Indirect Iterator

Author: David Abrahams, Jeremy Siek, Thomas Witt
Contact: dave@boost-consulting.com, jsiek@osl.iu.edu, witt@ive.uni-hannover.de
Organization: Boost Consulting, Indiana University Open Systems Lab, University of Hanover Institute for Transport Railway Operation and Construction
Date: 2004-11-01

abstract: `indirect_iterator` adapts an iterator by applying an *extra* dereference inside of `operator*()`. For example, this iterator adaptor makes it possible to view a container of pointers (e.g. `list<foo*>`) as if it were a container of the pointed-to type (e.g. `list<foo>`). `indirect_iterator` depends on two auxiliary traits, `pointee` and `indirect_reference`, to provide support for underlying iterators whose `value_type` is not an iterator.

Table of Contents

`indirect_iterator` synopsis

`indirect_iterator` requirements

`indirect_iterator` models

`indirect_iterator` operations

Example

`indirect_iterator` synopsis

```
template <
    class Iterator
  , class Value = use_default
  , class CategoryOrTraversal = use_default
  , class Reference = use_default
  , class Difference = use_default
>
class indirect_iterator
{
 public:
    typedef /* see below */ value_type;
    typedef /* see below */ reference;
    typedef /* see below */ pointer;
    typedef /* see below */ difference_type;
    typedef /* see below */ iterator_category;
    indirect_iterator();
    indirect_iterator(Iterator x);

    template <
        class Iterator2, class Value2, class Category2
      , class Reference2, class Difference2
    >
    indirect_iterator(
        indirect_iterator<
             Iterator2, Value2, Category2, Reference2, Difference2
        > const& y
      , typename enable_if_convertible<Iterator2, Iterator>::type* = 0 // exposition
    );

    Iterator const& base() const;
    reference operator*() const;
    indirect_iterator& operator++();
    indirect_iterator& operator--();
private:
   Iterator m_iterator; // exposition
};
```

The member types of `indirect_iterator` are defined according to the following pseudo-code, where `V` is `iterator_traits<Iterator>::value_type`

```
if (Value is use_default) then
    typedef remove_const<pointee<V>::type>::type value_type;
else
    typedef remove_const<Value>::type value_type;

if (Reference is use_default) then
    if (Value is use_default) then
        typedef indirect_reference<V>::type reference;
    else
        typedef Value& reference;
else
    typedef Reference reference;

if (Value is use_default) then
    typedef pointee<V>::type* pointer;
else
    typedef Value* pointer;

if (Difference is use_default)
    typedef iterator_traits<Iterator>::difference_type difference_type;
else
    typedef Difference difference_type;

if (CategoryOrTraversal is use_default)
    typedef iterator-category (
        iterator_traversal<Iterator>::type,‘‘reference‘‘,‘‘value_type‘‘
    ) iterator_category;
else
```

```
typedef iterator-category (
    CategoryOrTraversal,``reference``,``value_type``
) iterator_category;
```

`indirect_iterator` requirements

The expression `*v`, where `v` is an object of `iterator_traits<Iterator>::value_type`, shall be valid expression and convertible to `reference`. `Iterator` shall model the traversal concept indicated by `iterator_category`. `Value`, `Reference`, and `Difference` shall be chosen so that `value_type`, `reference`, and `difference_type` meet the requirements indicated by `iterator_category`.

[Note: there are further requirements on the `iterator_traits<Iterator>::value_type` if the `Value` parameter is not `use_default`, as implied by the algorithm for deducing the default for the `value_type` member.]

`indirect_iterator` models

In addition to the concepts indicated by `iterator_category` and by `iterator_traversal<indirect_iterator>::type`, a specialization of `indirect_iterator` models the following concepts, Where `v` is an object of `iterator_traits<Iterator>::value_type`:

- Readable Iterator if `reference(*v)` is convertible to `value_type`.
- Writable Iterator if `reference(*v) = t` is a valid expression (where `t` is an object of type `indirect_iterator::value_type`)
- Lvalue Iterator if `reference` is a reference type.

`indirect_iterator<X,V1,C1,R1,D1>` is interoperable with `indirect_iterator<Y,V2,C2,R2,D2>` if and only if `X` is interoperable with `Y`.

`indirect_iterator` operations

In addition to the operations required by the concepts described above, specializations of `indirect_iterator` provide the following operations.

`indirect_iterator();`

Requires: `Iterator` must be Default Constructible.

Effects: Constructs an instance of `indirect_iterator` with a default-constructed `m_iterator`.

`indirect_iterator(Iterator x);`

Effects: Constructs an instance of `indirect_iterator` with `m_iterator` copy constructed from `x`.

```
template <
    class Iterator2, class Value2, unsigned Access, class Traversal
  , class Reference2, class Difference2
>
indirect_iterator(
    indirect_iterator<
         Iterator2, Value2, Access, Traversal, Reference2, Difference2
    > const& y
  , typename enable_if_convertible<Iterator2, Iterator>::type* = 0 // expo-
sition
);
```

Requires: `Iterator2` is implicitly convertible to `Iterator`.

Effects: Constructs an instance of `indirect_iterator` whose `m_iterator` subobject is constructed from `y.base()`.

```
Iterator const& base() const;
```

Returns: `m_iterator`

```
reference operator*() const;
```

Returns: `**m_iterator`

```
indirect_iterator& operator++();
```

Effects: `++m_iterator`

Returns: `*this`

```
indirect_iterator& operator--();
```

Effects: `--m_iterator`

Returns: `*this`

Example

This example prints an array of characters, using `indirect_iterator` to access the array of characters through an array of pointers. Next `indirect_iterator` is used with the `transform` algorithm to copy the characters (incremented by one) to another array. A constant indirect iterator is used for the source and a mutable indirect iterator is used for the destination. The last part of the example prints the original array of characters, but this time using the `make_indirect_iterator` helper function.

```
char characters[] = "abcdefg";
const int N = sizeof(characters)/sizeof(char) - 1; // -
1 since characters has a null char
char* pointers_to_chars[N];                        // at the end.
for (int i = 0; i < N; ++i)
  pointers_to_chars[i] = &characters[i];

// Example of using indirect_iterator

boost::indirect_iterator<char**, char>
  indirect_first(pointers_to_chars), indirect_last(pointers_to_chars + N);

std::copy(indirect_first, indi-
rect_last, std::ostream_iterator<char>(std::cout, ","));
std::cout << std::endl;


// Example of making mutable and constant indirect iterators

char mutable_characters[N];
char* pointers_to_mutable_chars[N];
for (int j = 0; j < N; ++j)
  pointers_to_mutable_chars[j] = &mutable_characters[j];
```

```
boost::indirect_iterator<char* const*> muta-
ble_indirect_first(pointers_to_mutable_chars),
  mutable_indirect_last(pointers_to_mutable_chars + N);
boost::indirect_iterator<char* const*, char const> const_indirect_first(pointers_to_chars),
  const_indirect_last(pointers_to_chars + N);

std::transform(const_indirect_first, const_indirect_last,
               mutable_indirect_first, std::bind1st(std::plus<char>(), 1));

std::copy(mutable_indirect_first, mutable_indirect_last,
          std::ostream_iterator<char>(std::cout, ","));
std::cout << std::endl;


// Example of using make_indirect_iterator()

std::copy(boost::make_indirect_iterator(pointers_to_chars),
          boost::make_indirect_iterator(pointers_to_chars + N),
          std::ostream_iterator<char>(std::cout, ","));
std::cout << std::endl;
```

The output is:

```
a,b,c,d,e,f,g,
b,c,d,e,f,g,h,
a,b,c,d,e,f,g,
```

The source code for this example can be found here.