```
/*****************************************************************************************

        Diffusion Along a Circle
        (Canonical Ensemble)
        Eigensystem Approach

        Written by:  Christian Bracher
        Version   :  0.28
        Date      :  November 26, 2003

*****************************************************************************************/

DESCRIPTION OF THE Pattern CLASS

Overview:  The class Pattern contains a template good for all atomic patterns on up to 31
           sites, together with evaluating and/or manipulating member functions.  All
           functions are public.

           Pattern uses Pattern::Code, a 32-bit variable of type long, to store a single
           atomic pattern as a binary code.  The lowest bit indicates the occurrence of an
           atom at site 0, the next bit the occupancy of site 1, etc., and bit (p-1)
           describes the state of the site (p-1), where p, a global variable, is the
           number of sites.  The bits p through 31 are effectively ignored.

           Although the sites logically form a loop, the lowest bit is always designated
           as the leftmost site in string representations.  These representations use a
           sequence of p characters consisting of the symbols "o" (atom occupies site)
           and "." (site is empty).  Example:

                 Pattern sequence:  oo.o..o  (p=7)    Code:  75 (dec)


SOME SPECIAL DEFINITIONS:

Atom number ........ The number k of occupied sites in the pattern.

Rotation ........... A shift of the entire atomic pattern on the loop.  Elementary
                     rotations shift the pattern by one site (counter)clockwise.

Pattern sum ........ This quantity is defined as the sum over numbers of occupied sites,
                     modulo the number of sites p.  If p is prime, the following
                     statements hold:

                     * The set of all patterns is composed of subsets containing p
Congruence .........   congruent patterns each that may be transformed into each other
                       by rotations.

                     * Assume that two patterns are congruent; then, they are identical
                       if and only if their pattern sums are equal.

                     * In each subset, every value of the pattern sum (0,1,...,p-1) is
                       taken on exactly once.

                     Hence, the pattern sum may be used to select one member of each
                     subset to represent all essentially different atomic configurations.
                     For this purpose, we pick all patterns with vanishing pattern sum,
Primitives ......... the primitive patterns.  The number of elementary clockwise rotations
                     required to create a given pattern from its primitive counterpart is
Shift .............. called its shift.
```

Example of a congruent subset (p=7):

```
Sequence: oo.o..o   Code: 75 (dec)   Pattern sum: 3   Shift: 6
          ooo.o..         23 (dec)                 0         0
          .ooo.o.         46 (dec)                 4         1
          ..ooo.o         92 (dec)                 1         2
          o..ooo.         57 (dec)                 5         3
          .o..ooo        114 (dec)                 2         4
          o.o..oo        101 (dec)                 6         5
```

Here, "ooo.o.." represents the primitive member of the set.

Mirror symmetry .... Mirror reflections of the loop are defined to leave site 0 intact, while all other sites swap their places according to the rule m <--> p-m. Example:

```
Pattern sequence:  oo.o..o  (p=7)    Code:  75 (dec)
After reflection:  oo..o.o           Code:  83 (dec)
```

Palindromics ..... The reflection operation decomposes the various patterns into pairs of mirror images, except for patterns that equal their own mirror image. These are called palindromic. The images of primitive patterns are again primitive, and all palindromic patterns are also primitive.

Jumps ..... A shift of a single atom by a single position, while the remaining pattern is not affected. Atomic jumps are the elementary events of the diffusion process. Only single occupancy of sites is allowed: thus, jumps must end up on an neighbouring empty site. DIFF recognizes four different types of jumps, occurring with individual

Jump rates ..... rates A, B, C, D:

* Type A ... A jump of an atom to the site on its left or right, where the atom has neither a direct neighbour before the jump takes place, nor afterwards. Schematically:

  ```
  xxx.o..xxx  -->  xxx..o.xxx  (clockwise)
  xxx..o.xxx  -->  xxx.o..xxx  (counterclockwise)
  ```

  (The occupation of the sites marked by xxx does not influence the jump rate.)

* Type B ... A jump of an atom to the site on its left or right, where the atom has an initial direct neighbour, but not a final neighbouring atom. Schematically:

  ```
  xxxoo..xxx  -->  xxxo.o.xxx  (clockwise)
  xxx..ooxxx  -->  xxx.o.oxxx  (counterclockwise)
  ```

* Type C ... A jump of an atom to the site on its left or right, where the atom has no initial direct neighbour, but ends up with a final neighbouring atom. Schematically:

  ```
  xxx.o.oxxx  -->  xxx..ooxxx  (clockwise)
  xxxo.o.xxx  -->  xxxoo..xxx  (counterclockwise)
  ```

* Type D ... A jump of an atom to the site on its left or right, where the atom has both an initial and a final direct neighbour. Schematically:

  ```
  xxxoo.oxxx  -->  xxxo.ooxxx  (clockwise)
  xxxo.ooxxx  -->  xxxoo.oxxx  (counterclockwise)
  ```

Atom - hole equivalence ..... Note that for a jump of type B (C), the reverse jump will be of type C (B), while for jumps of type A and D the reverse jump will belong to the same class. We also mention that these jump patterns can also be interpreted as jumps of holes in the opposite direction.

Type D then refers to the jumps of holes without initial and final neighbours, and is the equivalent of a type A jump for atoms, and vice versa. The jumps of type B and C are equivalent to themselves.

Collapsing number .. The number of contiguous blocks of atoms in the pattern, less one. In other words, the number of atomic "block mergers" that can be achieved by a sequence of atomic jumps. These jumps change the collapsing number by at most one, according to the following rule:

Jump Type:	A	Collapsing number:	Invariant
	B		Increment
	C		Decrement
	D		Invariant

It is not affected by rotations or mirror reflections of the entire pattern. Examples:

Sequence:	Code:	Collapsing number:
oo.o..o	75 (dec)	2
oooo...	15 (dec)	0
o.o.o.o	85 (dec)	2
.oo.oo.	54 (dec)	1

(Note that the pattern is located on a loop - the clockwise neighbour of the last token is the first token.)

Indefinite patterns A string of p characters consisting of the symbols "o" (site is occupied by atom), "." (site is empty), and "x" (contents of site unspecified). A given atomic pattern is said to match the indefinite pattern if both patterns agree in the contents of the specified sites. Example:

Sequence: oo.o..o	Matches:	
	oo.xxxx ?	Yes
	xoxxx.o ?	Yes
	.oxxxxx ?	No

DETAILED DESCRIPTION OF THE MEMBERS OF THE Pattern CLASS

(1) CONSTRUCTORS

Pattern::Pattern()

Default constructor for an object of type Pattern. Initializes Pattern::Code to zero, which is equivalent to completely empty sites.

Pattern::Pattern(long Value)

Construct an object of type Pattern and assign the binary pattern code Value to Pattern::Code.

Pattern::Pattern(string Sequence)

Construct an object of type Pattern from its graphical representation Sequence. Sequence is a string of length p (number of sites), composed of the characters "o" (atom) and "." (empty site). The presence of other tokens or strings of a different size will lead to an error message; program execution is aborted.

(2) ASSIGNMENT OPERATORS

Pattern::operator=(long Value)

Assign the binary code Value to an already defined object of type Pattern. Formally, Pattern::Code = Value.

Pattern::operator=(string Sequence)

Assign the pattern encoded in the graphical representation Sequence to an already existing object of type Pattern. Sequence is a string of length p (number of sites), composed of the characters "o" (atom) and "." (empty site). The presence of other tokens or strings of a different size will lead to an error message; program execution is aborted.

(3) ELEMENTARY ROTATION OPERATORS

Pattern::operator++()

Rotate the entire encoded pattern by one site in clockwise direction.

Pattern::operator--()

Rotate the entire encoded pattern by one site in counterclockwise direction.

(4) PATTERN EVALUATION FUNCTIONS

int Pattern::AtomNum()

Count and return the number of occupied sites in the Pattern object.

int Pattern::PatternSum()

Evaluate and return the pattern sum of the encoded pattern (see above).

int Pattern::CollapsingNumber()

This function returns the number of contiguous blocks of atoms (or holes) in the pattern, less one. It also represents the number of extra number of jumps of type C, as compared to jumps of type B, that is required to collapse the pattern into a single contiguous block of atoms.

int Pattern::Element(int nu)

Extract the position of the nu.th atom in the pattern, starting from the "lowest" site 0.

bool IncrementSiteBlocked(int nu)

Examine whether the right (clockwise) neighbouring site of the nu.th atom in the pattern is occupied (blocked) or empty.

bool DecrementSiteBlocked(int nu)

Examine whether the left (counterclockwise) neighbouring site of the nu.th atom in the pattern is occupied (blocked) or empty.

(5) PATTERN MANIPULATION FUNCTIONS

int Pattern::Primitive()

Replace the encoded pattern by its congruent primitive pattern, and return the clockwise shift of the original pattern with respect to its primitive counterpart.

int Pattern::Increment(int nu)

Effect clockwise jump of the nu.th atom in the pattern to its neighbouring site. The function returns a code for the type of jump:

Return Code:	Jump Type:
0	A
1	B
2	C
3	D

int Pattern::Decrement(int nu)

Effect counterclockwise jump of the nu.th atom in the pattern to its neighbouring site. The function returns a code for the type of jump:

Return Code:	Jump Type:
0	A
1	B
2	C
3	D

(6) MIRROR REFLECTION FUNCTIONS

Pattern::Mirror()

Replace the pattern encoded in the object by its mirror image.

bool Pattern::IsPalindromic()

Examine whether the pattern is palindromic, i.e., whether it equals its own mirror image.

(7) PATTERN MATCHING FUNCTIONS

bool Pattern::Match(string Comparison)

Examine whether the indefinite pattern encoded in the string Comparison matches the atomic pattern stored in the Pattern object. Comparison must be a string of length p (p - number of sites), consisting of the symbols "o" (site is occupied by atom), "." (site is empty), or "x" (status of site not specified). Every other argument will lead to an error message, and the program will abort.

int Pattern::Multiplicity(string Comparison)

Return the number of matches of the indefinite pattern encoded in the string Comparison within the subset of patterns congruent to the encoded pattern.

(8) PATTERN OUTPUT FUNCTIONS

string Pattern::Structure()

Return an object of type String containing the numbers of the occupied sites in the Pattern object.

string Pattern::Graphics()

Return an object of type String of length p characters, consisting of the symbols "o" (for occupied sites) and "." (for empty sites).

(9) PATTERN VARIABLES

long Pattern::Code

The binary storage for the specified pattern.