```
;; TP 2 - Pretty Printer
;; Pierre Gaudichon & Cyril Leconte


;; Intro

;; Le but était de créer un programme qui, à partir de la syntaxe d'un
;; programme rends la même syntaxe mais plus lisible (selon des normes
;; définies).


;; Conclusion

;; Nous avons rencontré quelques difficultées :
;;
;; - mettre les points-virgules en fin de ligne.
;; - `pretty-print` ne peut pas être modifié dans Racket.



;; (I) Imports
;; -----------------------------------------------------------------------------

(load "test.ss")
(load "ccc.ss")



;; -----------------------------------------------------------------------------
;; (II) Preparation Tests

(define show-all-tests #f)



;; -----------------------------------------------------------------------------
;; (III) Indentation

;; default-indent :: Int
;; Default indentation for output programm. With the number of space.
(define indent-default 1)


;; indent-search :: context-name x [indent-spec] ->  Integer
;;                  context-name ,  indent-spec
;; Searches a list of indentation specifications.
(define indents-search (lambda (context-name indents-spec)
  (let (
        (pair (assoc context-name indents-spec)))
    (if pair
        (cdr pair)
        indent-default))))


;; make-indent :: Int -> String
;; String (length = a) composed of whitespace
(define make-indent (lambda (a)
 (make-string a #\space)))



;; -----------------------------------------------------------------------------
;; (IV) String Appends

;; append-string-before-all :: String x [String] -> [String]
;;                                b   ,    s
;; Append b before each string from l.
(define append-string-before-all (lambda (b l)
  (map
    (lambda (s)
      (string-append b s))
    l)))
```

```
;; append-string-after-all :: String x [String] -> [String]
;;                                b   ,    s
;; Append b after each string from l.
(define append-string-after-all (lambda (b l)
  (map
    (lambda (s)
      (string-append s b))
    l)))


;; append-string-after-all-butt-the-last-bitch :: String x [String] -> [String]
;;                                                  b    ,   l
;; Sefl-explanatory.
(define append-string-after-all-butt-the-last-bitch (lambda (b l)
  (if (<= (length l) 1)
    l
    (cons (string-append (car l) b) (append-string-after-all-butt-the-last-bitch b (cdr l))))))



;; ---------------------------------------------------------------------------
;; (V) Pretty Prints

;; pretty-print-expr :: expr x [indent-spec] -> String
;;                         e   ,      s
;; Pretty print an expression, return a string for that expression.
(define pretty-print-expr (lambda (e s) (cond
  ((NIL? e)  "nil")
  ((CST? e)  (CST->name e))
  ((VAR? e)  (VAR->name e))
  ((CONS? e) (string-append "(cons " (pretty-print-expr (CONS->arg1 e) s) " " (pretty-print-expr(CONS-
>arg2 e) s) ")"))
  ((HD?  e)  (string-append "(hd " (pretty-print-expr (HD->arg e) s) ")"))
  ((TL? e)   (string-append "(tl " (pretty-print-expr (TL->arg e) s) ")"))
  ((EQ? e)   (string-append (pretty-print-expr (EQ->arg1 e) s) " =? " (pretty-print-expr (EQ->arg2 e)
s))))))


;; pretty-print-command :: command x [indent-spec] -> [String]
;;                           c      ,       s
;; Pretty print a command and return a list of lines (String).
(define pretty-print-command (lambda (c s)
  (cond
    ((NOP? c) (list "nop"))
    ((SET? c) (list (string-append (VAR->name (SET->var c)) " := " (pretty-print-expr (SET->expr c)
s))))
    ((WHILE? c) (append
      (list (string-append "while " (pretty-print-expr (WHILE->cond c) s) " do"))
      (append-string-before-all (make-indent (indents-search "WHILE" s)) (pretty-print-commands (WHILE-
>body c) s))
      (list"od")))
    ((FOR? c) (append
      (list (string-append "for " (pretty-print-expr (FOR->count c) s) " do"))
      (append-string-before-all (make-indent (indents-search "FOR" s)) (pretty-print-commands (FOR-
>body c) s))
      (list "od")))
    ((IF? c) (append
      (list (string-append "if " (pretty-print-expr (IF->cond c) s) " then"))
      (append-string-before-all (make-indent (indents-search "IF" s)) (pretty-print-commands (IF->then
c) s))
      (list "else")
      (append-string-before-all (make-indent (indents-search "IF" s)) (pretty-print-commands (IF->else
c) s))
      (list "fi"))))))


;; pretty-print-commands :: [command] x [indent-spec] -> [String]
;;                              cs     ,     s
;; Pretty print some commands and return a list of lines (String).
(define pretty-print-commands (lambda (cs s)
  (cond
    ((null? cs)
      (list))
    ((= (length cs) 1)
```

```
      (pretty-print-command (car cs) s))
    (else (let* (
      (command (pretty-print-command (car cs) s))
      (commands (pretty-print-commands (cdr cs) s))
      (dnammoc (reverse command))
      (command-ready (reverse (cons (string-append (car dnammoc) " ;") (cdr dnammoc)))))
        (append command-ready commands))))))


;; pretty-print-in :: [Var] x [indent-spec] -> String
;;                    vs   ,        s
;; Pretty print the list of variables names, basically `vs.join(", ")`.
(define pretty-print-in (lambda (vs s)
  (cond
    ((null? vs) "")
    ((= (length vs) 1) (VAR->name (car vs)))
    (else (string-append (VAR->name (car vs)) ", " (pretty-print-in (cdr vs) s))))))


;; pretty-print-out :: [Var] x [indent-spec] -> String
;;                     vs   ,        s
;; Pretty print the list of variables names, basically `vs.join(", ")`.
(define pretty-print-out pretty-print-in)


;; pretty-print-progr :: Progr x [indent-spec] -> [String]
;;                       progr ,       s
;; Pretty print all the programm, return a list of lines (String).
(define pretty-print-progr (lambda (progr s)
  (append
   (list (string-append "read " (pretty-print-in (PROGR->in progr) s)) "%")
   (append-string-before-all (make-indent (indents-search "PROGR" s)) (pretty-print-commands (PROGR-
>body progr) s))
   (list "%" (string-append "write " (pretty-print-out (PROGR->out progr) s))))))



;; ---------------------------------------------------------------------------
;; (VI) Final

;; pretty-print :: Progr x [indent-spec] -> String
;;                 progr ,       s
;; Pretty print the programm. From a AST to a string.
(define pretty-printer (lambda (progr . s)
  (apply string-append (append-string-after-all-butt-the-last-bitch "\n" (pretty-print-progr progr
s)))))
```