

# Template-parameter dependent name

Adam Mizerski  
adam@mizerski.pl

Warsaw C++ Users Group

11 stycznia 2017

## Dawno, dawno temu...

- ▶ Potrzebowałem klasy – opakowanie na kontener.
- ▶ Z własnym iteratorem – opakowaniem na iterator kontenera.
- ▶ Zawartość dana przez konstruktor, dane tylko do odczytu.

---

```
1 const auto things = Container{"a", "b", "c"};
2 int main() {
3     for (const auto& thing : things) {
4         [...]
5     }
6     auto a = things.get("a");
7     auto b = things.get("b");
8     a.name(); // "a"
9     a < b; // szybsze od strcmp
10    (++a) == b; // j.w.
11 }
```

---

# Kontener

Wersja uproszczona na potrzeby prezentacji

---

```
1 class Container {
2     boost::container::flat_set<std::string> mSet;
3 public:
4     Container(std::initializer_list<std::string> l) : mSet{l} {}
5
6     class It [...];
7
8     It begin() const { return {mSet.cbegin()}; }
9     It end() const { return {mSet.cend()}; }
10 };
```

---

# Iterator

---

```
1 class It : public boost::iterator_facade<It, const std::string,  
2           boost::random_access_traversal_tag> {  
3     friend boost::iterator_core_access;  
4     friend Container;  
5     using SetIt = typename decltype(Container::mSet)::const_iterator;  
6     SetIt mlt;  
7     It(SetIt it) : mlt{it} {}
```

---

# Iterator

---

```
8     reference dereference() const { return *mIt; }
9
10    bool equal(const It& other) const { return mIt == other.mIt; }
11
12    void increment() { ++mIt; }
13
14    void decrement() { --mIt; }
15
16    void advance(difference_type n) { mIt += n; }
17
18    difference_type distance_to(const It& other) const
19        { return other.mIt - mIt; }
20 };
```

---

# Test

---

```
1 int main() {  
2     for (const auto& s : Container{"a", "b", "c"}) {  
3         std::cout << s << ' ';  
4     }  
5     std::cout << '\n';  
6 }
```

---

```
$ g++ -std=c++11 1.cpp -o 1
```

```
$ ./1
```

```
a b c
```

# Szablon

Niech `std::string` będzie parametrem szablonu.

# Szablon

---

```
1 template <typename T>
2 class Container {
3     boost::container::flat_set<T> mSet;
4 public:
5     Container(std::initializer_list<T> l) : mSet{l} {}
6
7     class It : public boost::iterator_facade<It, const T,
8                 boost::random_access_traversal_tag> {
9     [...]
10
11 int main() {
12     for (const auto& s : Container<std::string>{"a", "b", "c"}) {
13     [...]
```

---



## Nope

```
g++ -std=c++11 2.cpp -o 2
2.cpp:23:9: error: 'reference' does not name a type
      reference dereference() const { return *mIt; }
      ~~~~~
2.cpp:31:22: error: 'difference_type' has not been declared
      void advance(difference_type n) { mIt += n; }
                   ~~~~~
2.cpp:33:9: error: 'difference_type' does not name a type
      difference_type distance_to(const It& other) const
      ~~~~~
[...]
```



Co się stało?

Template-parameter dependent name

## Prosty przykład

---

```
1 template <typename T>  
2 void foo() {  
3     T::bar * i;  
4 }
```

---

## Drugi przykład

---

```
1 template <typename T>
2 void foo() {
3     T::bar < 5 > i;
4 }
```

---

Co robić? Jak żyć?

Kompilator potrzebuje odpowiedzi

## Domyślnie: zmienne

---

```
1  static const int i = 0;
2
3  template <typename T>
4  void foo() {
5      T::bar * i;
6      T::bar < 5 > i;
7  }
8
9  struct Bar {
10     static const int bar = 0;
11 };
12
13 int main() {
14     foo<Bar>();
15 }
```

---

# Typy

---

```
1 template <typename T>
2 void foo() {
3     typename T::bar * i;
4 }
5
6 struct Bar {
7     using bar = int;
8 };
9
10 int main() {
11     foo<Bar>();
12 }
```

---



# Metody Szablonowe

---

```
1  template <typename T>
2  void foo() {
3      T t;
4      t.template bar<5>();
5  }
6
7  struct Bar {
8      template <int I>
9      void bar() {}
10 };
11
12 int main() {
13     foo<Bar>();
14 }
```

---

# Typy Szablonowe

---

```
1 template <typename T>
2 void foo() {
3     typename T::template bar<5> i;
4 }
5
6 struct Bar {
7     template <int I>
8     struct bar {};
9 };
10
11 int main() {
12     foo<Bar>();
13 }
```

---

Zakończenie opowieści

Co się stało w `Container::It`?

## Co się stało w `Container::It`?

- ▶ `Container<T>` jest klasą szablonową,
- ▶ więc `Container<T>::It` jest klasą szablonową,
- ▶ więc `boost::iterator_facade<Container<T>::It, const T, boost::random_access_traversal_tag>` jest klasą szablonową,
- ▶ więc w `Container<T>::It` nazwy `reference` i `difference_type` stały się zależne od argumentu szablonu.

## Rozwiązanie problemu

---

```
1 class It : public boost::iterator_facade<It, const T,  
2           boost::random_access_traversal_tag> {  
3   using Base = boost::iterator_facade<It, const T,  
4           boost::random_access_traversal_tag>;  
5   using reference = typename Base::reference;  
6   // typedef typename Base::reference reference;  
7   using difference_type = typename Base::difference_type;  
8  
9   [...]
```

---

Dziękuję za uwagę