Doubly-linked ring

Overview:
The task was to design a doubly-linked ring including an iterator with basic functionality. Also, to write a method that should combine two rings in a way specified by parameters into a new ring.

The Ring template class contains of structure Node which is the vital element of the ring. It allows to connect consecutive objects and store data. It's defined as a private member and has 4 elements - data of type Info, id of type Key and two pointers of type Node - next and prev, as we deal with a doubly linked data structure.
There is also another private member, pointer of type Node - any - which points arbitrary element in a ring and is a way of entering a ring.

In public part we find the definition of crucial member of whole structure - class Iterator, tool which helps us to navigate on the list. To provide basic functionality the iterator class contains several operators.

Methods:
`iterator()` iterator constructor
`~iterator()` iterator destructor
`iterator(const iterator& copy)` iterator copy constructor
`iterator& operator=(const iterator& copy)` assignment operator
`bool operator==(const iterator& compare)` comparison of iterators if they are the same
`bool operator!=(const iterator& compare)` comparison of iterators if they differ
`iterator operator-(const unsigned int howmany)` moves the posiion of iterator couterclockwise by one
`Iterator& operator--()` moves iterator counterclockwise by one position
`iterator operator+(const unsigned int howmany)` moves the position of iterator clockwise by howmany step
`iterator& operator++()` moves the position of iterator clockwise by one

`Ring()` constructor
`Ring(const Double_Linked_Ring<Key, Info>& Copy)` copy constructor
`~Ring()` destructor of the ring
`int sizeOfRing()` returns number of nodes in the ring
`bool addPrevious(const Key& key, const Info& info)` adds new element before element with given key (counterclockwise)

`bool addNext(const Key& key, const Info& info)` adds new element after recently added elt with given key (clockwise)
`bool addAfter(const Key& key, const Info& info, const Key& nodeafter)` adds new element after particular element with given key
`void remove(const Key& k, bool every)` remove elements with given key
`Ring<Key, Info>& operator=(const Ring<Key, Info>& copyRing)` assignment operator
`friend Ring<Key, Info>& operator-(const Ring<Key, Info>& firstring, const Ring<Key, Info>& secondring)` operator -
`friend Ring<Key, Info>& operator+(const Ring<Key, Info>& firstring, const Ring<Key, Info>& secondring)` operator +

Function produce:

```
Ring<Key, Info> produce(const Ring<Key, Info>& ring1, int start1,
          int steps1, bool dir1, const Ring<Key, Info>& ring2,
          int start2, int steps2, bool dir2, int num, bool
direction_out)
```

Function produce takes sources of two rings, the starting point, steps and direction. The function repeats instruction of copying blocks of elements, until the iterators have gone through all of the nodes in both rings. As a result we obtain a combination of desired data which is put to the new ring