bitset.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;

namespace System.Collections
{
    public class BitSet
    {
        public nothrow BitSet(): numBits(0), bits()
        {
            try
            {
                bits.Add(0u);
            }
            catch (const Exception& ex)
            {
                exit(1);
            }
        }
        public BitSet(int numBits_): numBits(0), bits()
        {
            Resize(numBits_);
        }
        public BitSet(const string& bits_): numBits(0), bits()
        {
            Resize(bits_.Length());
            for (int i = 0; i < numBits; ++i)
            {
                if (bits_[i] == '1')
                {
                    Set(i);
                }
            }
        }
```

```
public default BitSet(const BitSet&);
public default void operator=(const BitSet&);
public default nothrow BitSet(BitSet&&);
public default nothrow void operator=(BitSet&&);
public default ~BitSet();
public nothrow void Clear()
{
    numBits = 0;
    bits.Clear();
}
public nothrow inline int Count() const
{
    return numBits;
}
public void Resize(int numBits_)
{
    numBits = numBits_;
    #assert(numBits >= 0);
    if (numBits > 0)
    {
        bits.Resize(1 + NumFullBlocks());
    }
    else
    {
        bits.Resize(1);
    }
}
public void Set()
{
    int n = bits.Count();
    for (int i = 0; i < n; ++i)
    {
        bits[i] = allOne;
    }
}
public void Reset()
{
    int n = bits.Count();
    for (int i = 0; i < n; ++i)
    {
        bits[i] = 0u;
    }
}
public void Set(int pos)
{
    #assert(pos >= 0 && pos < numBits);
    ulong& b = bits[pos / blockSize];
    b = b | (1u << (cast<ulong>(pos) & blockMask));
}
public void Reset(int pos)
{
    #assert(pos >= 0 && pos < numBits);
    ulong& b = bits[pos / blockSize];
```

```
        b = b & ~(1u << (cast<ulong>(pos) & blockMask));
    }
    public void Set(int pos, bool bit)
    {
        if (bit)
        {
            Set(pos);
        }
        else
        {
            Reset(pos);
        }
    }
    public void Flip()
    {
        int n = bits.Count();
        for (int i = 0; i < n; ++i)
        {
            bits[i] = ~(bits[i]);
        }
    }
    public void Flip(int pos)
    {
        #assert(pos >= 0 && pos < numBits);
        ulong& b = bits[pos / blockSize];
        b = b ^ (1u << (cast<ulong>(pos) & blockMask));
    }
    public bool operator[](int index) const
    {
        return Test(index);
    }
    public bool Test(int pos) const
    {
        #assert(pos >= 0 && pos < numBits);
        ulong b = bits[pos / blockSize];
        return (b & (1u << (cast<ulong>(pos) & blockMask))) != 0u;
    }
    public bool All() const
    {
        int n = NumFullBlocks();
        for (int i = 0; i < n; ++i)
        {
            if (bits[i] != allOne)
            {
                return false;
            }
        }
        for (int i = LastBlockStartIndex(); i < numBits; ++i)
        {
            if (!Test(i))
            {
                return false;
            }
        }
        return true;
    }
    public bool Any() const
    {
        int n = NumFullBlocks();
        for (int i = 0; i < n; ++i)
        {
            if (bits[i] != 0u)
            {
                return true;
            }
        }
        for (int i = LastBlockStartIndex(); i < numBits; ++i)
        {
            if (Test(i))
            {
                return true;
            }
        }
        return false;
    }
    public bool None() const
    {
        int n = NumFullBlocks();
        for (int i = 0; i < n; ++i)
        {
            if (bits[i] != 0u)
            {
                return false;
            }
        }
        for (int i = LastBlockStartIndex(); i < numBits; ++i)
        {
            if (Test(i))
            {
                return false;
            }
        }
        return true;
    }
    public bool operator==(const BitSet& that) const
    {
        if (numBits != that.numBits)
        {
            return false;
        }
        const List<ulong>& thatBits = that.bits;
        int n = NumFullBlocks();
        for (int i = 0; i < n; ++i)
        {
            if (bits[i] != thatBits[i])
            {
                    return false;
                }
            }
            for (int i = LastBlockStartIndex(); i < numBits; ++i)
            {
                bool test = Test(i);
                bool thatTest = that.Test(i);
                if (test != thatTest)
                {
                    return false;
                }
            }
            return true;
        }
        public string ToString() const
        {
            string s;
            s.Reserve(numBits);
            for (int i = 0; i < numBits; ++i)
            {
                if (Test(i))
                {
                    s.Append('1');
                }
                else
                {
                    s.Append('0');
                }
            }
            return s;
        }
        private inline nothrow int NumFullBlocks() const
        {
            if (numBits == 0)
            {
                return 0;
            }
            return (numBits - 1) / blockSize;
        }
        private inline nothrow int LastBlockStartIndex() const
        {
            return NumFullBlocks() * blockSize;
        }
        private const int blockSize = 64;
        private const ulong blockMask = 63u;
        private const ulong allOne = 0xFFFFFFFFFFFFFFFFu;
        private int numBits;
        private List<ulong> bits;
    }
}
```