Why is there a separation of algorithms, iterators and containers in C++ STL

Ask Question

I can't figure out why they have separated algorithms, iterators and containers in C++ STL. If it's a heavy use of templates everywhere, then we can have classes having all stuff in one place with template parameters.

Some text that I got explains that iterators helps algorithms to interact with containers data but what if containers expose some mechanism to access the stl iterator

containers

edited Aug 14 '12 at 8:38

TemplateRex

52.2k 14 118 228

asked Aug 14 '12 at 6:50

Rahul

703 2 10 30

1 I didn't understand a word you wrote. :(– Mehrdad Aug 14 '12 at 6:51

Ok sorry for confusion caused, what I mean is we have different classes for containers, iterators etc. I want to figure what's wrong if we put all in one class using templates, containers some interfaces to see it or modify. why they are separate? I mean why there are different iterators, algorithms etc. – Rahul Aug 14 '12 at 6:54

3 This question might give you some pointers. This interview with Alex Stephanov, the creator of the STL, also contains some insights. – Björn Pollex Aug 14 '12 at 6:56

12 The question might not be

an answer would be that `M` containers + `N` algorithms would normally require `M * N` pieces of code, but with iterators acting as "glue", you can have only `M + N` pieces of code. – TemplateRex Aug 14 '12 at 6:59

1 @rhalbersma: Voted for reopen, and your comment is the best answer I could come up with myself. – DevSolar Aug 14

With `M` containers + `N` algorithms, one would normally need `M * N` pieces of code, but with iterators acting as "glue", this can be reduced to `M + N` pieces of code.

Example: run 2 algorithms on 3 containers

```
std::list<in
std::vector<
std::array<i

auto l_conta
auto v_conta
auto a_conta

auto l_count
auto v_count
auto a_count
```

You are calling only 2 different algorithms, and only have code for 3 containers. Each container passes the `begin()` and `end()` iterators to the container. Even though you have `3 *` pieces of functionality that need to be written.

For more containers and algorithms, this separation is an enormous reduction in the combinatorial explosion in code that would otherwise ensue: there are 5 sequence containers, 8 associative containers and 3 container adapters in the STL, and there are almost 80 algorithms in `<algorithm>` alone (not even counting those in `<numeric>`) so that you have only `16 + 80` instead of `16 * 80` , an 13-fold reduction in code! (Of course, not every algorithm makes sense

The iterators can be divided into 5 categories (input, output, forward, bidirectional and random access), and some algorithms will delegate to specialized versions depending on the iterator capabilities. This will diminish the code reduction somewhat, but greatly improve efficiency by selecting the best adapted algorithm to the iterator at hand.

Note that the STL is not completely consistent in the separation: `std::list` has its own `sort` member function that uses implementation specific details to sort itself, and `std::string` has an could have been implemented as non-member functions.

lited Aug 14 '12 at 8:57

ıswered Aug 14 '12 at 8:22

TemplateRex
52.2k 14 118 228