Option PF2, L2-Info&Math

« Programmation Avancée en Scheme »

EXPRESSIONS RÉGULIÈRES

Exercice 8.3 En utilisant des **expressions régulières**, écrivez :

a) La fonction `(nb-blancs-en-tete str)` prenant une chaîne `str` et retournant le nombre d'espaces en tête :

`(nb-blancs-en-tete "_____une__chaîne_") → 5`

b) La fonction `(premier-mot str)` retournant le premier « mot » [suite de lettres] d'une phrase `str`, ou bien `#f` :

`(premier-mot "12345Mot!78") → "Mot"`

c) La fonction `(today)` retournant la date d'aujourd'hui sous forme de liste d'entiers :

```
(today) → (19 3 2009)   ; le 19 mars 2009
```

Vous utiliserez la formule `(date->string (seconds->date (current-seconds)))` qui retourne la date sous la forme d'une chaîne de caractères, puis vous traiterez cette chaîne avec des *regexp* pour construire le résultat (19 3 2009). Utilisez une A-liste pour traduire le mois en un nombre entier…

N.B. Il y a une solution plus rapide consistant à ouvrir les champs de la structure `date`*, mais l'exercice a pour objet le traitement des regexp… Il y a une solution encore plus rapide consistant à demander la date en format indian, mais vous ne le ferez pas, ce serait de la triche ! Oh, faites-le si ça vous amuse…*

ENTRÉES-SORTIES A LA CONSOLE

Exercice 9.1 Ecrire une fonction sans résultat `(integration)` qui demande à l'utilisateur d'entrer une expression en `x` [pas une fonction !], ainsi qu'un pas dx représentant la précision, et qui retourne une valeur approchée de l'intégrale de cette expression sur l'intervalle fixe [0,2]. On souhaite voir fonctionner exactement la session ci-dessous :

```
> (integration)
Donnez une expression en x : (* x (sin x))
Donnez le pas dx : 0.001
L'intégrale sur [0,2] est approximativement 0.871250201881867
```

FICHIERS

Exercice 9.2 Ecrire une fonction `(file->list f)` prenant un nom de fichier disque `f` contenant uniquement des nombres, sur plusieurs lignes, disposés n'importe comment, et retournant la liste de ces nombres, dans l'ordre où ils apparaissent.

Exercice 9.3 Ecrire la fonction `(read-line-until port pred?)` prenant un port d'entrée `port` et un prédicat unaire `pred?`. Elle lit le port ligne à ligne, et renvoie la première ligne vérifiant le prédicat `pred?`. Cette fonction est utilisée dans le cours page 20. Dans le cas où aucune ligne ne vérifie `pred?`, elle retournera `#f`. *Pour la tester, cherchez la première ligne contenant le mot* `define` *dans ce fichier* `tp10.rkt`.

PROGRAMMATION D'UN CLIENT WEB

Exercice 9.4 En prenant modèle sur ce qui a été vu en cours [`tcp-connect`, `regexp`, etc], on vous demande d'écrire une fonction `(departement num)` retournant le nom du département de numéro num en vous basant sur la page web http://www.les-departements.fr/carte-des-departements.html.

i) Vous allez vous connecter avec votre navigateur Web sur cette page. Enregistrez le *source HTML* de cette page Web au format texte, dans un fichier `departements.html` sur votre disque dur.

ii) Par un clic droit sur ce fichier `departements.html`, ouvrez-le avec Emacs par exemple. Positionnez-vous à l'endroit dans le source où se trouvent les numéros de département. Supposons que nous souhaitons trouver le département 06. Le « mot magique » qui nous intéresse est un mot qui intervient si possible pour la première fois dans ce source sur la ligne où se trouve l'information que nous cherchons. Notez le « mot magique ». C'est gagné ! En effet :

iii) Il suffit alors d'ouvrir un canal HTTP avec le serveur, lire la page HTML ligne à ligne jusqu'à tomber sur le mot magique. En possession de la ligne convoitée, il suffit d'en extraire le nom du département par une expression régulière.

Et voilà comment extraire de l'information automatiquement sur le Web ! ☺…

```
> (departement "06")
"Alpes-maritimes"
```

N.B. Ici nous lisons les lignes de la page Web une à une. Mais pour récupérer dans une string la totalité du texte d'une page Web pour la traiter ensuite (c'est une autre stratégie), utilisez la librairie `net/url`. *Par exemple :*

```
#lang racket
(require net/url)
(define str
 (port->string (get-pure-port (string->url "http://www.les-departements.fr/carte-des-departements.html"))))
(printf "Web page : ~s\n" str)
```

EXERCICES SUPPLÉMENTAIRES

Exercice 9.5 Ecrire une procédure `(remove-comments f-in f-out)` prenant en entrée un fichier Scheme `f-in` et produisant dans le fichier `f-out` le même contenu que `f-in` mais sans aucun commentaire. On souhaite que le fichier `f-out` soit agréable à regarder, avec une indentation correcte même si ce n'est pas exactement la même que dans `f-in`. Testez votre programme avec un fichier de TP contenant de grosses définitions…

N.B. « Agréable à regarder » se dit « pretty » en anglais. Exemple :
« Eva is writing a pretty-writer in Scheme, that's a fine job ! ».

Bref, utilisez `pretty-write`.

Exercice 9.7 a) En utilisant la fonction `get-output-cmd` du cours page 16, programmez une fonction `(heure)` retournant une liste `(h m)` contenant l'heure actuelle :

```
(heure) → (9 17)   ; il est 9h17
```

La commande Unix à utiliser est la commande « date »…

c) Documentez-vous sur la possibilité de lancer à partir de Racket un logiciel externe… *Notez que la commande « open » propre à l'UNIX de Mac permet d'ouvrir un fichier à partir de la ligne de commande en lançant automatiquement l'application MacOS-X adéquate [testez open xxx.pdf par exemple au Terminal].*

Exercice 9.9 Programmez un aspirateur d'images `(image-sucker url)` prenant l'adresse d'une page Web contenant des images, cherchant toutes les références aux images `<img src="…">`, et affichant ces images dans un canvas, ou bien les enregistrant sur votre disque dur…

etc. etc. Créez vos propres logiciels ***exploitant les ressources du Web*** *!…*

COMPLEMENT : LE WEB ET LES IMAGES

Dans cette section à lecture optionnelle, vous apprendrez :

- à **télécharger** une image d'un serveur Web sur votre disque dur, par programme.
- à **lire** cette image sur votre disque dur, octet par octet, pour en extraire les dimensions.
- à **afficher** cette image dans un canvas de votre interface graphique Scheme.

Sur le serveur `www.hohohu.com`, j'ai trouvé [manuellement ou par programme] une image `bathyscaphe.gif` que j'ai envie de m'approprier [je vous laisse la partie légale, qu'il ne faut pas négliger, il y a peut-être des droits sur l'image]. Suivant le principe qu'un bon code vaut mieux qu'un long discours, allons-y :

```
;;; Connexion a un serveur Web en mode HTTP
(define-values (p-in p-out) (tcp-connect "www.hohohu.com" 80))
;;; je ne veux pas être tamponné quand je parle !
(file-stream-buffer-mode p-out 'none)
;;; je poste ma requête GET au serveur
(fprintf p-out "GET http://www.hohohu.com/puzzle/gif/bathyscaphe.gif HTTP/1.0\n\n")
(printf "Connexion au serveur www.hohohu.com\n")
;;; je commence à lire ligne par ligne le header jusqu'à la ligne "Content-Length:"
(printf "Je lis le header :\n")
(define CONTENT #f)          ; contiendra le nombre d'octets de l'image qui m'intéresse
(while (not CONTENT)
       (let* ((str (read-line p-in 'any)) (essai (regexp-match "Length:[^0-9]*([0-9]+)" str)))
         (printf "   ~a\n" str)
         (when essai (set! CONTENT (string->number (cadr essai)))
           (printf "Je vais donc lire ~a bytes.\n" CONTENT))))    ; ok, je tiens le nombre d'octets !
;;; je continue à lire le header jusqu'à une ligne vide [longueur = 0] :
(printf "Je continue dans le header :\n")
(define str (read-line p-in 'any))           ; 'any car la fin de ligne dépend de l'O.S.
(while (> (string-length str) 0)
       (printf "   ~a\n" str)
       (set! str (read-line p-in 'any)))
(printf "\nLigne vide, le header est fini !\n")
;;; J'ai passé la ligne vide, je suis sur la zone data de l'image.
;;; Je lis octet à octet avec read-byte et je les transfère sur mon disque dur.
(printf "Je lis les ~a octets du fichier et je les ecrits dans un fichier bathyscaphe.gif\n" CONTENT)
(call-with-output-file "bathyscaphe.gif"
  (lambda (img-out)
    (do ((i 0 (+ i 1)))
      ((= i CONTENT) (void))
      (let ((b (read-byte p-in)))
        (write-byte b img-out))))
  #:mode 'binary
  #:exists 'replace)
;;; Je ferme les portes pour éviter les courants d'air.
(printf "C'est fait, je ferme les ports !\n")
(close-input-port p-in)
(close-output-port p-out)
```

Ok, vous êtes encore vivants ? Cool, on tourne la page...

```
;;; Une petite fonction qui prend une image GIF sur le disque et extrait
;;; ses dimensions. Après un peu de surf, je me suis documenté sur :
;;;   http://www.colosseumbuilders.com/imageformats/gif87a.txt (page 4)
(define (size-of-gif-image fich)        ; fich est un fichier contenant une image gif
  (let ((v (make-vector 4)))
    (call-with-input-file fich
      (lambda (p-in)
        ; je lis les 6 premiers octets [type de l'image]
        (printf "  - L'image est de type ")
        (do ((i 0 (+ i 1)))
          ((= i 6) (printf "\n"))
          (let ((b (read-byte p-in)))
            (printf "~a" (integer->char b))))
        ; je stocke les 4 octets suivants dans un vecteur
        (let ((v (build-vector 4
                               (lambda (i) (read-byte p-in)))))
          (let ((largeur (+ (vector-ref v 0) (* 256 (vector-ref v 1))))
                (hauteur (+ (vector-ref v 2) (* 256 (vector-ref v 3)))))
            (printf "  - L'image a pour taille ~a x ~a\n" largeur hauteur)
            (list largeur hauteur)))))))
;;; Bon, j'applique ma fonction au fichier-image que je viens de construire :
(printf "Je lis les premiers octets de mon fichier pour extraire de l'information :\n")
(define DIMS (size-of-gif-image "bathyscaphe.gif"))
```

;;; Maintenant je vais lire l'image dans mon fichier et la placer dans un canvas Scheme. Happy ?

```
 (define FRAME (new frame% (label "bathyscaphe.gif")
                    (stretchable-width #f) (stretchable-height #f)))
(define BITMAP (make-object bitmap% "bathyscaphe.gif" 'gif))
(define CANVAS (new canvas% (parent FRAME)
                    (min-width (car DIMS))
                    (min-height (cadr DIMS))
                    (paint-callback (lambda (c dc)
                                      (send dc draw-bitmap BITMAP 0 0)))))
(send FRAME show #t)
```