A PERMUTATION TYPE FOR JULIA

This is documentation for a `Permutation` data type for Julia. We only consider permutations of the set $\{1, 2, \ldots, n\}$ where n is a positive integer.

The `Permutation` type is defined in the file `permutation.jl`. This class is embedded in a module named `permutation`. So, to get started in Julia, one needs to give these two commands:

```
include("permutation.jl")
using permutation
```

A `Permutation` object is created from a one-dimensional array of integers containing each of the values 1 through n exactly once, like this:

```
julia> a = [4,1,3,2,6,5];
julia> p = Permutation(a)
(1,4,2)(3)(5,6)
```

Observe that `Permutation`s are printed in disjoint cycle format.

The number of elements in a `Permutation` is determined with the `length` function:

```
julia> length(p)
6
```

We can convert a `Permutation` to an array (the array used to construct the `Permutation` in the first place) or represent as a two-row matrix as follows:

```
julia> array(p)
6-element Array{Int64,1}:
 4
 1
 3
 2
 6
 5
julia> two_row(p)
2x6 Array{Int64,2}:
 1  2  3  4  5  6
 4  1  3  2  6  5
```

Evaluation of a `Permutation` on a particular element is performed with square-bracket notation:

```
julia> p[2]
1
```

Of course, bad things happen if an inappropriate element is given:

```
julia> p[7]
ERROR: BoundsError()
```

```
 in getindex at ....
```

Note that Permutations are immutable and the value at an index cannot be changed.

Composition is denoted by `*`:

```
julia> q = Permutation([1,6,2,3,4,5])
(1)(2,6,5,4,3)
julia> p*q
(1,4,3)(2,5)(6)
julia> q*p
(1,3,2)(4,6)(5)
```

The inverse is computed with `inv`:

```
julia> q = inv(p)
(1,2,4)(3)(5,6)
julia> p*q
(1)(2)(3)(4)(5)(6)
```

Repeated multiplication (exponentiation) is calculated with `^` like this: `p^n`. The exponent, `n` can be negative.

The *order* of a permutation π is the smallest positive integer n such that π^n is the identity. This can be calculated with `order(p)`.

The parity of a `Permutation` can be determined using `parity` which returns 1 for an odd permutation and 0 for an even permutation:

```
julia> parity(p)
1
julia> parity(p*p)
0
```

To get the cycle structure of a `Permutation` (not as a string), use `cycles`:

```
julia> cycles(p)
3-element Array{Array{Int64,1},1}:
 [1,4,2]
 [3]
 [5,6]
```

For convenience, identity and random permutations can be constructed like this:

```
julia> IdentityPermutation(10)
(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)
julia> RandomPermutation(10)
(1,7,6,10,3,2,8,4)(5,9)
```

In addition, we can use `Permutation(n,k)` to create the k^{th} permutation of the set $\{1, 2, \ldots, n\}$. Of course, this requires $1 \le k \le n!$.

```
julia> Permutation(6,701)
(1,6,3)(2,5)(4)
```

The function `matrix` converts a permutation $\pi \in S_n$ to an $n \times n$-matrix whose i, j-entry is 1 when $j = \pi(i)$ and 0 otherwise. By default, this creates a matrix with full storage; to get a sparse result use `matrix(p,true)`.

```
julia> p = RandomPermutation(6)
(1,2,6,4)(3,5)
julia> matrix(p)
6x6 Array{Int64,2}:
 0  1  0  0  0  0
 0  0  0  0  0  1
 0  0  0  0  1  0
 1  0  0  0  0  0
 0  0  1  0  0  0
 0  0  0  1  0  0
```

Ed Scheinerman, Johns Hopkins University
ers@jhu.edu   2014:07:24:10:37