Université Nice Sophia-Antipolis
L1 Info / Math-Info
Algorithmique et Programmation Fonctionnelle
Examen : 22 mai 2018
Durée : 2 heures

Prénom _____

Nom _____

Note

Num. Etu.

Tous les documents sont interdits. Les calculatrices et les téléphones portables doivent être rangés ainsi que tout autre matériel à l'exception d'un stylo et d'un effaceur. Un mémo est donné en fin de sujet

Exercice 1 Nombres parfaits (4 points)

1. Écrivez une fonction `(diviseur? a b)` qui renvoie `#t` si `b` divise `a`. Par exemple, `(diviseur 6 3)` renvoie `#t` et `(diviseur 3 6)` renvoie `#f`.

2. Écrivez une fonction `(somme-diviseurs n)` qui renvoie la somme des diviseurs de `n`. Par exemple, `(somme-diviseurs 6)` renvoie 12 ($= 1+2+3+6$).

3. En déduire une fonction `(parfait? n)` qui renvoie `#t` si n est parfait, autrement dit s'il est égal à la moitié de la somme de ses diviseurs. Par exemple, `(parfait? 6)` renvoie `#t` et `(parfait? 5)` renvoie `#f`.

.

```
(define (maximum L)
   (cond
      [(empty? L) (error "liste␣vide")]
      [(> (first L) (maximum (rest L))) (first L)]
      [else (maximum (rest L))]))

(define (premier-pair L)
   (cond
      [(empty? L) #f]
      [(even? (first L)) (first L)]
      [else (premier-pair (rest L))]))

(define ($revappend L1 L2)
   (if (empty? L1)
      L2
      ($revappend (rest L1) (cons (first L1) L2))))
```

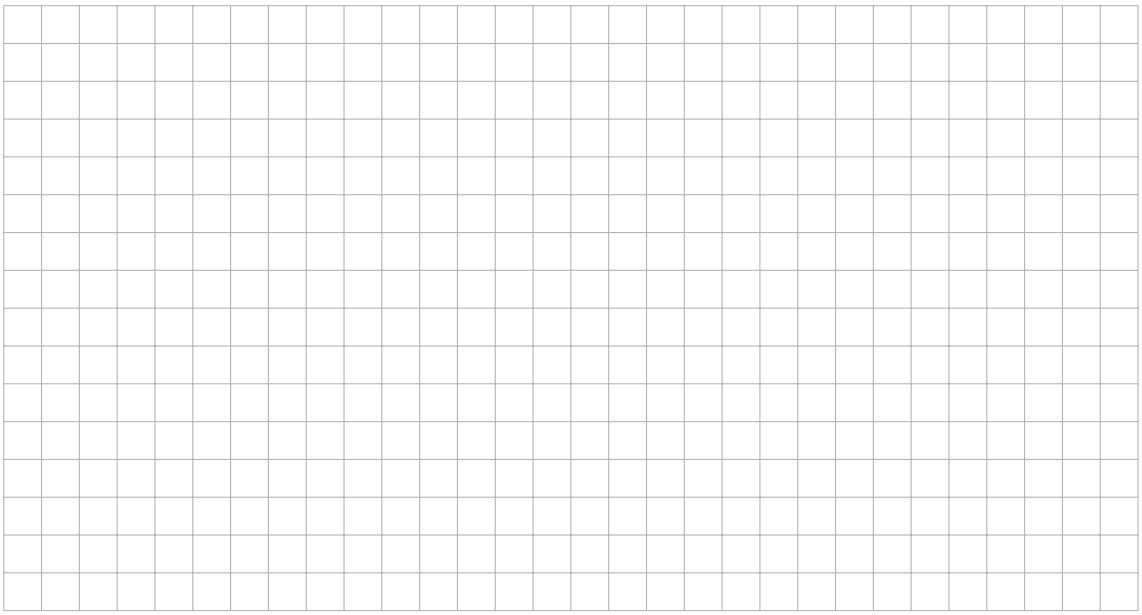
Exercice 2 Fonctions récursives terminales (3 points)

Parmi les fonctions ci-dessus, indiquez celle qui n'est pas récursive terminale, autrement dit celle qui n'est pas "itérative", parce que sa récurrence est "enveloppée".

La fonction récursive enveloppée est :

- ☐ `maximum`
- ☐ `premier-pair`
- ☐ `$revappend`

Transformez cette fonction en une fonction récursive terminale (=itérative).

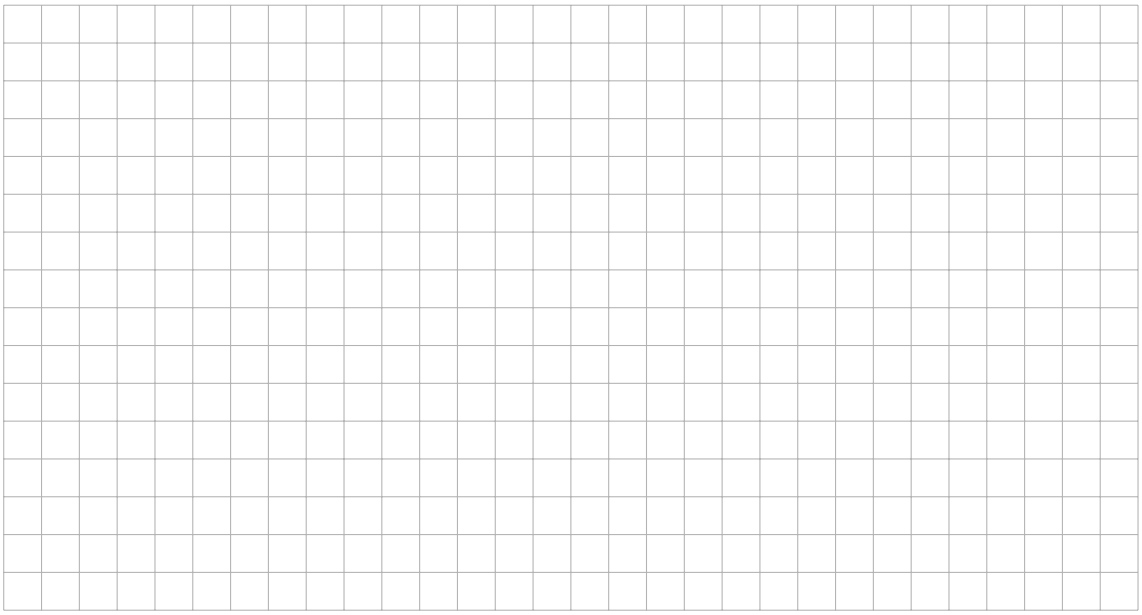


.

Exercice 3 Fibonacci (4 points)

Écrivez une fonction `(fibonacci? L)` qui renvoie `#t` si `L` contient le début d'une suite de Fibonacci : pour tout $i \geq 2$, le $i$ème entier de `L` est la somme des deux précédents. Par exemple

- `(fibonacci? '(2 1 3 4 7))` renvoie `#t`
- `(fibonacci? '(1 3 4 7))` renvoie `#t`
- `(fibonacci? '(3 1 3 4 7))` renvoie `#f`
- `(fibonacci? '())`, `(fibonacci? '(2))`, et `(fibonacci? '(2 1))` renvoient `#t`.



Exercice 4 Nombre de feuilles contenant une variable dans un arbre (3 points)

Écrivez une fonction `(nb-vars A)` qui attend en argument un arbre `A` et qui renvoie le nombre de feuilles de l'arbre qui sont des variables.
Par exemple, `(nb-vars '(+ (- 1 x) (* x (- 12 y))))` renvoie 3, car les feuilles variables sont `x`, `x`, et `y`.

.

Exercice 5 Animation : système solaire (6 points)

On veut réaliser une animation figurant les mouvements de 7 planètes autour d'un soleil, chacune ayant sa propre orbite circulaire et avançant à sa propre vitesse. La scène sera un carré noir 400×400 avec en son centre un disque jaune de rayon 25.

```
(define SCENE





)
```

Chaque planète aura en propre :

- un rayon terrestre r, compris entre 1 et 8,
- une couleur c parmi rouge, vert, bleu
- une distance D au soleil (autrement dit le rayon de son orbite), comprise entre 100 et 200
- une position angulaire θ (initialement à 0) et une vitesse angulaire $d\theta$ comprise entre $\frac{\pi}{10}$ et $\frac{\pi}{50}$ (les planètes tournent toutes dans le même sens)

Une structure `planete` groupe toutes ces informations, une fonction `(tire-planete)` renvoie une nouvelle planète tirée au hasard, et le monde initial, construit à l'aide de build-list, est une liste de 7 planètes tirées au hasard.

```
(define-struct planete (r c D theta dtheta))
(define (tire-planete) ;  -> planete















)
(define MONDE-INITIAL ; une liste de planetes
   (build-list 7 (lambda (i) (tire-planete))))
```

.

A chaque tic d'horloge, on déplace chaque planète en incrémenant sa position angulaire θ de sa vitesse angulaire $d\theta$.

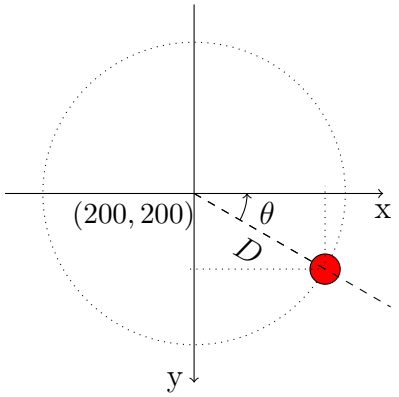
```
(define (deplace planet) ; planete -> planete




)
(define (suivant monde)
  ; liste de planetes -> liste de planetes
  ; en une ligne avec une fonction d'ordre superieur

)
```

Les coordonnées (x, y) d'une planète dépendent de D et θ et se calculent avec un peu de trigonométrie. On en déduit alors la fonction de dessin, qui procède par récurrence sur le nombre de planètes dans le monde. Il ne manque alors plus que le big-bang, dont on vous fait grâce (voir plus loin).




```
(define (x-coord D theta) ; nombre * nombre -> nombre

)
(define (y-coord D theta) ; nombre * nombre -> nombre

)
(define (dessine monde) ; monde -> image









)
```

.

```
(big-bang MONDE-INITIAL
 (on-draw dessine)
 (on-tick suivant))
```

Memento

```
;; MATH

(quotient n m) ; quotient de la division de n par m
(modulo n m)   ; reste de la division de n par m
(random n)     ; renvoie un nombre au hasard
               ; compris entre 0 et n-1


;; IMAGES
empty-image          ; l'image vide
(rectangle l h style color) ;; un rectangle
(circle r style color) ;; un cercle/ un disque
(place-image IMG1 x y IMG2)
                     ; ajoute IMG1 en x,y dans IMG2

;; LISTES : on suppose L = (L[1] ... L[n])
(cons x L)           ; = (x L[1] ... L[n])
(first L)            ; = L[1]
(rest L)             ; = (L[2] ... L[n])
(second L)           ; = L[2]
(third L)            ; = L[3]
(length L)           ; = n

;; ARBRES
(arbre op A1 A2)     ; construit un arbre
(racine A)           ; operateur racine de A
(fg A)               ; sous-arbre gauche de A
(fd A)               ; sous-arbre droit de A
(feuille? A)         ; vrai si A est une feuille
(number? A)          ; vrai si A est une feuille nombre
(symbol? A)          ; vrai si A est une feuille variable
```