Replace OutputIterator and Extended Range

Akira Takahashi (Japan)

LongGate CO.,LTD.

Site: https://sites.google.com/site/faithandbrave/about/en

Twitter: @cpp_akira

C++Now! 2012 Library in a Week

Akira Takahashi profile

- C++ Standard Committee, Japan Expert Member
- P-Stade C++ Libraries committer
- Japanese C++ Community Manager
  - boostjp : Boost Japanese Information Site
    https://sites.google.com/site/boostjp/
  - cpprefjp: C++11 Library Reference Site
    https://sites.google.com/site/cpprefjp/
  - Blog : Japanese C++ Programmers Activity
    http://cppjp.blogspot.com/
  - Boost.StudyMeeting (so to say, Japanese BoostCon/C++Now!)
    - participation person is 100+
- My Book : C++ Template Techniques
  http://www.amazon.co.jp/dp/4797354534/
- My Magazine: Programmers' Grimoire
  http://longgate.co.jp/products.html

note:

I can speak very little English!

I may not answer your question immediately…

1st

OutputIterators Must Go

This Idea Overview

- Output Iterators are now unnecessary because C++11 is there.
- Some STL algorithm can replace from Output Iterator to UnaryFunction.

Basic Example: std::copy

std::copy can replace std::for_each with lambda.

Before:

```
std::vector<int> v = {1, 2, 3};
std::vector<int> result;

std::copy(v.begin(), v.end(), std::back_inserter(result));
```

After:

```
std::vector<int> v = {1, 2, 3};
std::vector<int> result;

std::for_each(v.begin(), v.end(),
                  [&](int x) { result.push_back(x); });
```

This replacement is failry useful.

More Useful Example:
set_union, set_intersection, set_difference

STL set algorithms using Output Iterator aren't useful.

Now STL Algorithm

```
std::set<int> a = {1, 2, 3};
std::set<int> b = {4, 5, 6};
std::set<int> result;

std::set_union(a.begin(), a.end(),
                b.begin(), b.end(),
                std::inserter(result, result.end()));
```

Insert Iterator Adaptor is not useful!
Custom operation is not easy.

More Useful Example:
set_union, set_intersection, set_difference

STL set algorithm using Output Iterator. Not useful.

New STL Algorithm

```
std::set<int> a = {1, 2, 3};
std::set<int> b = {4, 5, 6};
std::set<int> result;

make_union(a.begin(), a.end(),
            b.begin(), b.end(),
            [](int x) { result.insert(x); });
```

Output Iterator canreplace to UnaryFunction.
It's accutually useful, easily to customize operation.
This implementation is here:
https://github.com/faithandbrave/Set-Algorithm

2nd

OvenToBoost project

OvenToBoost project overview

- Oven is Range Library in P-Stade C++ Libraries
- Oven is more useful than Boost.Range
- OvenToBoost project is porting from Oven To Boost as extended Boost.Range
- https://github.com/faithandbrave/OvenToBoost

Boost.Range issues

- There are not many Range adaptors.
  - nothing "taken"
  - nothing "dropped"
  - nothing Infinite Range
  - etc…
- Boost.Range's Range adaptors can't use lambda
- Oven has solution for these issues

taken Range Adaptor

```
const std::vector<int> v = {3, 1, 4, 2, 5};

boost::for_each(v | taken(2), print);
```

```
3
1
```

dropped Range Adaptor

```
const std::vector<int> v = {3, 1, 4, 2, 5};

boost::for_each(v | dropped(2), print);
```

```
4
2
5
```

elements Range Adaptor

```
struct Person {
  int id;
  std::string name;
  …
};
BOOST_FUSION_ADAPT_STRUCT(…)

const std::vector<Person> v = {
    {1, "Alice"}
    {2, "Carol"}
    {3, "Bob"}
 };

boost::for_each(v | elements<1>(), print);
```

```
Alice,Carol,Bob
```

elements_key Range Adaptor

```
struct id_tag {}; struct name_tag {};

struct Person {
  int id;
  std::string name;
  …
};
BOOST_FUSION_ADAPT_ASSOC_STRUCT(…)

const std::vector<Person> v = {
    {1, "Alice"}
    {2, "Carol"}
    {3, "Bob"}
 };

boost::for_each(v | elements_key<name_tag>(), print);
```

```
Alice,Carol,Bob
```

iteration function

```
int next(int x) { return x * 2; }

boost::for_each(iteration(1, next) | taken(5), print);
```

```
1
2
4
8
16
```

regular function

```
template <class InputIterator, class F>
F for_each_(InputIterator first, InputIterator last, F f) {
  InputIterator it; // default construct
  it = first; // copy assign

  while (it != last) { f(*it); ++i; }
  return f;
}

template <class Range, class F>
F for_each_(const Range& r, F f)
{ return for_each(boost::begin(r), boost::end(r), f); }

using boost::lambda::_1;
for_each_(r | filtered(_1 % 2 == 0), f);          // Error!
for_each_(r | filtered(regular(_1 % 2 == 0)), f); // OK
```

regular operator|+()

```
template <class InputIterator, class F>
F for_each_(InputIterator first, InputIterator last, F f) {
  InputIterator it; // default construct
  it = first; // copy assign

  while (it != last) { f(*it); ++i; }
  return f;
}

template <class Range, class F>
F for_each_(const Range& r, F f)
{ return for_each(boost::begin(r), boost::end(r), f); }

using boost::lambda::_1;
for_each_(r |  filtered(_1 % 2 == 0), f); // Error!
for_each_(r |+ filtered(_1 % 2 == 0), f); // OK
```

Combination Example: Prime list

```
range sieve(range r)
{
  return r | dropped(1) |+ filtered(_1 % value_front(r) != 0);
}

range primes =
    iteration(range(
      iteration(2, regular(_1 + 1))), sieve) | transformed(value_front);


for_each(primes, print);
```

```
2 3 5 7 11 …
```

OvenToBoost now status

- Primary implementation has been complete.
- Test has been complete.
- But documentation is late…
- I would like to submit a review request to Boost.