map.cm

```
/*
    ════════════════════════════════════════════════════════════════════

        Copyright (c) 2012−2015 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;
using System.Concepts;

namespace System.Collections
{
    public class Map<Key, Value, KeyCompare = Less<Key>> where Key is
        Semiregular and Value is Semiregular and KeyCompare is Relation
        and KeyCompare.Domain is Key
    {
        public typedef Key KeyType;
        public typedef Value MappedType;
        public typedef KeyCompare Compare;
        public typedef Pair<KeyType, MappedType> ValueType;
        private typedef Map<KeyType, MappedType, KeyCompare> Self;
        private typedef RedBlackTree<KeyType, ValueType, SelectFirst<
            KeyType, MappedType>, KeyCompare> TreeType;
        public typedef TreeType.ConstIterator ConstIterator;
        public typedef TreeType.Iterator Iterator;

        public nothrow inline Iterator Begin()
        {
            return tree.Begin();
        }
        public nothrow inline ConstIterator Begin() const
        {
            return tree.CBegin();
        }
        public nothrow inline ConstIterator CBegin() const
        {
            return tree.CBegin();
        }
        public nothrow inline Iterator End()
```

```cminor
{
    return tree.End();
}
public nothrow inline ConstIterator End() const
{
    return tree.CEnd();
}
public nothrow inline ConstIterator CEnd() const
{
    return tree.CEnd();
}
public nothrow inline int Count() const
{
    return tree.Count();
}
public nothrow inline bool IsEmpty() const
{
    return tree.IsEmpty();
}
public nothrow inline void Clear()
{
    tree.Clear();
}
public nothrow inline Iterator Find(const KeyType& key)
{
    return tree.Find(key);
}
public nothrow ConstIterator Find(const KeyType& key) const
{
    return tree.CFind(key);
}
public nothrow ConstIterator CFind(const KeyType& key) const
{
    return tree.CFind(key);
}
public MappedType& operator[](const KeyType& key)
{
    ValueType valueType(key, MappedType());
    Pair<Iterator, bool> ib = Insert(valueType);
    Iterator i = ib.first;
    return i->second;
}
public inline Pair<Iterator, bool> Insert(const ValueType& value)
    where ValueType is Copyable
{
    return tree.Insert(value);
}
public nothrow inline bool Remove(const KeyType& key)
{
    return tree.Remove(key);
}
public nothrow inline void Remove(Iterator pos)
{
```

```
                tree.Remove(pos);
        }
        private TreeType tree;
}

public nothrow bool operator==<Key, Value, KeyCompare>(const Map<Key,
        Value, KeyCompare>& left, const Map<Key, Value, KeyCompare>&
    right)
        where Key is Regular and Value is Regular and KeyCompare is
            Relation and KeyCompare.Domain is Key
{
        if (left.Count() != right.Count())
        {
            return false;
        }
        Map<Key, Value, KeyCompare>.ConstIterator first1 = left.CBegin();
        Map<Key, Value, KeyCompare>.ConstIterator last1 = left.CEnd();
        Map<Key, Value, KeyCompare>.ConstIterator first2 = right.CBegin()
            ;
        Map<Key, Value, KeyCompare>.ConstIterator last2 = right.CEnd();
        while (first1 != last1 && first2 != last2)
        {
            if (first1->first != first2->first || first1->second !=
                first2->second)
            {
                return false;
            }
            ++first1;
            ++first2;
        }
        return first1 == last1 && first2 == last2;
}

public nothrow bool operator<<Key, Value, KeyCompare>(const Map<Key,
        Value, KeyCompare>& left, const Map<Key, Value, KeyCompare>& right
    )
        where Key is Semiregular and Value is TotallyOrdered and
            KeyCompare is Relation and KeyCompare.Domain is Key
{
        Map<Key, Value, KeyCompare>.ConstIterator first1 = left.CBegin();
        Map<Key, Value, KeyCompare>.ConstIterator last1 = left.CEnd();
        Map<Key, Value, KeyCompare>.ConstIterator first2 = right.CBegin()
            ;
        Map<Key, Value, KeyCompare>.ConstIterator last2 = right.CEnd();
        KeyCompare less;
        while (first1 != last1 && first2 != last2)
        {
            if (less(first1->first, first2->first))
            {
                return true;
            }
            else if (less(first2->first, first1->first))
            {
```

```
                    return false;
            }
            else if ( first1 −>second < first2 −>second)
            {
                    return true;
            }
            else if ( first2 −>second < first1 −>second)
            {
                    return false;
            }
            ++first1;
            ++first2;
        }
        return first1 == last1 && first2 != last2;
    }
}
```