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.

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

Kontener

Wersja uproszczona na potrzeby prezentacji

```
class Container {
   boost::container::flat_set<std::string> mSet;

public:
   Container(std::initializer_list<std::string> l) : mSet{l} {}

class It [...];

lt begin() const { return {mSet.cbegin()}; }

lt end() const { return {mSet.cend()}; }

};
```

Iterator

```
class It : public boost::iterator_facade<It, const std::string,
boost::random_access_traversal_tag> {
friend boost::iterator_core_access;
friend Container;
using SetIt = typename decItype(Container::mSet)::const_iterator;
SetIt mIt;
It(SetIt it) : mIt{it} {}
```

Iterator

```
reference dereference() const { return *mlt; }
8
        bool equal(const lt& other) const { return mlt == other.mlt; }
10
11
        void increment() { ++mlt; }
12
13
        void decrement() { --mlt; }
14
15
        void advance(difference type n) \{ mlt += n; \}
16
17
        difference type distance to(const lt& other) const
18
            { return other.mlt - mlt; }
19
20
```

Test

```
int main() {
    for (const auto& s : Container{"a", "b", "c"}) {
        std::cout << s << ' ';
    }
    std::cout << '\n';
    }
}</pre>
```

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

Szablon

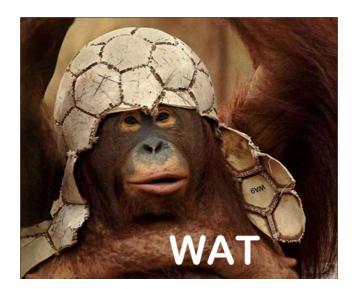
Niech std::string będzie parametrem szablonu.

Szablon

```
template <typename T>
    class Container {
        boost::container::flat set<T> mSet;
    public:
        Container(std::initializer list<T> I) : mSet{I} {}
        class It : public boost::iterator facade<It, const T,
                          boost::random access traversal tag> {
8
    [...]
10
    int main() {
11
        for (const auto& s : Container<std::string>{"a", "b", "c"}) {
12
    [...]
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

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

Drugi przykład

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

Co robić? Jak żyć?

Kompilator potrzebuje podpowiedzi

Domyślnie: zmienne

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

Туру

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

Metody Szablonowe

```
template <typename T>
   void foo() {
        Tt;
        t.template bar<5>();
5
6
   struct Bar {
        template <int |>
8
        void bar() {}
9
   };
10
11
   int main() {
12
        foo < Bar > ();
13
14
```

Typy Szablonowe

```
template <typename T>
   void foo() {
       typename T::template bar<5> i;
5
   struct Bar {
       template <int |>
       struct bar {};
   };
10
   int main() {
11
       foo<Bar>();
12
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

```
class It : public boost::iterator_facade<It, const T,
boost::random_access_traversal_tag> {
    using Base = boost::iterator_facade<It, const T,
    boost::random_access_traversal_tag>;
    using reference = typename Base::reference;
// typedef typename Base::reference;
using difference_type = typename Base::difference_type;

[...]
```

Dziękuję za uwagę