Document number: **N3643**
Date: 2013-04-16
Project: Programming Language C++
Reference: N3485
Reply to: **Alan Talbot**
cpp@alantalbot.com

Range Adaptor for Selecting from Pair or Tuple

I use maps a lot, for a lot of different things, but my most common use case is to implement a database-like table, with the primary key (ID) as the key type of the map and a record class as the mapped type. For this use, almost all of my iterator operations involve only the mapped type of the value pair—the key is used only to look up a record, or occasionally to access the ID (but my records almost always have to know their own ID). Because of the nature of the map interface, this means my code usually looks like this (or will, once I have a compiler with range-based for loops):

```
for (auto& i : m)
{
   i.second.foo();
   i.second.bar();
}
```

This is a notational nuisance, but the problem becomes much worse in generic contexts:

```
template<typename C>
void print(const C& c)
{
   for (const auto& i : c)
      cout << i << endl;
}
```

I would like to call this with whatever container I happen to be using, but **operator<<** isn't overloaded on pair so it won't compile for maps. And if I implement **operator<<** for pairs, I still want to be able choose whether to print the key/mapped pair or only the key type or mapped type.

The solution I'm proposing is to create a **selector_t** wrapper for types that have iterators. This wrapper replaces the native iterators with ones that dereference a selected member of the value to which the native iterator refers. It takes an integer or type template argument and uses it to access the selected member with **get**. The wrapper works on any container type that provides a **begin** and **end**, and has elements of a type that provides **get**.

There is also a convenience function **select** which provides automatic creation of the wrapper type. (Another possible name would be **make_selector**.) Since this will be used frequently with maps, **select_key** and **select_value** will also be provided (meaning **select<0>** and **select<1>** respectively).

These tools solve my map problem without making any changes to map, and offer other advantages. For instance I can iterate over a vector of tuples, selecting only the third value.

With **select**, my code can look like this:

```
for (auto& i : select<1>(m))
{
   i.foo();
   i.bar();
}
```

And I can use my generic print function:

```
print(select<0>(m));        // Print only the key type.
print(select<1>(m));        // Print only the mapped type.
print(m);                   // Print the pair (given an operator<<).
```

Here is the same code using the convenience functions:

```
print(select_key(m));       // Print only the key type.
print(select_value(m));     // Print only the mapped type.
```

Selection can also be composed:

```
map<int, tuple<int, float, string>> m = ...
print(select<string>(select_value(m))); // Print each string in the map.
```

Proposed Wording

Complete wording will be provided in a revision of this paper.

Acknowledgements

Stephan Lavavej suggested making what was originally a mapped-type adaptor more general.