# Conceptualizing the Range-Based `for` Loop

Author: Douglas Gregor, Indiana University <dgregor@cs.indiana.edu>
Document number: N2049=06-0119
Date: June 24, 2006
Project: Programming Language C++, Evolution Working Group

The new range-based `for` loop (N1961, N1868) drastically simplifies iteration over containers, with a new syntax that is concise, easy to teach, and easy to use:

```
vector<int> vec = ...;
for( int i : vec )
    std::cout << i;
```

Of course, as well as working with library-defined containers and built-in arrays, the range-based `for` loop is extensible to user-defined sequences and containers. Unfortunately, this extensibility relies on argument-dependent lookup, the introduction of four new function names into the library that extract iterators from sequences and containers (range_begin(), range_end(), begin(), and end()), and a series of highly-generalized, unsafe function templates that make containers work with the range-based `for` loop. Concepts (N2042) allow us to restate the ideas of that proposal directly in C++0x, providing a cleaner, safer implementation of the range-based `for` loop without changing the intended syntax or semantics.

We begin by building a concept For that captures all of the functionality we need to iterate over a sequence or container. Like the pre-concept `for` proposal, we iterate over the iterator range [begin(c), end(c)). Unlike the pre-concept version, however, we place begin() and end() inside a *concept*:

```
concept For<typename C> {
  InputIterator iterator;
  iterator begin(C&);
  iterator end(C&);
}
```

Using this concept, we make the range-based `for` statement:

```
for( type-specifier-seq simple-declarator : expression ) statement
```

syntactically equivalent to

```
{
    typedef decltype(expression) _C;
    auto&& __rng(( expression ));

    for( auto __begin( std::For<_C>::begin(__rng) ), __end( std::For<_C>::end(__rng) );
          __begin != __end; ++__begin ) {
```

```
        type-specifier-seq simple-declarator ( *__begin );
        statement
    }
}
```

The range-based `for` loop works for any type C that meets the requirements of the concept For. One can state that a certain type or set of types C meets these requirements, and how those requirements are met, with a *concept map*. For instance, the following concept map makes it possible to use the range-based `for` loop with arrays:

```
template<typename T, size_t N>
concept_map For<T[N]> {
  typedef T* iterator;
  T* begin(T (&array)[N]) { return array; }
  T* end(T (&array)[N]) { return array + N; }
}
```

The range-based `for` proposal also allows iteration over pairs of iterators. We implement the same functionality with concept maps defined only for pairs of input iterators:

```
template<InputIterator Iter >
concept_map For<pair<Iter, Iter> > {
  typedef Iter  iterator ;
  Iter  begin(pair <Iter,  Iter >& p)
  { return p.first  ; }
  Iter  end(pair <Iter,  Iter >& p)
  { return p.second; }
}
```

```
template<InputIterator Iter >
concept_map For<const pair<Iter, Iter> > {
  typedef Iter  iterator ;
  Iter  begin(const pair <Iter,  Iter >& p)
  { return p.first  ; }
  Iter  end(const pair <Iter,  Iter >& p)
  { return p.second; }
}
```

Finally, we can support iteration over the contents of any Container:

```
template<Container C>
concept_map For<C> {
  typedef C::iterator  iterator ;
  iterator  begin(C& c)
  { return c.begin(); }
  iterator  end(C& c)
  { return c.end(); }
}
```

```
template<Container C>
concept_map For<const C> {
  typedef C::const_iterator  iterator ;
  iterator  begin(const C& c)
  { return c.begin(); }
  iterator  end(const C& c)
  { return c.end(); }
}
```

User-defined Containers will work with the range-based `for` loop through these model templates, and users are, of course, free to provide their own concept maps for anything that permits iteration.

The For concept and all of its concept maps will be placed in a new header, `<for>`, which must be included before the range-based `for` loop can be used. Each of the Standard Library container headers (`<vector>`, `<map>`, etc.) imply the inclusion of `<for>`.

Using concepts, we can simplify the implementation and extension of range-based `for` loops, eliminating the confusion caused by argument-dependent name lookup, the distinction between range_begin() and begin(), and the poor error messages that will result from instantiation-time failures in the library-provided range_begin() and range_end(). We still retain the same flexibility and extensibility as the pre-concept range-based `for` proposal.