

iterator.cm

```
/*  
  
    Copyright (c) 2012–2016 Seppo Laakko  
    http://sourceforge.net/projects/cmajor/  
  
    Distributed under the GNU General Public License, version 3 (GPLv3).  
    (See accompanying LICENSE.txt or http://www.gnu.org/licenses/gpl.html  
    )  
  
*/  
  
// Copyright (c) 1994  
// Hewlett–Packard Company  
// Copyright (c) 1996  
// Silicon Graphics Computer Systems, Inc.  
// Copyright (c) 2009 Alexander Stepanov and Paul McJones  
  
using System.Concepts;  
  
namespace System  
{  
    public class RandomAccessIter<T, R, P>  
    {  
        public typedef T ValueType;  
        public typedef R ReferenceType;  
        public typedef P PointerType;  
        private typedef RandomAccessIter<ValueType, ReferenceType,  
            PointerType> Self;  
  
        public nothrow inline RandomAccessIter(): ptr(null)  
        {  
        }  
        public nothrow inline explicit RandomAccessIter(PointerType ptr_)  
            : ptr(ptr_)  
        {  
        }  
        public nothrow inline Self& operator++()  
        {  
            #assert(ptr != null);  
            ++ptr;  
            return *this;  
        }  
        public nothrow inline Self& operator--()  
        {  
            #assert(ptr != null);  
            --ptr;  
            return *this;  
        }  
        public nothrow inline ReferenceType operator*() const
```

```

{
    #assert(ptr != null);
    return *ptr;
}
public nothrow inline PointerType operator->() const
{
    #assert(ptr != null);
    return ptr;
}
public nothrow inline ReferenceType operator[](int index) const
{
    #assert(ptr != null);
    return ptr[index];
}
public nothrow inline PointerType GetPtr() const
{
    return ptr;
}
private PointerType ptr;
}

public nothrow inline RandomAccessIter<T, R, P> operator+<T, R, P>(
    const RandomAccessIter<T, R, P>& it, int offset)
{
    #assert(it.GetPtr() != null);
    return RandomAccessIter<T, R, P>(it.GetPtr() + offset);
}

public nothrow inline RandomAccessIter<T, R, P> operator+<T, R, P>(
    int offset, const RandomAccessIter<T, R, P>& it)
{
    #assert(it.GetPtr() != null);
    return RandomAccessIter<T, R, P>(it.GetPtr() + offset);
}

public nothrow inline RandomAccessIter<T, R, P> operator-<T, R, P>(
    const RandomAccessIter<T, R, P>& it, int offset)
{
    #assert(it.GetPtr() != null);
    return RandomAccessIter<T, R, P>(it.GetPtr() - offset);
}

public nothrow inline int operator-<T, R, P>(const RandomAccessIter<T,
    R, P>& left, const RandomAccessIter<T, R, P>& right)
{
    #assert(left.GetPtr() == null && right.GetPtr() == null || left.
        GetPtr() != null && right.GetPtr() != null);
    if (left.GetPtr() == null && right.GetPtr() == null)
    {
        return 0;
    }
    return left.GetPtr() - right.GetPtr();
}
}

```

```

public nothrow inline bool operator==<T, R, P>(const RandomAccessIter
    <T, R, P>& left, const RandomAccessIter<T, R, P>& right)
{
    return left.GetPtr() == right.GetPtr();
}

public nothrow inline bool operator<<<T, R, P>(const RandomAccessIter<
    T, R, P>& left, const RandomAccessIter<T, R, P>& right)
{
    #assert(left.GetPtr() == null && right.GetPtr() == null || left.
        GetPtr() != null && right.GetPtr() != null);
    return left.GetPtr() < right.GetPtr();
}

public class BackInsertProxy<C> where C is BackInsertionSequence
{
    private typedef BackInsertProxy<C> Self;
    private typedef C.ValueType ValueType;

    public nothrow BackInsertProxy(): c(null)
    {
    }
    public nothrow explicit BackInsertProxy(C* c_): c(c_)
    {
    }
    public nothrow default BackInsertProxy(const Self&);
    public nothrow default void operator=(const Self&);
    public inline void operator=(const ValueType& value)
    {
        c->Add(value);
    }
    private C* c;
}

public class BackInsertIterator<C> where C is BackInsertionSequence
{
    private typedef BackInsertIterator<C> Self;
    private typedef BackInsertProxy<C> Proxy;
    public typedef Proxy ValueType;
    public typedef Proxy& ReferenceType;
    public typedef Proxy* PointerType;

    public nothrow BackInsertIterator(): proxy()
    {
    }
    public nothrow BackInsertIterator(C& c): proxy(&c)
    {
    }
    public nothrow ReferenceType operator*()
    {
        return proxy;
    }
}

```

```

    public nothrow PointerType operator->()
    {
        return &proxy;
    }
    public nothrow Self& operator++()
    {
        return *this;
    }
    private Proxy proxy;
}

public nothrow BackInsertIterator<C> BackInserter<C>(C& c) where C is
    BackInsertionSequence
{
    return BackInsertIterator<C>(c);
}

public class FrontInsertProxy<C> where C is FrontInsertionSequence
{
    private typedef FrontInsertProxy<C> Self;
    private typedef C.ValueType ValueType;

    public nothrow FrontInsertProxy(): c(null)
    {
    }
    public nothrow explicit FrontInsertProxy(C* c_): c(c_)
    {
    }
    public nothrow default FrontInsertProxy(const Self&);
    public nothrow default void operator=(const Self&);
    public inline void operator=(const ValueType& value)
    {
        c->InsertFront(value);
    }
    private C* c;
}

public class FrontInsertIterator<C> where C is FrontInsertionSequence
{
    private typedef FrontInsertIterator<C> Self;
    private typedef FrontInsertProxy<C> Proxy;
    public typedef Proxy ValueType;
    public typedef Proxy& ReferenceType;
    public typedef Proxy* PointerType;

    public nothrow FrontInsertIterator(): proxy()
    {
    }
    public nothrow FrontInsertIterator(C& c): proxy(&c)
    {
    }
    public nothrow ReferenceType operator*()
    {

```

```

        return proxy;
    }
    public nothrow PointerType operator->()
    {
        return &proxy;
    }
    public nothrow Self& operator++()
    {
        return *this;
    }
    private Proxy proxy;
}

public nothrow FrontInsertIterator<C> FrontInserter<C>(C& c) where C
    is FrontInsertionSequence
{
    return FrontInsertIterator<C>(c);
}

public class InsertProxy<C> where C is InsertionSequence
{
    private typedef InsertProxy<C> Self;
    private typedef C.Iterator Iterator;
    private typedef C.ValueType ValueType;

    public nothrow InsertProxy(): c(null), i()
    {
    }
    public nothrow InsertProxy(C* c_, Iterator i_): c(c_), i(i_)
    {
    }
    public nothrow default InsertProxy(const Self&);
    public nothrow default void operator=(const Self&);
    public nothrow default ~InsertProxy();
    public inline void operator=(const ValueType& value)
    {
        i = c->Insert(i, value);
        ++i;
    }
    private C* c;
    private Iterator i;
}

public class InsertIterator<C> where C is InsertionSequence
{
    private typedef InsertIterator<C> Self;
    private typedef InsertProxy<C> Proxy;
    public typedef Proxy ValueType;
    public typedef Proxy& ReferenceType;
    public typedef Proxy* PointerType;

    public nothrow InsertIterator(): proxy()
    {

```

```

    }
    public nothrow InsertIterator(C& c, C.Iterator i): proxy(&c, i)
    {
    }
    public nothrow ReferenceType operator*()
    {
        return proxy;
    }
    public nothrow PointerType operator->()
    {
        return &proxy;
    }
    public nothrow Self& operator++()
    {
        return *this;
    }
    private Proxy proxy;
}

public nothrow InsertIterator<C> Inserter<C, I>(C& c, I i) where C is
    InsertionSequence and I is C.Iterator
{
    return InsertIterator<C>(c, i);
}
}

```