KwCombinatorics

KwCombinatorics: K-Combination, Permutation, Cartesian Product classes in C#

Combinatorics library: k-combinations, multi-combinations, permutations, k-permutations and cartesian products with lexicographical ordering and random access.

The KwCombinatorics library consists of four classes that provide four different ways of generating ordered lists of combinations of numbers. These combinations may be used to permute (rearrange) other lists of objects. Combinatorics are useful for software testing by allowing the generation of various types of possible combinations of input. Other applications include mathematical problems, games of chance, and cryptography.

The classes provided are:

- **Combination**: An ascending sequence of non-repeating picks from a supplied number of choices.
- **Multicombination**: An ascending sequence of repeating picks from a supplied number of choices.
- **Permutation**: An arrangement of all or a subset of values from a supplied range.
- **Product**: A join of values from a supplied array of ranges.

Two key features this library provides are *unranking* and *ranking* for every combinatoric. Unranking is the ability to quickly retrieve any row in the combinatoric's lexicographically ordered table by setting its **Rank** property, or by supplying the rank to a constructor. Ranking is where an array of integers is supplied to a constructor. The Rank property will then contain its place in the ordered table. All combinatorics include a **GetRows()** method that generates all its rows ordered by Rank.

In addition to lexicographical ordering, the **Permutation** class includes an ordering where adjacent rows differ by only a single swap of adjacent elements. Support for this ordering is provided with the **PlainRank** property and the **GetRowsOfPlainChanges()** iterator.

The default appearance of a combinatoric row is a list of integers (starting at zero) enclosed in braces. This appearance may be tailored three ways:

- Map each element to a different value using the combinatoric's default enumerator or its indexer.
- Use the Permute method and output the rearranged values.
- Subclass of the combinatoric and override ToString().

Below are complete example programs. Most of the techniques shown may be applied to any combinatoric in the library.

Here is an example of the **Multicombination** class:

```
using Kw.Combinatorics;
using System;

namespace Kw.CombinatoricsExamples
{
    class McExample01
    {
        static void Main ()
        {
            var mc = new Multicombination (choices:4, picks:3);

            Console.WriteLine ("n={0}, k={1}:\n", mc.Choices, mc.Picks);

            foreach (var row in mc.GetRows())
                Console.WriteLine ("{0,2}:  {1}", row.Rank, row);
        }

        /* Output:

        n=4, k=3:

         0:  { 0, 0, 0 }
         1:  { 0, 0, 1 }
         2:  { 0, 0, 2 }
         3:  { 0, 0, 3 }
         4:  { 0, 1, 1 }
         5:  { 0, 1, 2 }
         6:  { 0, 1, 3 }
         7:  { 0, 2, 2 }
         8:  { 0, 2, 3 }
         9:  { 0, 3, 3 }
        10:  { 1, 1, 1 }
        11:  { 1, 1, 2 }
```

```
        12:  { 1, 1, 3 }
        13:  { 1, 2, 2 }
        14:  { 1, 2, 3 }
        15:  { 1, 3, 3 }
        16:  { 2, 2, 2 }
        17:  { 2, 2, 3 }
        18:  { 2, 3, 3 }
        19:  { 3, 3, 3 }

        */
    }
}
```

Here is an example of the **Combination** class:

```
using Kw.Combinatorics;
using System;
using System.Linq;

namespace Kw.CombinatoricsExamples
{
    // Subclassing is one way to get user-friendly output:
    public class Fruit : Combination
    {
        string[] choices;

        public Fruit (string[] list) : base (list.Length)
        { this.choices = list; }

        public override string ToString ()
        { return String.Join (" ", from ei in this select choices[ei]); }
    }

    class CnExample02
    {
        static void Main ()
        {
            string[] names = { "apple", "banana", "cherry", "durian" };

            foreach (var basket in new Fruit (names).GetRowsForAllPicks())
                Console.WriteLine (basket);
        }

        /* Output:

        apple
        banana
        cherry
        durian
        apple banana
        apple cherry
        apple durian
        banana cherry
        banana durian
        cherry durian
        apple banana cherry
        apple banana durian
        apple cherry durian
        banana cherry durian
        apple banana cherry durian

        */
    }
}
```

Here is an example of the **Permutation** class:

```
using Kw.Combinatorics;
using System;
using System.Collections.Generic;

namespace Kw.CombinatoricsExamples
{
    public class Furniture
    {
        private string name;
        public Furniture (string newName) { name = newName; }
        public override string ToString () { return name; }
    }

    public class Fruit
    {
        private string name;
        public Fruit (string newName) { name = newName; }
        public override string ToString () { return name; }
    }

    class PnExample03
    {
        static void Main ()
        {
            var things = new List<object>
```

```
            {
                new Fruit ("apple"),
                new Furniture ("bench"),
                new Furniture ("chair")
            };

            // Use permutations to get rearrangements of other objects:

            foreach (var row in new Permutation (things.Count).GetRows())
            {
                foreach (var mix in Permutation.Permute (row, things))
                    Console.Write ("{0} ", mix);
                Console.WriteLine ();
            }
        }

        /* Output:

        apple bench chair
        apple chair bench
        bench apple chair
        bench chair apple
        chair apple bench
        chair bench apple

        */
    }
}
```

Here is another example of the **Permutation** class:

```
using Kw.Combinatorics;
using System;

namespace Kw.CombinatoricsExamples
{
    class PnExample04
    {
        static void Main ()
        {
            // Create a permutation of the supplied rank:

            var pn = new Permutation (choices:8, picks:8, rank: 119);
            Console.WriteLine ("{0}  rank={1}", pn, pn.Rank);

            // Assign -1 to get the last rank, treat row like a string:

            pn.Rank = -1;
            string text = pn.ToString() + "  last=" + pn.Rank;
            Console.WriteLine ("\n" + text);

            // Rank will always stay in bounds:

            pn.Rank = pn.Rank + 1;
            Console.WriteLine ("\n{0}  rank={1}", pn, pn.Rank);

            // Create a permutation from the supplied sequence:

            pn = new Permutation (new int[] { 6, 4, 2, 0, 7, 5, 3, 1 });
            Console.WriteLine ("\n{0}  rank={1}", pn, pn.Rank);
        }

        /* Output:

        { 0, 1, 2, 7, 6, 5, 4, 3 }  rank=119

        { 7, 6, 5, 4, 3, 2, 1, 0 }  last=40319

        { 0, 1, 2, 3, 4, 5, 6, 7 }  rank=0

        { 6, 4, 2, 0, 7, 5, 3, 1 }  rank=33383

        */
    }
}
```

Here is another example of the **Combination** class:

```
using Kw.Combinatorics;
using System;
using System.Linq;

namespace Kw.CombinatoricsExamples
{
    class CnExample05
    {
        static void Main ()
        {
            // Create a k-combination:

            var cn = new Combination (choices:10, picks:7, rank:110);
            Console.WriteLine ("n={0}, k={1}, rank={2}:\n", cn.Choices, cn.Picks, cn.Rank);
```

```
            // Access elements using the default enumerator:

            var text = String.Concat (cn.Select (ei => (char) ('A' + ei)));
            Console.WriteLine (text + "\n");

            // Access elements using the indexer:

            for (int i = 0; i < cn.Picks; ++i)
                Console.WriteLine ("Element at {0} is {1}", i, cn[i]);
        }

        /* Output:

        n=10, k=7, rank=110:

        BDFGHIJ

        Element at 0 is 1
        Element at 1 is 3
        Element at 2 is 5
        Element at 3 is 6
        Element at 4 is 7
        Element at 5 is 8
        Element at 6 is 9

        */
    }
}
```

Here is another example of the **Permutation** class:

```
using Kw.Combinatorics;
using System;

namespace Kw.CombinatoricsExamples
{
    class PnExample06
    {
        static void Main ()
        {
            var pn = new Permutation (choices:4, picks:3);

            Console.WriteLine ("n={0}, k={1}, count={2}:\n",
                               pn.Choices, pn.Picks, pn.RowCount);

            foreach (var row in pn.GetRows())
                Console.WriteLine ("{0,2}:  {1}", row.Rank, row);
        }

        /* Output:

        n=4, k=3, count=24:

         0:  { 0, 1, 2 }
         1:  { 0, 1, 3 }
         2:  { 0, 2, 1 }
         3:  { 0, 2, 3 }
         4:  { 0, 3, 1 }
         5:  { 0, 3, 2 }
         6:  { 1, 0, 2 }
         7:  { 1, 0, 3 }
         8:  { 1, 2, 0 }
         9:  { 1, 2, 3 }
        10:  { 1, 3, 0 }
        11:  { 1, 3, 2 }
        12:  { 2, 0, 1 }
        13:  { 2, 0, 3 }
        14:  { 2, 1, 0 }
        15:  { 2, 1, 3 }
        16:  { 2, 3, 0 }
        17:  { 2, 3, 1 }
        18:  { 3, 0, 1 }
        19:  { 3, 0, 2 }
        20:  { 3, 1, 0 }
        21:  { 3, 1, 2 }
        22:  { 3, 2, 0 }
        23:  { 3, 2, 1 }

        */
    }
}
```

Here is another example of the **Permutation** class:

```
using Kw.Combinatorics;
using System;

namespace Kw.CombinatoricsExamples
{
    class PnExample07
    {
```

```
        static void Main ()
        {
            var pn = new Permutation (4);

            Console.WriteLine ("Plain changes, n={0}:\n", pn.Choices);

            foreach (var row in pn.GetRowsOfPlainChanges())
                Console.WriteLine ("{0,2}: {1}  Swaps={2}, Rank={3}",
                                   row.PlainRank, row, row.Swaps, row.Rank);
        }

        /* Output:

        Plain changes, n=4:

         0: { 0, 1, 2, 3 }  Swaps=0, Rank=0
         1: { 0, 1, 3, 2 }  Swaps=1, Rank=1
         2: { 0, 3, 1, 2 }  Swaps=2, Rank=4
         3: { 3, 0, 1, 2 }  Swaps=3, Rank=18
         4: { 3, 0, 2, 1 }  Swaps=2, Rank=19
         5: { 0, 3, 2, 1 }  Swaps=1, Rank=5
         6: { 0, 2, 3, 1 }  Swaps=2, Rank=3
         7: { 0, 2, 1, 3 }  Swaps=1, Rank=2
         8: { 2, 0, 1, 3 }  Swaps=2, Rank=12
         9: { 2, 0, 3, 1 }  Swaps=3, Rank=13
        10: { 2, 3, 0, 1 }  Swaps=2, Rank=16
        11: { 3, 2, 0, 1 }  Swaps=3, Rank=22
        12: { 3, 2, 1, 0 }  Swaps=2, Rank=23
        13: { 2, 3, 1, 0 }  Swaps=3, Rank=17
        14: { 2, 1, 3, 0 }  Swaps=2, Rank=15
        15: { 2, 1, 0, 3 }  Swaps=1, Rank=14
        16: { 1, 2, 0, 3 }  Swaps=2, Rank=8
        17: { 1, 2, 3, 0 }  Swaps=3, Rank=9
        18: { 1, 3, 2, 0 }  Swaps=2, Rank=11
        19: { 3, 1, 2, 0 }  Swaps=1, Rank=21
        20: { 3, 1, 0, 2 }  Swaps=2, Rank=20
        21: { 1, 3, 0, 2 }  Swaps=3, Rank=10
        22: { 1, 0, 3, 2 }  Swaps=2, Rank=7
        23: { 1, 0, 2, 3 }  Swaps=1, Rank=6

        */
    }
}
```