The game of life is normally played on a grid. See http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life . You can imagine (as I was inspired by a nice old prof☺), a B23/S24 version of it played on the line. For the line version, the 4 significant neighbours of a cell are the next and second next cells at its right and left.
Write a set of Lisp functions to implement this variation of the game.
The main function must be **`(LifeLine conf gen)`**, where **`conf`** is the initial configuration of the game – represented as a list containing the 'relative' coordinates of live cells, and **`gen`** is the number of generations the game is to be played for. The output should be the sequence of configurations corresponding to each generation.
For example **`(LifeLine '(1 2) 5)`** should produce the output:

```
(1 2)
(0 3)
(1 2)
(0 3)
(1 2)
(0 3)
```

While **`(LifeLine '(13 14 15 16 17) 8)`** should produce the output:

```
(13 14 15 16 17)
(12 13 15 17 18)
(11 13 14 15 16 17 19)
(12 15 18)
(13 14 16 17)
(12 14 16 18)
(13 14 15 16 17)
(12 13 15 17 18)
(11 13 14 15 16 17 19)
```

You may only use the following LISP functions and predicates

```
(car x)
(cdr x)
(cons x y)

(atom x)
(null x)
(eq x y)
(equal x y)
(numberp x)
(listp x)

(eval y)
(funcall x ...)
(apply x y)
```

special forms (including logic connectives)

```
(defun ...)
(defmacro ...)
(let ((x y) (u v)...) z)
(lambda (x y) ...)

(quote x)      and its short form 'x
(function y)     as well as #'
`( ...)
(list a1 a2 ...)
(if x y z)
(cond  ... )
(and x y ...)
(or x y ...)
(not x)
(mapcar f l)
```



and numeric operators and comparisons such as

```
(+ x y)
(- x y)
(* x y)
(/ x y)
(< x y)
(mod x y)
(floor x)
(ceiling x)
(> x y)
(= x y)
(<= x y)
(>= x y)
```

You may also use a combination of car and cdr, such as

```
(cadr ...), (cdaar ...)
```

as well as

```
(first x)
(second x)
(third x)
...

(rest x)
```

etc.

For printing purposes you may use the functions:

**print**

**format**

If you would like to use any other functions of forms, please talk to me.