Iterators and Ranges for numerical problems

Karsten Ahnert

Ambrosys GmbH, Potsdam

November 7, 2014

Outline

Introduction

Iterators

Unique way to traverse containers
Unique way to apply iterative IO
Unique way of expressing algorithms

Example – basic use

```
for( auto iter = values.begin() ;
     iter != values.end() ;
     ++iter )
{
    cout << *iter << endl;
}
```

Example – basic use

```cpp
for( auto iter = values.begin() ;
     iter != values.end() ;
     ++iter )
{
    cout << *iter << endl;
}
```

C++11 - use range based for

```cpp
for( auto v : values )
{
    cout << v << endl;
}
```

Example – Container traversal

```
list< double > values;
list< double > values2( values.size() );
```

Can be used in

```
transform( values.begin() , values.end() ,
           values2.begin() ,
           []( double x ) {
               return x * 2.0; } );
```

Example – Container traversal

```
vector< double > values;
vector< double > values2( values.size() );
```

Can be used in

```
transform( values.begin() , values.end() ,
           values2.begin() ,
           []( double x ) {
               return x * 2.0; } );
```

Examples – IO

Input

```
vector< double > values;
copy_if( istream_iterator< double >( cout ) ,
         istream_iterator< double >() ,
         back_inserter( values ) ,
         []( double x ) { return x > 0.0; } );
```

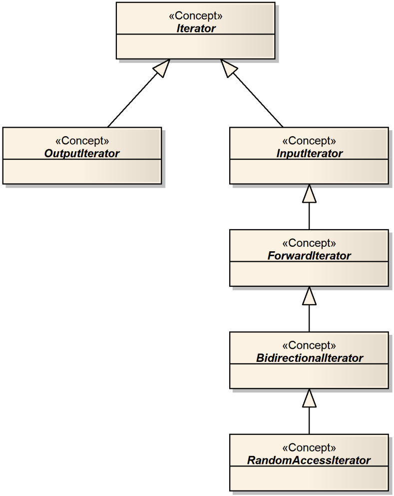
Output

```
vector< double > values;
// fill values
copy_if( values.begin() , values.end() ,
         ostream_iterator< double >( std::cout , "\n" ) ,
         []( double x ) { return x > 0.0; } );
```

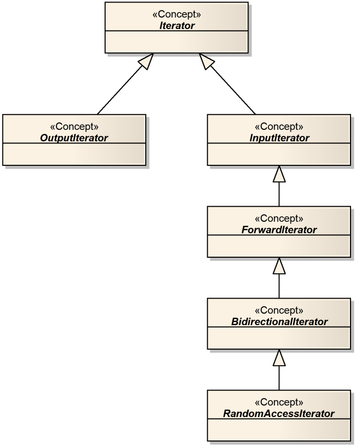
Examples – Combine algorithms

Find a nice real life example.

Iterator types – Concepts



Iterator types – Concepts

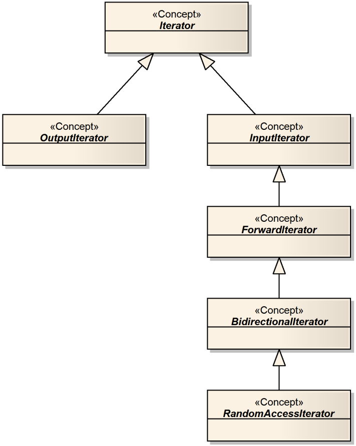


OutputIterator

```
*i = o;
*i++ = o;
i++;
++i;
```

Are special, `back_inserter`, `ostream_iterator`, ...

Iterator types – Concepts



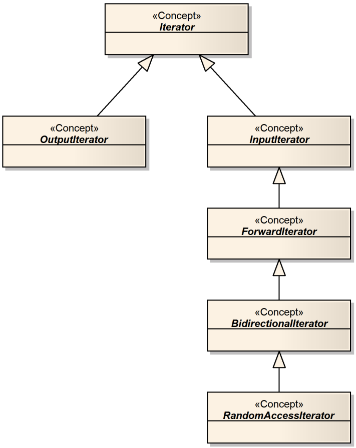
InputIterator a.k.a. Single-Pass Iterator

```
bool r = i != j;
val x = *i;
iterator j = ++i;
i++;
val x = *i++;
```

`istream_iterator`, `istreambuf_iterator`

But, if `i == j` then `++i != ++j`

Iterator types – Concepts

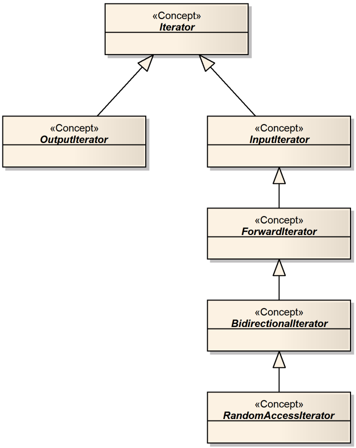


ForwardIterator

```
iterator j = i++;
```

But, if `i == j` then `++i == ++j`

Iterator types – Concepts

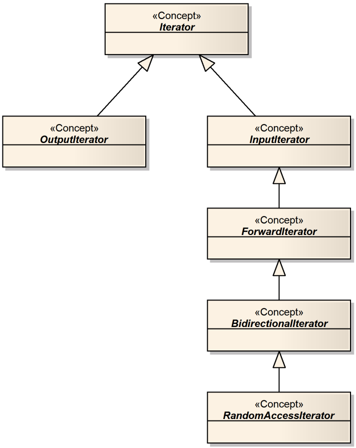


BidirectionalIterator

```
iterator j = --i;
iterator j = i--;
val x = *i--;
```

```
map< K , V >::iterator,
list< T >::iterator
```

Iterator types – Concepts



RandomAccessIterator

```
i += n;
i -= n;
val x = i[n];
long dist = i - j;
bool b = i < j;
```

`vector< T >::iterator`

Algorithms

```
I all_of
I any_of
I none_of
I for_each
I count
I count_if
I mismatch
I equal
I find
I find_if
I find_if_not
F find_end
I,F find_first_if
F adjacent_find
F search
F search_n
```

```
I,O copy
I,O copy_if
I,O copy_n
B,O copy_backward
I,O move
B,O move_backward
F fill
F fill_n
I,O transform
F generate
I generate_n
```

```
F remove
F remove_if
I,O remove_copy
I,O
remove_copy_if
F replace
F replace_if
I,O replace_copy
I,O
replace_copy_if
F swap_ranges
F iter_swap
B reverse
B,O reverse_copy
F rotate
F,O rotate_copy
R random_shuffle
R shuffle
F unique
I,O unique_copy
```

```
I is_partitioned
F,B partition
I,O partition_copy
B stable_partition
F partition_point
F is_sorted
F is_sorted_until
R sort
R partial_sort
I,R partial_sort_copy
R stable_sort
R nth_element
```

```
F lower_bound
F upper_bound
F binary_search
F equal_range
I,O merge
B inplace_merge
I includes
I,O set_difference
I,O set_intersection
I,O
set_symmetric_difference
I,O set_unition
```

```
R is_heap
R is_heap_until
R make_heap
R push_heap
R pop_heap
R sort_heap
```

```
F max_element
F min_element
F minmax_element
I lexicographical_compare
F is_permutation
B next_permutation
B prev_permutation
```

```
F iota
I accumulate
I inner_product
I,O adjacent_difference
I,O partial_sum
```

Ranges

Simplifying iterators
Generalization of iterators
First defined in Boost
Soon in the standard library?

```
vector< double > values;
boost::for_each( values , []( double x ) { cout
     << x << endl; } );
```

Ranges – more examples from boost

Filters
complicated algorithms

Ranges in Boost

Ranges are pairs of iterators.
Memory overhead
Filters grow exponential in size

Ranges for the native C++

The range is the main abstraction not the iterator
It holds all informations
Concepts, asymmetric algorithms, sentinels have their own type.

Iterators and ranges for dynamical systems

Dynamical systems – Maps

$$x_{n+1} = f(x_n)$$

picture of logistic map

Dynamical systems – Maps

$$x_{n+1} = f(x_n)$$

Iota:

```
std::iota( first , last , 1 );
```

$$x_{n+1} = x_n + 1 \qquad (1, 2, 3, 4, \dots)$$

Generalized iota:

$$x_{n+1} = x_n + 2 \qquad (1, 3, 5, 7, \dots)$$

$$x_{n+1} = 2x_n \qquad (1, 2, 4, 8, \dots)$$

$$x_{n+1} = x_n^2 \qquad (1, 1, 1, 1, \dots)$$

Dynamical systems – ODEs

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x, t)$$

picture of lorenz system?

Numerical integration

Lagrange integration

Map iterator

Abstraction for $x_{n+1} = f(x_n)$
Problems:

1. x could be intrinsic state of the iterator, or the iterator iterates x.
2. Stop criterium, which is the end iterator

Map iterator

Implementation
two flavours, with count, with predicate
Naive implementation

Map iterator - applications

Generalized iota
Functional random number generators

Map iterator - pros and cons

Pro:

- Generalization of dynamical systems

Contra:

- end iterators needs to be unnecessary complex

Map range

Better implementation

Map iterators and ranges for the new standard

Sentinels

Iterators for GPUs algorithms

High-level libraries for GPUs

1. Thrust
2. VexCL
3. Boost.Compute
4. ViennaCL
5. Cuda-MTL

Thrust

STL-like library for Cuda
Design is based on iterators

Iterators in Thrust

`device_vector::iterator`
`host_vector::iterator`
special iterators
Algorithms

Implementation details of Thrust iterators

Special iterators for Thrust

zip iterator
transform iterator

Special problems - and solutions

Norm

Special problems - and solutions

Bucket sort

Solving an ensemble of low-dimensional ODEs

Lorenz example and ODEs

Conclusion

Outlook

References