```
In file included from /usr/include/c++/6/bits/stl_tree.h:65:0,
                 from /usr/include/c++/6/set:60,
                 from c2.cpp:3:
/usr/include/c++/6/bits/stl_function.h: In instantiation of ‘constexpr bool
std::less<_Tp>::operator()(const _Tp&, const _Tp&) const [with _Tp = main()::X]’:
/usr/include/c++/6/bits/stl_tree.h:1836:11:   required from ‘std::pair<std::_Rb_tree_node_base*,
std::_Rb_tree_node_base*> std::_Rb_tree<_Key, _Val, _KeyOfValue, _Compare,
_Alloc>::_M_get_insert_unique_pos(const key_type&) [with _Key = main()::X; _Val = main()::X;
_KeyOfValue = std::_Identity<main()::X>; _Compare = std::less<main()::X>; _Alloc =
std::allocator<main()::X>; std::_Rb_tree<_Key, _Val, _KeyOfValue, _Compare, _Alloc>::key_type =
main()::X]’
/usr/include/c++/6/bits/stl_tree.h:1889:28:   required from ‘std::pair<std::_Rb_tree_iterator<_Val>,
bool> std::_Rb_tree<_Key, _Val, _KeyOfValue, _Compare, _Alloc>::_M_insert_unique(_Arg&&) [with _Arg =
main()::X; _Key = main()::X; _Val = main()::X; _KeyOfValue = std::_Identity<main()::X>; _Compare =
std::less<main()::X>; _Alloc = std::allocator<main()::X>]’
/usr/include/c++/6/bits/stl_set.h:492:40:   required from ‘std::pair<typename std::_Rb_tree<_Key, _Key,
std::_Identity<_Key>, _Compare, typename
__gnu_cxx::__alloc_traits<_Alloc>::rebind<_Key>::other>::const_iterator, bool> std::set<_Key, _Compare,
_Alloc>::insert(std::set<_Key, _Compare, _Alloc>::value_type&&) [with _Key = main()::X; _Compare =
std::less<main()::X>; _Alloc = std::allocator<main()::X>; typename std::_Rb_tree<_Key, _Key,
std::_Identity<_Key>, _Compare, typename
__gnu_cxx::__alloc_traits<_Alloc>::rebind<_Key>::other>::const_iterator =
std::_Rb_tree_const_iterator<main()::X>; std::set<_Key, _Compare, _Alloc>::value_type = main()::X]’
c2.cpp:8:15:   required from here
/usr/include/c++/6/bits/stl_function.h:386:20: error: no match for ‘operator<’ (operand types are
‘const main()::X’ and ‘const main()::X’)
     { return __x < __y; }
              ~~~~^~~~
In file included from /usr/include/c++/6/bits/stl_algobase.h:64:0,
                 from /usr/include/c++/6/bits/stl_tree.h:63,
                 from /usr/include/c++/6/set:60,
                 from c2.cpp:3:
/usr/include/c++/6/bits/stl_pair.h:369:5: note: candidate: template<class _T1, class _T2> constexpr
bool std::operator<(const std::pair<_T1, _T2>&, const std::pair<_T1, _T2>&)
     operator<(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)
```

Co się stało?

```
#include <set>

int main() {
  class X{};
  std::set<X> x;
  x.insert(X{});

  return 0;
}

$ g++-6 set.cpp -o set
```

Użyjmy konceptów!

```
$ g++-6 -fconcepts set.cpp -o set
```

```
In file included from /usr/include/c++/6/bits/stl_tree.h:65:0,
                 from /usr/include/c++/6/set:60,
                 from c2.cpp:3:
/usr/include/c++/6/bits/stl_function.h: In instantiation of ‘constexpr bool
std::less<_Tp>::operator()(const _Tp&, const _Tp&) const [with _Tp = main()::X]’:
/usr/include/c++/6/bits/stl_tree.h:1836:11:   required from ‘std::pair<std::_Rb_tree_node_base*,
std::_Rb_tree_node_base*> std::_Rb_tree<_Key, _Val, _KeyOfValue, _Compare,
_Alloc>::_M_get_insert_unique_pos(const key_type&) [with _Key = main()::X; _Val = main()::X;
_KeyOfValue = std::_Identity<main()::X>; _Compare = std::less<main()::X>; _Alloc =
std::allocator<main()::X>; std::_Rb_tree<_Key, _Val, _KeyOfValue, _Compare, _Alloc>::key_type =
main()::X]’
/usr/include/c++/6/bits/stl_tree.h:1889:28:   required from ‘std::pair<std::_Rb_tree_iterator<_Val>,
bool> std::_Rb_tree<_Key, _Val, _KeyOfValue, _Compare, _Alloc>::_M_insert_unique(_Arg&&) [with _Arg =
main()::X; _Key = main()::X; _Val = main()::X; _KeyOfValue = std::_Identity<main()::X>; _Compare =
std::less<main()::X>; _Alloc = std::allocator<main()::X>]’
/usr/include/c++/6/bits/stl_set.h:492:40:   required from ‘std::pair<typename std::_Rb_tree<_Key, _Key,
std::_Identity<_Key>, _Compare, typename
__gnu_cxx::__alloc_traits<_Alloc>::rebind<_Key>::other>::const_iterator, bool> std::set<_Key, _Compare,
_Alloc>::insert(std::set<_Key, _Compare, _Alloc>::value_type&&) [with _Key = main()::X; _Compare =
std::less<main()::X>; _Alloc = std::allocator<main()::X>; typename std::_Rb_tree<_Key, _Key,
std::_Identity<_Key>, _Compare, typename
__gnu_cxx::__alloc_traits<_Alloc>::rebind<_Key>::other>::const_iterator =
std::_Rb_tree_const_iterator<main()::X>; std::set<_Key, _Compare, _Alloc>::value_type = main()::X]’
c2.cpp:8:15:   required from here
/usr/include/c++/6/bits/stl_function.h:386:20: error: no match for ‘operator<’ (operand types are
‘const main()::X’ and ‘const main()::X’)
     { return __x < __y; }
              ~~~~^~~~
In file included from /usr/include/c++/6/bits/stl_algobase.h:64:0,
                 from /usr/include/c++/6/bits/stl_tree.h:63,
                 from /usr/include/c++/6/set:60,
                 from c2.cpp:3:
/usr/include/c++/6/bits/stl_pair.h:369:5: note: candidate: template<class _T1, class _T2> constexpr
bool std::operator<(const std::pair<_T1, _T2>&, const std::pair<_T1, _T2>&)
     operator<(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)
```

Porównanie

```
$ g++-6 set.cpp -o set 2>&1 | wc -l
132
```

Porównanie

```
$ g++-6 set.cpp -o set 2>&1 | wc -l
132
$ g++-6 -fconcepts set.cpp -o set 2>&1 | wc -l
132
```

Porównanie

```
$ g++-6 set.cpp -o set 2>&1 | wc -l
132
$ g++-6 -fconcepts set.cpp -o set 2>&1 | wc -l
132

$ g++-6 set.cpp -o set 2>&1 | md5sum
775c5ea8b8cd59fdebb94b4fad6bcc1c
$ g++-6 -fconcepts set.cpp -o set 2>&1 | md5sum
775c5ea8b8cd59fdebb94b4fad6bcc1c
```

Koncepty. Dlaczego to dobrze, że nie pojawią się w C++17?

Piotr Kozłowski

Warszawa, 15 lutego 2017

Motywacja

Lepsza diagnostyka kompilatora

Kod szablonowy jest delikatny

Autor kodu może mieć (niejawne) założenia co do argumentów

Motywacja

Założenia podane w dokumentacji

Motywacja

Założenia podane w dokumentacji

Kompilator nie zagląda do dokumentacji

Ramy czasowe

Pomysł: lata ‘90

Concepts: odrzucone w 2009

Concepts “Lite”: odrzucone w 2016

GCC6.1: Concepts “Lite” dostępne

clang: WIP

Concept

a principle or idea:

The concept of free speech is unknown to them.

It is very difficult to define the concept of beauty.

I failed to grasp the film's central concept.

Kleenbrite is a whole new concept in toothpaste!*

* *http://dictionary.cambridge.org/dictionary/english/concept*

Template	templejt
std::future	std::fjuczer
Concept	~~zasada, pojęcie,~~ koncept

Koncept

«szybko nasuwający się pomysł»

«pomysłowość, zmyślność»

daw. «dowcip, żart»*

* *http://sjp.pwn.pl/szukaj/koncept.html*

Kiedy koncepty pomogą

Błędy składniowe

> *“Czy to się skompiluje?”*

Kiedy koncepty nie pomogą

Błędy logiki

> *“Czy to zadziała zgodnie z oczekiwaniami?”*

Prosty koncept

```
template <typename T, typename U>
requires requires(T t, U u) { t == u; }
bool check(T && lhs, U && rhs) {
    return lhs == rhs;
}
```

Problem #1

```
template <typename T, typename U>
requires requires(T t, U u) { t == u; }
bool check(T && lhs, U && rhs) {
    return lhs == rhs;
}
```

Diagnostyka

```
template <typename T, typename U>
requires requires(T t, U u) { t == u; }
bool check(T && lhs, U && rhs) {
    return lhs == rhs;
}

class X{};

int main() {
    return check(X{}, X{});
}
```

Diagnostyka

```
c1.cpp: In function ‘int main()’:
c1.cpp:10:23: error: cannot call function ‘bool check(T&&, U&&) [with T = X; U = X]’
  return check(X{}, X{});
                       ^
c1.cpp:3:6: note:  constraints not satisfied
 bool check(T && lhs, U && rhs) {
      ^~~~~
c1.cpp:3:6: note:    with ‘X t’
c1.cpp:3:6: note:    with ‘X u’
c1.cpp:3:6: note: the required expression ‘(t == u)’ would be ill-formed
```

Diagnostyka - klasycznie

```
template <typename T, typename U>
// requires requires(T t, U u) { t == u; }
bool check(T && lhs, U && rhs) {
    return lhs == rhs;
}

class X{};

int main() {
    return check(X{}, X{});
}
```

Diagnostyka - klasycznie

```
c0.cpp: In instantiation of ‘bool check(T&&, U&&) [with T = X; U = X]’:
c0.cpp:9:23:   required from here
c0.cpp:3:16: error: no match for ‘operator==’ (operand types are ‘X’ and ‘X’)
     return lhs == rhs;
            ~~~~^~~~~~
```

Nazwane koncepty (funkcyjne)

```
template <typename T, typename U>
concept bool Equality_comparable() {
    return requires(T t, U u) {
        { t == u } -> bool;
    };
}

template <typename T, typename U>
requires Equality_comparable<T, U>()
bool check(T && lhs, U && rhs) {
    return lhs == rhs;
}
```

Diagnostyka

```
cf.cpp: In function ‘int main()’:
cf.cpp:17:23: error: cannot call function ‘bool check(T&&, U&&) [with T = X; U = X]’
  return check(X{}, X{});
                       ^
cf.cpp:10:6: note:  constraints not satisfied
 bool check(T && lhs, U && rhs) {
      ^~~~~
cf.cpp:2:14: note: within ‘template<class T, class U> concept bool
Equality_comparable() [with T = X; U = X]’
 concept bool Equality_comparable() {
              ^~~~~~~~~~~~~~~~~~~
cf.cpp:2:14: note:    with ‘X t’
cf.cpp:2:14: note:    with ‘X u’
cf.cpp:2:14: note: the required expression ‘(t == u)’ would be ill-formed
```

Problem #2: definicje konceptów

```
template <typename T, typename U>
concept bool Equality_comparable() {
    return requires(T t, U u) {
        { t == u } -> bool;
    }
}
```

Nazwane koncepty (zmienne)

```
template <typename T, typename U>
concept bool Equality_comparable =
    requires(T t, U u) {
        { t == u } -> bool;
    };


template <typename T, typename U>
requires Equality_comparable<T, U>
bool check(T && lhs, U && rhs) {
    return lhs == rhs;
}
```

Problem #3: dwie różne składnie

```
template <typename T, typename U>
concept bool Equality_comparable() {
    return requires(T t, U u) {
        { t == u } -> bool;
    };
}

template <typename T, typename U>
requires Equality_comparable<T, U>()
bool check(T && lhs, U && rhs) {
    return lhs == rhs;
}
```

Koncepty zmienne i funkcyjne - różnice

	Funkcyjne	Zmienne
Overloading	TAK	NIE

Koncepty zmienne i funkcyjne - różnice

```
template <typename T, typename U>
concept bool Equality_comparable() {
    return requires(T t, U u) {
        { t == u } -> bool;
    };
}

template <typename T>
concept bool Equality_comparable() {
    return requires(T t1, T t2) {
        { t1 == t2 } -> bool;
    };
}
```

```
template <typename T>
requires Equality_comparable<T>()
bool check(T && lhs, T && rhs) {
    return lhs == rhs;
}
```

Koncepty zmienne i funkcyjne - różnice

```
template <typename T, typename U>
concept bool Equality_comparable =
    requires(T t, U u) {
        { t == u } -> bool;
    };

template <typename T>
concept bool Equality_comparable =
    requires(T t1, T t2) {
        { t1 == t2 } -> bool;
    };
```

```
template <typename T>
requires Equality_comparable<T>
bool check(T && lhs, T && rhs) {
    return lhs == rhs;
}

int main() {
    return 0;
}
```

Diagnostyka

```
cov.cpp:11:14: error: redeclaration of ‘template<class T> concept const bool Equality_comparable’
 concept bool Equality_comparable =
              ^~~~~~~~~~~~~~~~~~~
cov.cpp:5:14: note: previous declaration ‘template<class T, class U> concept const bool
Equality_comparable<T, U>’
 concept bool Equality_comparable =
              ^~~~~~~~~~~~~~~~~~~
cov.cpp:17:10: error: invalid reference to concept ‘Equality_comparable<T>’
 requires Equality_comparable<T>
          ^~~~~~~~~~~~~~~~~~~~~
```

Problem #4: zmiana składni

```
template <typename T, typename U>
concept bool Equality_comparable() {
    return requires(T t, U u) {
        { t == u } -> bool;
    };
}

template <typename T, typename U>
requires Equality_comparable<T, U>()
bool check(T && lhs, U && rhs) {
    return lhs == rhs;
}
```

Problem #4: zmiana składni

```
template <typename T, typename U>
concept bool Equality_comparable =
    requires(T t, U u) {
        { t == u } -> bool;
    };


template <typename T, typename U>
requires Equality_comparable<T, U>
bool check(T && lhs, U && rhs) {
    return lhs == rhs;
}
```

Problem #5a: zbytnie ograniczenia

```
template <typename T, typename U>
concept bool Equality_comparable =
    requires(T t, U u) {
        { t == u } -> bool;
        { u == t } -> bool;
        { t != u } -> bool;
        { u != t } -> bool;
    };

template <typename T, typename U>
requires Equality_comparable<T, U>
bool check(T && lhs, U && rhs) {
    return lhs == rhs;
}
```

Diagnostyka

```
cr.cpp: In function ‘int main()’:
cr.cpp:20:23: error: cannot call function ‘bool check(T&&, U&&) [with T = X; U = X]’
  return check(X{}, X{});
                       ^
cr.cpp:13:6: note:  constraints not satisfied
 bool check(T && lhs, U && rhs) {
      ^~~~
cr.cpp:2:14: note: within ‘template<class T, class U> concept const bool
Equality_comparable<T, U> [with T = X; U = X]’
 concept bool Equality_comparable =
              ^~~~~~~~~~~~~~~~~~
cr.cpp:2:14: note:    with ‘X t’
cr.cpp:2:14: note:    with ‘X u’
cr.cpp:2:14: note: the required expression ‘(t == u)’ would be ill-formed
cr.cpp:2:14: note: the required expression ‘(u == t)’ would be ill-formed
cr.cpp:2:14: note: the required expression ‘(t != u)’ would be ill-formed
cr.cpp:2:14: note: the required expression ‘(u != t)’ would be ill-formed
```

Problem #5b: niedostateczne ograniczenia

```
template <typename T, typename U>
concept bool Equality_comparable =
    requires(T t, U u) {
        { t == u } -> bool;
    };

template <typename T, typename U>
requires Equality_comparable<T, U>
bool check(T && lhs, U && rhs) {
    return rhs == lhs;
}
```

```
class X{};
class Y{};

bool operator==(X& x, Y& y);

int main() {
    return check(X{}, Y{});
}
```

Diagnostyka

```
ci.cpp: In instantiation of ‘bool check(T&&, U&&) [with T = X; U = Y]’:
ci.cpp:21:23:   required from here
ci.cpp:10:16: error: no match for ‘operator==’ (operand types are ‘Y’ and ‘X’)
     return rhs == lhs;
            ~~~~^~~~~~
ci.cpp:16:6: note: candidate: bool operator==(const X&, const Y&)
 bool operator==(const X & x, const Y & y) {
      ^~~~~~~~
ci.cpp:16:6: note:  no known conversion for argument 1 from ‘Y’ to ‘const X&’
```

Concept introducer

```
template <typename T>
concept bool Int = std::is_integral<T>::value;

template <Int T>
void check(T&& t) {}
```

Concept introducer - diagnostyka

```
template <typename T>
concept bool Int = std::is_integral<T>::value;

template <Int T>
void check(T&& t) {}

class X{};

int main() {
    return check(X{});
}
```

Concept introducer - diagnostyka

```
ci.cpp: In function ‘int main()’:
ci.cpp:12:18: error: cannot call function ‘void check(T&&) [with T = X]’
  return check(X{});
                  ^
ci.cpp:7:6: note:  constraints not satisfied
 void check(T&& t) {}
      ^~~~~
ci.cpp:4:14: note: within ‘template<class T> concept const bool Int<T> [with T = X]’
 concept bool Int = std::is_integral<T>::value;
              ^~~
ci.cpp:4:14: note: ‘std::is_integral<X>::value’ evaluated to false
```

Problem #6

```
template <typename T>
concept bool Int = std::is_integral<T>::value;

template <Int T>
void check(T&& t) {}

class X{};

int main() {
    return check(X{});
}
```

Problem #6: bool?

```
template <typename T>
concept bool Int = std::is_integral<T>::value;
```

Składnia skrócona

```
template <typename T>
concept bool Int = std::is_integral<T>::value;


void check(Int&& t) {}
```

Problem #7: dwie różne składnie

```
template <Int T>
void check(T&& t) {}


void check(Int&& t) {}
```

auto-magiczne szablony

```
bool check(auto value);

template <typename T>
bool check(T value);
```

Rozbudowany przykład

```
template<typename T>
concept bool InputIterator = requires(T t) {
     { typename T::iterator_category{} } -> std::input_iterator_tag;
     };
template<typename T>
concept bool ForwardIterator = InputIterator<T>
     && requires(T t) {
     { typename T::iterator_category{} } -> std::forward_iterator_tag;
     };
template<typename T>
concept bool BidirectionalIterator = ForwardIterator<T>
     && requires(T t) {
     { typename T::iterator_category{} } -> std::bidirectional_iterator_tag;
     };
template<typename T>
concept bool RandomAccessIterator =  BidirectionalIterator<T>
     && requires(T t) {
     { typename T::iterator_category{} } -> std::random_access_iterator_tag;
     };
```

http://honermann.net/blog/2016/07/18/refining-concepts-improving-error-messages

Rozbudowany przykład

```
template<ForwardIterator T>
T next(T, typename T::difference_type = 1); // Uses operator++().
template<BidirectionalIterator T>
T next(T, typename T::difference_type = 1); // Uses operator++() and operator--().
template<RandomAccessIterator T>
T next(T, typename T::difference_type = 1); // Uses operator+=().

struct MyInputIter {
     using iterator_category = std::input_iterator_tag;
     using difference_type = int;
};

void f(MyInputIter i) {
     next(i);
}
```

http://honermann.net/blog/2016/07/18/refining-concepts-improving-error-messages

Problem #8: diagnostyka bywa gorsza

```
error.cpp: In function ‘void f(I)’:
error.cpp:46:11: error: no matching function for call to ‘next(I&)’
     next(i);
          ^
error.cpp:30:3: note: candidate: T next(T, typename T::difference_type) [with T = I; typename T::difference_type = int]
 T next(T, typename T::difference_type = 1); // Uses operator++().
   ^~~~
error.cpp:30:3: note:  constraints not satisfied
error.cpp:10:14: note: within ‘template<class T> concept const bool ForwardIterator<T> [with T = I]’
 concept bool ForwardIterator =
              ^~~~~~~~~~~~~~~
error.cpp:10:14: note:    with ‘I t’
error.cpp:10:14: note: ‘I::iterator_category{}’ is not implicitly convertible to ‘std::forward_iterator_tag’
error.cpp:32:3: note: candidate: T next(T, typename T::difference_type) [with T = I; typename T::difference_type = int]
 T next(T, typename T::difference_type = 1); // Uses operator++() and operator--().
   ^~~~
error.cpp:32:3: note:  constraints not satisfied
error.cpp:16:14: note: within ‘template<class T> concept const bool BidirectionalIterator<T> [with T = I]’
 concept bool BidirectionalIterator =
              ^~~~~~~~~~~~~~~~~~~~~
error.cpp:10:14: note: within ‘template<class T> concept const bool ForwardIterator<T> [with T = I]’
 concept bool ForwardIterator =
              ^~~~~~~~~~~~~~~
error.cpp:10:14: note:    with ‘I t’
error.cpp:10:14: note: ‘I::iterator_category{}’ is not implicitly convertible to ‘std::forward_iterator_tag’
error.cpp:16:14: note:    with ‘I t’
 concept bool BidirectionalIterator =
              ^~~~~~~~~~~~~~~~~~~~~
error.cpp:16:14: note: ‘I::iterator_category{}’ is not implicitly convertible to ‘std::bidirectional_iterator_tag’
error.cpp:34:3: note: candidate: T next(T, typename T::difference_type) [with T = I; typename T::difference_type = int]
 T next(T, typename T::difference_type = 1); // Uses operator+=().
   ^~~~
error.cpp:34:3: note:  constraints not satisfied
error.cpp:22:14: note: within ‘template<class T> concept const bool RandomAccessIterator<T> [with T = I]’
 concept bool RandomAccessIterator =
              ^~~~~~~~~~~~~~~~~~~~
error.cpp:16:14: note: within ‘template<class T> concept const bool BidirectionalIterator<T> [with T = I]’
 concept bool BidirectionalIterator =
              ^~~~~~~~~~~~~~~~~~~~~
error.cpp:10:14: note: within ‘template<class T> concept const bool ForwardIterator<T> [with T = I]’
 concept bool ForwardIterator =
              ^~~~~~~~~~~~~~~
error.cpp:10:14: note:    with ‘I t’
error.cpp:10:14: note: ‘I::iterator_category{}’ is not implicitly convertible to ‘std::forward_iterator_tag’
error.cpp:16:14: note:    with ‘I t’
 concept bool BidirectionalIterator =
              ^~~~~~~~~~~~~~~~~~~~~
error.cpp:16:14: note: ‘I::iterator_category{}’ is not implicitly convertible to ‘std::bidirectional_iterator_tag’
error.cpp:22:14: note:    with ‘I t’
 concept bool RandomAccessIterator =
```

Motywacja

Lepsza diagnostyka kompilatora

Kod szablonowy jest delikatny

Autor kodu może mieć (niejawne) założenia co do argumentów

Podsumowanie

1. requires requires
2. Brakujące definicje typowych konceptów
3. Funkcyjne/zmienne: 2 różne składnie
4. Zmiana składni utrudniona
5. Ograniczenia
   a. zbytnie
   b. niedostateczne
6. Wymagane “bool”
7. Concept introducer/składnia skrócona: 2 różne składnie
8. Diagnostyka czasem gorsza

Bibliografia

1. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4377.pdf
2. 'C++ Concepts "Lite" in Practice' - Roger Orr (ACCU 2016)
3. http://honermann.net/blog/category/c-concepts/
4. 'Fastware' - Andrei Alexandrescu (ACCU 2016)

Kontakt

Piotr Kozłowski

Senior Software Developer@Codilime

Go, Python & JS

piotr@pkozlowski.pl

Pytania?

Dziękuję za uwagę

Mam Ochotę
Hasło “koncepty”