Document Number: WG21/N0967R1
X3J16/96-0149R1
Date: 11 July 1996
Project: Programming Language C++
Reply to: Randy Smithey
smithey@roguewave.com

Relational Operators for Standard Library Classes

I. Summary of the Issue

In a library reflector message I pointed out that the new library namespace, rel_ops (adopted\001in Santa Cruz), prevents us from assuming that the global relational operators will be available for standard objects. The working paper, as it stands now, does assume these operators are available. The following specifies all locations in the library whererelational operators must be specified explicitly in order to remove this assumption.

II. Proposed Resolution

Change The Working Paper as follows:

Section 17.3.1.1 [Paragraph 9 Add the following to table 5 (Standard Template Operators):

operator!= (pair)
operator> (pair)
operator>=(pair)
operator<=(pair)
operator!= (deque)
operator> (deque)
operator>=(deque)
operator<=(deque)
operator!= (list)
operator> (list)
operator>=(list)
operator<=(list)
operator!= (vector)
operator> (vector)
operator>=(vector)
operator<=(vector)
operator!= (stack)
operator> (stack)
operator>=(stack)
operator<=(stack)
operator!= (queue)
operator> (queue)
operator>=(queue)
operator<=(queue)
operator!= (map)
operator> (map)
operator>=(map)
operator<=(map)
operator!= (multimap)
operator> (multimap)
operator>=(multimap)
operator<=(multimap)
operator!= (set)
operator> (set)
operator>=(set)
operator<=(set)
operator!= (multiset)
operator> (multiset)
operator>=(multiset)

```
operator<=(multiset)
operator!= (reverse_bidirectional_iterator)
operator!= (reverse_iterator)
operator> (reverse_iterator)
operator>=( reverse_iterator)
operator<=( reverse_iterator)
operator!= (istream_iterator)
operator!= (istreambuf_iterator)
```

Section 20.2 [lib.utility] and Section 20.2.2 [lib.pairs] Add the following declarations:

```
template <class T1, class T2>
bool operator!=(const pair<T1,T2>&, const pair<T1,T2>&);
template <class T1, class T2>
bool operator> (const pair<T1,T2>&, const pair<T1,T2>&);
template <class T1, class T2>
bool operator>=(const pair<T1,T2>&, const pair<T1,T2>&);
template <class T1, class T2>
bool operator<= (const pair<T1,T2>&, const pair<T1,T2>&);
```

Section 23.2 [lib.sequences] Add the following declarations in the appropriate synopsis':

```
template <class T, class Allocator>
bool operator!=(const deque<T,Allocator>& x,
                const deque<T,Allocator>& y);
template <class T, class Allocator>
bool operator> (const deque<T,Allocator>& x,
                const deque<T,Allocator>& y);
template <class T, class Allocator>
bool operator>=(const deque<T,Allocator>& x,
                const deque<T,Allocator>& y);
template <class T, class Allocator>
bool operator<=(const deque<T,Allocator>& x,
                const deque<T,Allocator>& y);

template <class T, class Allocator>
bool operator!=(const list<T,Allocator>& x,
                const list<T,Allocator>& y);
template <class T, class Allocator>
bool operator> (const list<T,Allocator>& x,
                const list<T,Allocator>& y);
template <class T, class Allocator>
bool operator>=(const list<T,Allocator>& x,
                const list<T,Allocator>& y);
template <class T, class Allocator>
bool operator<= (const list<T,Allocator>& x,
                 const list<T,Allocator>& y);

template <class T, class Allocator>
bool operator!=(const vector<T,Allocator>& x,
                const vector<T,Allocator>& y);
template <class T, class Allocator>
bool operator> (const vector<T,Allocator>& x,
                const vector<T,Allocator>& y);
template <class T, class Allocator>
bool operator>=(const vector<T,Allocator>& x,
                const vector<T,Allocator>& y);
template <class T, class Allocator>
bool operator<=(const vector<T,Allocator>& x,
                const vector<T,Allocator>& y);

template <class Allocator>
```

```
bool operator!=(const vector<bool,Allocator>& x,
                const vector<bool,Allocator>& y);
template <class Allocator>
bool operator> (const vector<bool,Allocator>& x,
                const vector<bool,Allocator>& y);
template <class Allocator>
bool operator>=(const vector<bool,Allocator>& x,
                const vector<bool,Allocator>& y);
template <class Allocator>
bool operator<=(const vector<bool,Allocator>& x,
                const vector<bool,Allocator>& y);

template <class T, class Container, class Allocator>
bool operator!=(const queue<T,Container,Allocator>& x,
                const queue<T,Container,Allocator>& y);
template <class T, class Container, class Allocator>
bool operator> (const queue<T,Container,Allocator>& x,
                const queue<T,Container,Allocator>& y);
template <class T, class Container, class Allocator>
bool operator>=(const queue<T,Container,Allocator>& x,
                const queue<T,Container,Allocator>& y);
template <class T, class Container, class Allocator>
bool operator<=(const queue<T,Container,Allocator>& x,
                const queue<T,Container,Allocator>& y);

template <class T, class Container, class Allocator>
bool operator!=(const stack<T,Container,Allocator>& x,
                const stack<T,Container,Allocator>& y);
template <class T, class Container, class Allocator>
bool operator> (const stack<T,Container,Allocator>& x,
                const stack<T,Container,Allocator>& y);
template <class T, class Container, class Allocator>
bool operator>=(const stack<T,Container,Allocator>& x,
                const stack<T,Container,Allocator>& y);
template <class T, class Container, class Allocator>
bool operator<=(const stack<T,Container,Allocator>& x,
                const stack<T,Container,Allocator>& y);
```

Section 23.3 [lib.associative] Add the following declarations to the synopsis:

```
template <class Key, class T, class Compare, class Allocator>
bool operator!=(const map<Key,T,Compare,Allocator>& x,
                const map<Key,T,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator> (const map<Key,T,Compare,Allocator>& x,
                const map<Key,T,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator>=(const map<Key,T,Compare,Allocator>& x,
                const map<Key,T,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator<=(const map<Key,T,Compare,Allocator>& x,
                const map<Key,T,Compare,Allocator>& y);

template <class Key, class T, class Compare, class Allocator>
bool operator!=(const multimap<Key,T,Compare,Allocator>& x,
                const multimap<Key,T,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator> (const multimap<Key,T,Compare,Allocator>& x,
                const multimap<Key,T,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator>=(const multimap<Key,T,Compare,Allocator>& x,
                const multimap<Key,T,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator<= (const multimap<Key,T,Compare,Allocator>& x,
                const multimap<Key,T,Compare,Allocator>& y);
```

```
template <class Key, class T, class Compare, class Allocator>
bool operator!=(const set<Key,Compare,Allocator>& x,
                const set<Key,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator> (const set<Key,Compare,Allocator>& x,
                const set<Key,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator>=(const set<Key,Compare,Allocator>& x,
                const set<Key,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator<= (const set<Key,Compare,Allocator>& x,
                 const set<Key,Compare,Allocator>& y);

template <class Key, class T, class Compare, class Allocator>
bool operator!=(const multiset<Key,Compare,Allocator>& x,
                const multiset<Key,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator> (const multiset<Key,Compare,Allocator>& x,
                const multiset<Key,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator>=(const multiset<Key,Compare,Allocator>& x,
                const multiset<Key,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
bool operator<=(const multiset<Key,Compare,Allocator>& x,
                const multiset<Key,Compare,Allocator>& y);
```

Section 24.2 [lib.iterator.synopsis] Add the following declarations to the synopsis:

```
template <class BidirectionalIterator, class T,
          class Reference, class Pointer, class Distance>
bool operator!=(const reverse_bidirectional_iterator
    <BidirectionalIterator,T,Reference,Pointer,Distance>& x,
    const reverse_bidirectional_iterator
    <BidirectionalIterator,T,Reference,Pointer,Distance> &y);

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator!=(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator>(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator>=(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator<=(const reverse_iterator
```

```
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

template <class T, class Distance>
bool operator!=(const istream_iterator<T,Distance>& x,
                const istream_iterator<T,Distance>& y);
```

Section 24.4.1.1 [lib.reverse.bidir.iter] Add the following declaration:

```
template <class BidirectionalIterator, class T,
          class Reference, class Pointer, class Distance>
bool operator!=(const reverse_bidirectional_iterator
    <BidirectionalIterator,T,Reference,Pointer,Distance>& x,
    const reverse_bidirectional_iterator
    <BidirectionalIterator,T,Reference,Pointer,Distance> &y);
```

Add the following working paper text after Section 24.4.1.2.7 [lib..reverse.bidir.iter.op==]:

24.4.1.2.? operator!= [lib.reverse.bidir.iter.op!=]

```
template <class BidirectionalIterator, class T,
          class Reference, class Pointer, class Distance>
bool operator!=(const reverse_bidirectional_iterator
    <BidirectionalIterator,T,Reference,Pointer,Distance>& x,
    const reverse_bidirectional_iterator
    <BidirectionalIterator,T,Reference,Pointer,Distance> &y);
```

Returns:

```
    !(x == y)
```

Section 24.4.1.3 [lib.reverse.iterator] Add the following working paper text:

```
template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator!=(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator>(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator>=(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator<=(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
```

```
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);
```

Add the following after 24.4.1.4.12 [lib.reverse.iter.op==]:

```
24.4.1.4.??  operator!=    [lib.reverse.iter.op!=]

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator!=(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

Returns:
  !(x == y)
```

Add the following after 24.4.1.4.13 [lib.reverse.iter.op<]:

```
24.4.1.4.??  operator>    [lib.reverse.iter.op>]

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator>(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

Returns:
  y < x


24.4.1.4.??  operator>=    [lib.reverse.iter.op>=]

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator>=(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

Returns:
  !(x < y)


24.4.1.4.??  operator<=    [lib.reverse.iter.op<=]

template <class RandomAccessIterator,
          class T, class Reference,
          class Pointer, class Distance>
bool operator<=(const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& x,
     const reverse_iterator
     <RandomAccessIterator,T,Reference,Pointer,Distance>& y);

Returns:
  !(y < x)
```

Section 24.5.1 Add the following declaration:

```
template <class T, class Distance>
bool operator!=(const istream_iterator<T,Distance>& x,
                const istream_iterator<T,Distance>& y);
```