Macros

```
(defmacro nil! (var)
  (list 'setf var nil))
```

Evaluation en deux phases: macro-expansion puis évaluation

Important: la macro-expansion est faite par le compilateur. L'expression :

(nil! x)

est transformée en l'expression :

(setf x nil)

qui sera compilée à la place de l'expression initiale Les macros permettent donc de programmer le compilateur.

Backquote

```
* (setf a 1 b 2)

2
* `(a is ,a and b is ,b)

(A IS 1 AND B IS 2)
* (defmacro nil! (var)
    `(setf ,var nil))

NIL!
* (nil! *x*)

NIL
*
```

Backquote

```
* (setf lst '(1 2 3))

(1 2 3)
* `(the list is ,lst I think)

(THE LIST IS (1 2 3) I THINK)
* `(the elements are ,@lst I think)

(THE ELEMENTS ARE 1 2 3 I THINK)
*
```

Exemples

```
(defmacro while (test &rest body)
  `(do ()
       ((not ,test))
     ,@body))

(defmacro for (init test update &rest body)
  `(progn
     ,init
     (while ,test
       ,@body
       ,update)))
```

Difficultés

Capture de variables

```
;;; incorrect
(defmacro ntimes (n &rest body)
  `(do ((x 0 (1+ x)))
       ((>= x ,n))
     ,@body))

* (let ((x 10))
    (ntimes 5
      (setf x (1+ x)))
    x)
10
*
```

Difficultés

Multiple évaluation

```
;;; incorrect
(defmacro ntimes (n &rest body)
  (let ((g (gensym)))
    `(do ((,g 0 (1+ ,g)))
         ((>= ,g ,n))
       ,@body)))
```

```
* (let ((v 10))
    (ntimes (setf v (1- v))
            (format t "*")))
*****
NIL
*
```

Difficultés

```
;;; correct
(defmacro ntimes (n &rest body)
  (let ((g (gensym))
        (h (gensym)))
    `(let ((,h ,n))
       (do ((,g 0 (1+ ,g)))
           ((>= ,g ,h))
         ,@body))))
```