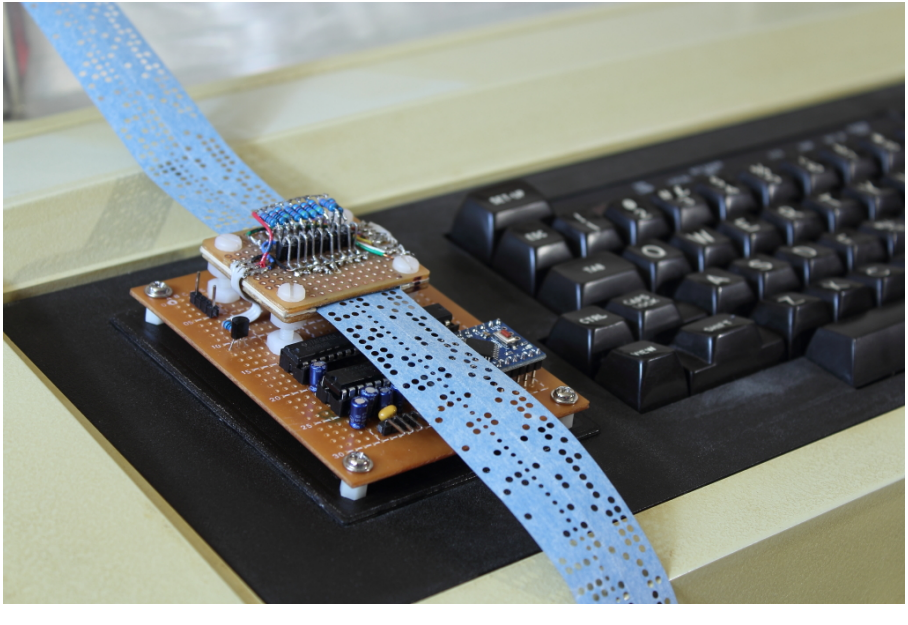
Czy C++ jest skomplikowany?

Zacznijmy od początku



W tym samym czasie. . .

Karty nadal żyją, tak mamo



Trend

- Chronologiczne:
  - Taśma – można (prawie) wszystko
  - asembler – minimum składni, zero kontroli
  - C – szybkość, przenośność, podstawowa kontrola błędów

Trend

- Chronologiczne:
  - Taśma – można (prawie) wszystko
  - asembler – minimum składni, zero kontroli
  - C – szybkość, przenośność, podstawowa kontrola błędów
  - C++98 – C z tanimi abstrakcjami i nowymi paradygmatami
  - C++11 – więcej tanich abstrakcji, poprawiona czytelność
  - C++14 – jeszcze więcej tanich abstrakcji, poprawy spójność biblioteki
  - C++17 – można się domyśleć. . . ;-)

Trend

- Chronologiczne:
  - Taśma – można (prawie) wszystko
  - asembler – minimum składni, zero kontroli
  - C – szybkość, przenośność, podstawowa kontrola błędów
  - C++98 – C z tanimi abstrakcjami i nowymi paradygmatami
  - C++11 – więcej tanich abstrakcji, poprawiona czytelność
  - C++14 – jeszcze więcej tanich abstrakcji, poprawy spójność biblioteki
  - C++17 – można się domyśleć. . . ;-)
- Nowe języki:
  - Więcej abstrakcji
  - . . .
  - Więcej ograniczeń
  - Łatwiejsze programowanie

Abstrakcja a wydajność

- C++ – programista decyduje
  - Statyczny vs. dynamiczny polimorfizm
  - Funkcje *inline*

Abstrakcja a wydajność

- C++ – programista decyduje
  - Statyczny vs. dynamiczny polimorfizm
  - Funkcje *inline*
- Java – ustalone odgórnie (JVM)
  - Wszystkie metody wirtualne
  - Automatyczne zarządzanie pamięcią

Abstrakcja a wydajność

- C++ – programista decyduje
  - Statyczny vs. dynamiczny polimorfizm
  - Funkcje *inline*
- Java – ustalone odgórnie (JVM)
  - Wszystkie metody wirtualne
  - Automatyczne zarządzanie pamięcią
- Języki skryptowe
  - Interpretowane w trakcie wykonania
  - Jaka wydajność? ;)

Abstrakcja dla wydajności

- Jak wpisać zero do rejestru CPU?

Abstrakcja dla wydajności

- Jak wpisać zero do rejestru CPU?
  - Zależy od architektury!
  - x86: *MOV* vs. *XOR*
  - Trwają tyle samo...

Abstrakcja dla wydajności

- Jak wpisać zero do rejestru CPU?
  - Zależy od architektury!
  - x86: *MOV* vs. *XOR*
  - Trwają tyle samo...
- Intel x86 developer's guide – 1500 stron
- Intel x86 optimization manual – 650 stron
  - *XOR* – przerywa łańcuch zależności
  - *XOR* – krótsza instrukcja
  - Kompilator wiedział...

Abstrakcja dla wydajności

- Jak wpisać zero do rejestru CPU?
  - Zależy od architektury!
  - x86: *MOV* vs. *XOR*
  - Trwają tyle samo...
- Intel x86 developer's guide – 1500 stron
- Intel x86 optimization manual – 650 stron
  - *XOR* – przerywa łańcuch zależności
  - *XOR* – krótsza instrukcja
  - Kompilator wiedział...
- Kompilator (często) wie lepiej
  - Przypadki brzegowe
  - Propagacja stałych
  - Automatyczna wektoryzacja
  - Relacje między instrukcjami
  - ...


SAY LOW-LEVEL
ONE MORE GODDAMN TIME
quickmeme.com

Parasol
Smarter computing
Texas A&M University

C++11 Style – A Touch of Class

Bjarne Stroustrup
Texas A&M University
www.research.att.com/~bs

"C++11 feels like a new language."

— Bjarne Stroustrup

Bezpieczny printf?

- C++11 prostszy w użyciu
- Dodatkowe elementy upraszczają

Bezpieczny printf?

- C++11 prostszy w użyciu
- Dodatkowe elementy upraszczają
- *printf* da się zaimplementować bezpiecznie

```
1 void oldPrintf(const char *format, ...);
2
3 template<typename ...Args>
4 void newPrintf(std::string const& format, Args&&... args);
```

- Błąd w formacie – wyjątek
- A gdyby tak...

Printf - argumenty czasu kompilacji

- Format w czasie kompilacji?

```
1  template<typename ...Args>
2  void printfMk2(Args&&... args);
3
4  // ...
5
6  template<typename T>
7  void outputValue(T const& t)
8  {
9    printfMk2("your␣value␣is:␣", t);
10 }
```

- Co z różnym formatowaniem tych samych typów?

Printf a format

- Format w czasie kompilacji?

```
1  template<typename ...Args>
2  void printfMk2(Args&&... args);
3
4  template<typename T>
5  struct AsHexImpl;
6
7  template<typename T>
8  AsHexImpl<T> asHex(T const& t);
9
10 template<typename T>
11 void outputValue(T const& t)
12 {
13   printfMk2("or␣in␣hex:␣", asHex(t));
14 }
```

- Debugger? Dziękuję – kompilator starczy!

Mediana z wektora

- Poprawny czy nie?

```
1 std::vector<long> v;
2 // ... filling v ...
3 const unsigned n = v.size();
4 cout << "mediana:␣" << v[n/2] << endl;
```

Mediana z wektora

- Poprawny czy nie?

```
1 std::vector<long> v;
2 // ... filling v ...
3 const unsigned n = v.size();
4 cout << "mediana:␣" << v[n/2] << endl;
```

- Poprawnie to:

```
1 std::vector<long> v;
2 // ... filling v ...
3 const std::vector<long>::size_type n = v.size();    // <-- !!!
4 cout << "mediana:␣" << v[n/2] << endl;
```

Mediana z wektora

- Poprawny czy nie?

```
1 std::vector<long> v;
2 // ... filling v ...
3 const unsigned n = v.size();
4 cout << "mediana:␣" << v[n/2] << endl;
```

- Poprawnie to:

```
1 std::vector<long> v;
2 // ... filling v ...
3 const std::vector<long>::size_type n = v.size();    // <-- !!!
4 cout << "mediana:␣" << v[n/2] << endl;
```

- Albo prościej:

```
1 std::vector<long> v;
2 // ... filling v ...
3 const auto n = v.size();    // read as: "just use the right type" :-D
4 cout << "mediana:␣" << v[n/2] << endl;
```

Iteracja po mapie

- Poprawnie czy nie?

```
1 std::map<int, std::string> m;
2 // ... filling m ...
3 for(std::pair<int, std::string> const& elem: m)
4   cout << elem.first << "␣->␣" << elem.second << endl;
```

Iteracja po mapie

- Poprawnie czy nie?

```
1 std::map<int, std::string> m;
2 // ... filling m ...
3 for(std::pair<int, std::string> const& elem: m)
4   cout << elem.first << "␣->␣" << elem.second << endl;
```

- Poprawnie to:

```
1 std::map<int, std::string> m;
2 // ... filling m ...
3 for(std::pair<const int, std::string> const& elem: m)   // <-- !!!
4   cout << elem.first << "␣->␣" << elem.second << endl;
```

Iteracja po mapie

- Poprawnie czy nie?

```
std::map<int, std::string> m;
// ... filling m ...
for(std::pair<int, std::string> const& elem: m)
  cout << elem.first << "␣->␣" << elem.second << endl;
```

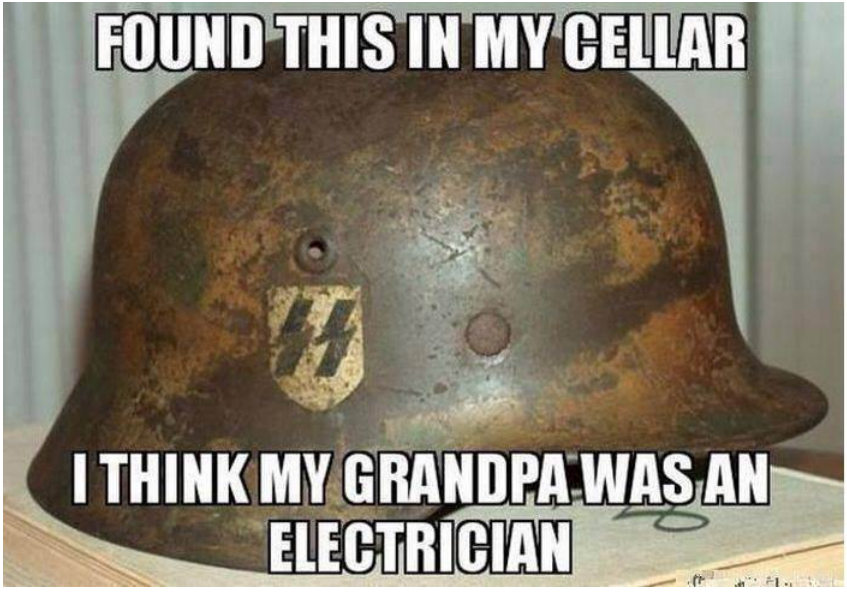
- Poprawnie to:

```
std::map<int, std::string> m;
// ... filling m ...
for(std::pair<const int, std::string> const& elem: m)   // <-- !!!
  cout << elem.first << "␣->␣" << elem.second << endl;
```

- Albo prościej:

```
std::map<int, std::string> m;
// ... filling m ...
for(auto const& elem: m)    // read as: "just use the right type" :-D
  cout << elem.first << "␣->␣" << elem.second << endl;
```

Pozory często mylą



Fibonacci czasu kompilacji - C++03

- Ikona metaprogramowania:

```
1  template<uint64_t N>
2  struct Fib
3  {
4    static const uint64_t value =
5      Fib<N-1>::value + Fib<N-2>::value;
6  };
7
8  template<>
9  struct Fib<1>
10 { static const uint64_t value = 1; };
11
12 template<>
13 struct Fib<0>
14 { static const uint64_t value = 0; };
```

- Jadalne dla początkujących? :-/

Fibonacci czasu kompilacji - C++03

- Ikona metaprogramowania:

```
1  template<uint64_t N>
2  struct Fib
3  {
4    static const uint64_t value =
5      Fib<N-1>::value + Fib<N-2>::value;
6  };
7
8  template<>
9  struct Fib<1>
10 { static const uint64_t value = 1; };
11
12 template<>
13 struct Fib<0>
14 { static const uint64_t value = 0; };
```

- Jadalne dla początkujących? :-/

Fibonacci czasu kompilacji - C++11

- Naturalna składnia
- Jedna linijka:

```
1 constexpr uint64_t fib(uint64_t n)
2 {
3   return n<2 ? n : fib(n-1) + fib(n-2);
4 }
```

- . . . ale trochę „ciasna”
- Musi być „jedna linijka”

Fibonacci czasu kompilacji - C++11

- Naturalna składnia
- Jedna linijka:

```
1 constexpr uint64_t fib(uint64_t n)
2 {
3   return n<2 ? n : fib(n-1) + fib(n-2);
4 }
```

- . . . ale trochę „ciasna”
- Musi być „jedna linijka”
- Bonus:
  - Działa w czasie kompilacji
  - Działa w czasie wykonania (!)

Fibonacci czasu kompilacji - C++14

- Naturalna składnia
- Naturalna implementacja

```
1 constexpr uint64_t fib(uint64_t n)
2 {
3   if(n<2)
4     return n;
5   return fib(n-1) + fib(n-2);
6 }
```

Fibonacci czasu kompilacji - C++14

- Naturalna składnia
- Naturalna implementacja

```
constexpr uint64_t fib(uint64_t n)
{
  if(n<2)
    return n;
  return fib(n-1) + fib(n-2);
}
```

- Albo prościej:

```
constexpr auto fib(uint64_t n)
{
  if(n<2)
    return n;
  return fib(n-1) + fib(n-2);
}
```

Jaka wartość zwracana? (C++03)

```
1 template<typename F, typename T>
2 ??? apply(F f, T const& t)
3 {
4   return f(t);
5 }
```

Jaka wartość zwracana? (C++03)

```
1 template<typename F, typename T>
2 ??? apply(F f, T const& t)
3 {
4   return f(t);
5 }
```

- Nie da się...
- Konwencje (*F::return_value*)?
- Co gdy przeciążone funkcje?

Jaka wartość zwracana? (C++11)

```
1 template<typename F, typename T>
2 auto apply(F f, T const& t) -> decltype(f(t))
3 {
4   return f(t);
5 }
```

- Da się! :D
- „Verbośna” składnia...
- Kopy-pasta kodu...

Jaka wartość zwracana? (C++14)

```
1 template<typename F, typename T>
2 auto apply(F f, T const& t)
3 {
4   return f(t);
5 }
```

- Da się – i to lepiej! :D
- Zwięzły kod
- Naturalna składnia
- (uwaga: nieco inne znaczenie *auto* / *decltype()*)

Mniej kodu == lepiej

Sortowanie napisów po długości (C++03)

```
1  namespace
2  {
3  bool cmp(std::string const& lhs, std::string const& rhs)
4  {
5    return lhs.size() < rhs.size();
6  }
7  }
8
9  void sortByLen(std::vector<std::string>& inOut)
10 {
11   std::sort(inOut.begin(), inOut.end(), cmp);
12 }
```

Sortowanie napisów po długości (C++03)

```
1  namespace
2  {
3  bool cmp(std::string const& lhs, std::string const& rhs)
4  {
5    return lhs.size() < rhs.size();
6  }
7  }
8
9  void sortByLen(std::vector<std::string>& inOut)
10 {
11   std::sort(inOut.begin(), inOut.end(), cmp);
12 }
```

- Nielokalna implementacja operatora...
- Parametr wyjściowy...

Sortowanie napisów po długości (C++11)

```
1 std::vector<std::string> sortByLen(std::vector<std::string> v)
2 {
3   auto cmp = [](std::string const& lhs, std::string const& rhs)
4              { return lhs.size() < rhs.size(); };
5   std::sort(begin(v), end(v), cmp);
6   return v;
7 }
```

Sortowanie napisów po długości (C++11)

```
1 std::vector<std::string> sortByLen(std::vector<std::string> v)
2 {
3   auto cmp = [](std::string const& lhs, std::string const& rhs)
4              { return lhs.size() < rhs.size(); };
5   std::sort(begin(v), end(v), cmp);
6   return v;
7 }
```

- Kod lokalny
- Krótsza implementacja
- Semantyka funkcji
- Można krócej?

Sortowanie napisów po długości (C++14)

```
1 std::vector<std::string> sortByLen(std::vector<std::string> v)
2 {
3   auto cmp = [](auto& lhs, auto& rhs) { return lhs.size() < rhs.size(); };
4   std::sort(begin(v), end(v), cmp);
5   return v;
6 }
```

Sortowanie napisów po długości (C++14)

```
1 std::vector<std::string> sortByLen(std::vector<std::string> v)
2 {
3   auto cmp = [](auto& lhs, auto& rhs) { return lhs.size() < rhs.size(); };
4   std::sort(begin(v), end(v), cmp);
5   return v;
6 }
```

- Generyczne lambdy – mniej kodu
- A gdyby tak...

Generyczne sortowanie napisów po długości (C++14)

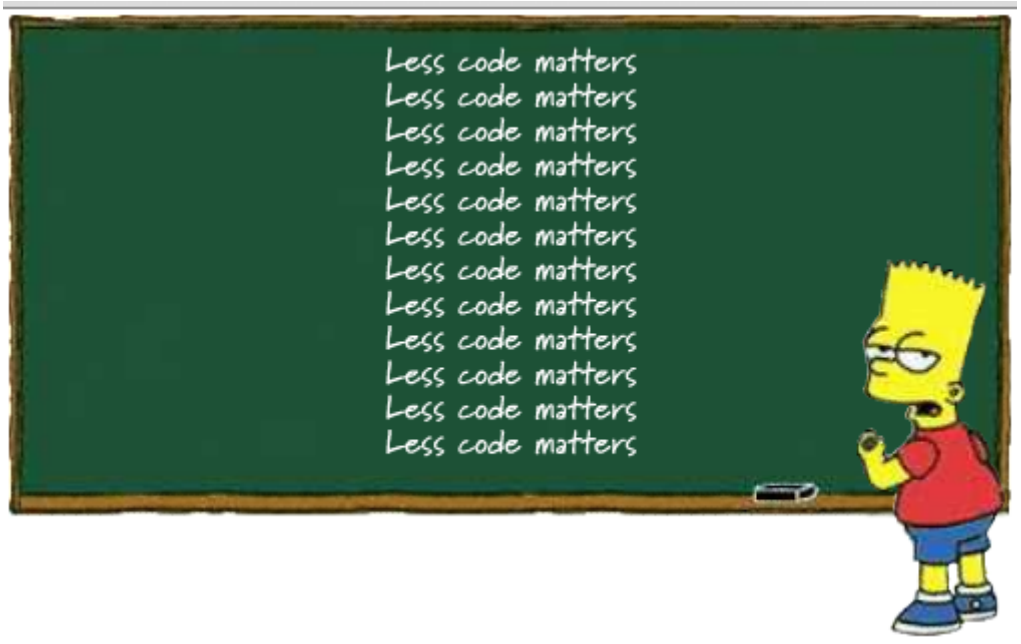
```
1 template<typename C>
2 auto sortByLen(C c)
3 {
4   auto cmp = [](auto& lhs, auto& rhs) { return lhs.size() < rhs.size(); };
5   using std::begin;
6   using std::end;
7   std::sort(begin(c), end(c), cmp);
8   return c;
9 }
```

Generyczne sortowanie napisów po długości (C++14)

```
template<typename C>
auto sortByLen(C c)
{
  auto cmp = [](auto& lhs, auto& rhs) { return lhs.size() < rhs.size(); };
  using std::begin;
  using std::end;
  std::sort(begin(c), end(c), cmp);
  return c;
}
```

- Bardzo ogólna implementacja
- Dedukcja typu zwracanego
- Generyczne lambdy i szablony
- Zadziała także dla tablic!
- Zwięzły kod

Zwięzłość jest zaletą



C++ coraz prostszy!

- Specyfikacja rośnie:
  - C++03: 786 stron
  - C++11: 1324 stron
  - C++14: 1374 stron

C++ coraz prostszy!

- Specyfikacja rośnie:
  - C++03: 786 stron
  - C++11: 1324 stron
  - C++14: 1374 stron
- Nowe elementy ułatwiają:
  - Funktory vs. (generyczne) lambdy
  - Metaprogramy vs. *constexpr*
  - Ręczne podawanie typów vs. *auto*
  - *typedef* vs. *using*
  - Inteligentne wskaźniki
  - Wątki (język i biblioteka)
  - . . .

C++ coraz prostszy!

- Specyfikacja rośnie:
  - C++03: 786 stron
  - C++11: 1324 stron
  - C++14: 1374 stron
- Nowe elementy ułatwiają:
  - Funktory vs. (generyczne) lambdy
  - Metaprogramy vs. *constexpr*
  - Ręczne podawanie typów vs. *auto*
  - *typedef* vs. *using*
  - Inteligentne wskaźniki
  - Wątki (język i biblioteka)
  - . . .
- „Nie płać za to czego nie używasz”
  - Lepsza abstrakcja
  - Ta sama wydajność

C++ coraz prostszy!

- Specyfikacja rośnie:
  - C++03: 786 stron
  - C++11: 1324 stron
  - C++14: 1374 stron
- Nowe elementy ułatwiają:
  - Funktory vs. (generyczne) lambdy
  - Metaprogramy vs. *constexpr*
  - Ręczne podawanie typów vs. *auto*
  - *typedef* vs. *using*
  - Inteligentne wskaźniki
  - Wątki (język i biblioteka)
  - . . .
- „Nie płać za to czego nie używasz”
  - Lepsza abstrakcja
  - Ta sama wydajność
- Uczmy C++14, z pominięciem C++03!
  - Użyteczność++
  - Skomplikowanie- -

Wykorzystane grafiki

- http://2.bp.blogspot.com/-96Pbx_x35eQ/U9nwnxjuGQI/AAAAAAAABQE/6QdsoOjOlCA/s1600/text-to-code-ratio.png
- http://i.imgur.com/Db2aqwL.jpg
- http://i11.photobucket.com/albums/a166/ballsandy/Computer%20related/CGS_0671.jpg
- http://indianapublicmedia.org/amomentofscience/files/2014/07/amos_14_152_mammoth-940x626.jpg
- http://static.fjcdn.com/pictures/Learn+c+in+21+days_7ee339_3181601.jpg
- http://techoctave.com/c7/static/say-low-level-one-more-goddamn-time.jpg
- http://www.coderscity.pl/download.php?id=4819
- http://www.mscarita.com/images/products/warning_sign_radiation_hazard.jpg
- http://www.techsling.com/wp-content/uploads/2014/02/slow-computer.jpg
- https://maurits.files.wordpress.com/2011/06/dry.png
- https://toldyah.files.wordpress.com/2014/11/computer-says-no.jpg

Compiler-friendly

Pytania?