INFORMATIQUE Classe de Seconde

Correction Contrôle n° 1

Première partie (répondre sur cette feuille)

Sur les langages de programmation et la programmation

1. Expliquer la différence entre un langage dit de bas niveau et un langage évolué.

…… « bas niveau » : proche de la machine (du binaire)……

…… « évolué » : proche du langage humain……

2. Expliquer la différence entre un langage interprété et un langage compilé.

……analogie : traduction simultanée (interprété) et en bloc (compilé)……...

……

3. Les deux premiers langages évolués sont FORTRAN et LISP (dont DrRacket est un dialecte) et les noms de ces deux langages sont des contractions de deux expressions anglaises : quelles sont ces deux expressions ?

……FORTRAN : ***FOR***mula ***TRAN***slator……(langage adapté aux calculs mathématiques)……

……LISP : ***LIS***t ***P***rocessing……(langage adapté au traitement de listes)…….

4. Quand on cherche à écrire une fonction (en DrScheme), avant d'écrire quoi que ce soit, quelle est la première chose à faire ?

……Définir les ensembles de départ et de sortie : à savoir la nature des données et résultats……

……

5. Expliquer en quelques mots :

a) ce qu'est une fonction récursive

……fonction qui s'appelle elle-même dans sa définition……

……

b) ce qu'est une fonction récursive terminale

……terminale si l'appel récursif n'est enveloppé par aucune autre fonction que le IF……

……

c) ce qu'est une fonction récursive non terminale

…… non terminale si l'appel récursif est enveloppé par une autre fonction que le IF …….

……

Ces deux notions (récursivité terminale ou non terminale – enveloppée-) ont eu de nombreuses illustrations dans les exemples traités en cours. Exemple le plus simple du cours :

Récursivité terminale	*Récursivité non terminale*
(define (add x y) (if (zero ? y) x (add (1+ x) (1- y))	(define (add2 x y) (if (zero ? y) x (1+ (add2 x (1- y))
Relation de récurrence associée	
$x + y = (x + 1) + (y - 1)$	$x + y = 1 + [x + (y - 1)]$

Soit la fonction suivante :

```
define (inconnu  n)
      (if (<= n 2 )
           n
           (if (even? n)       ; anglais    even = pair
                     (inconnu (div2 n))
                     (inconnu (1+ (* 3 n))))))
```

Faire tourner « à la main » (*sur cette feuille*) cette fonction sur l'exemple proposé :
(*en clair, faire une trace de la fonction comme il a été vu en cours – respecter l'indentation des appels !- et bien faire apparaître les résultats intermédiaires en correspondance avec les appels correspondants):*

```
(inconnu 6)
      ⟶ (inconnu 3)
            ⟶ (inconnu 10)
                  ⟶ (inconnu 5)
                        ⟶ (inconnu 16)
                              ⟶ (inconnu 8)
                                    ⟶ (inconnu 4)
                                          ⟶ (inconnu 2) condition arrêt !
                                          ⟶ 2
                                    ⟶ 2
                              ⟶ 2
                        ⟶ 2
                  ⟶ 2
            ⟶2
      ⟶ 2
2                                                    Résultat final : …2
```

Que calcule cette fonction ?
……Pour l'argument n, cette fonction retourne toujours 2 car quelque soit l'argument n on finit par arriver sur une puissance de 2 qui, par divisions successives par 2, …. donnera 2 lors du test d'arrêt !
Cette fonction est –elle récursive terminale ?
Cette fonction est récursive terminale : elle n'est pas enveloppée et le résultat est obtenu lorsque la condition d'arrêt est vérifiée

Remarque : Cette trace peut être faite dans DrRacket en tapant :

```
(require (lib "trace.ss"))          (…avant d'écrire toutes les définitions !)
```

6. ***Soit la fonction suivante :***

```
(define (inconnu a b)
        (if (zero ? b)
            a
            (inconnu b (remainder a b))))
```

Information : la fonction ***remainder*** (qui est une fonction prédéfinie de Scheme) retourne le reste de la division entière de a par b
Exemples
(remainder 15 5) *retourne 0*
(remainder 23 4) *retourne 3*

Faire tourner « à la main » (au brouillon) cette fonction en prenant :

a) a = 45 b = 10 Réponse : ……5……

b) a = 206 b = 4 Réponse : ……2……

c) a = 77 b = 6 Réponse : ……1……

et dans chaque cas, préciser le résultat retourné par la fonction.
Que calcule cette fonction ?……Cette fonction calcule le pgcd de deux nombres.

```
; correction DST1  14-15

(require (lib "trace.ss"))
(define (1+ n) (+ n 1))
(define (1- n) (- n 1))
(define (mul2 n) (* 2 n))
(define (div2 n) (quotient n 2))



(define (inconnu  n)
     (if (<= n 2 )
          n
          (if (even? n)
                    (inconnu (div2 n))
                    (inconnu (1+ (* 3 n))))))

; somme entiers jusque n
(define (somme_entiers n)
  (if (zero? n)
      0
      (+ n (somme_entiers (1- n)))))
;on peut montrer que  S = n (n + 1)/2

; somme impairs jusque n (lui-même impair)
(define (somme_impairs n)
  ; n nécessairement impair !
  (if (= n 1)
      n
      (+ n (somme_impairs (1- (1- n))))))
; faire tester (somme_impairs 99) => 2500  (ou somme_impairs de 49,
199 etc

; on peut montrer que
; Simp = ((n+1)/2)^2
; soit Spair jusque n+1    Spair = 2 S(n+1/2) = (n+1)/2 * (n +3)/2
; Simp(n) = S(n+1) - Spair(n+1) = (n+1)(n+2)/2 - (n+1)/2 * (n +3)/2
=((n+1)/2)*[n+2 -(n+3)/2]=((n+1)/2)^2

; faire calculer la somme des pairs jusque n (lui-même pair)
; deux façons :
; méthode directe
(define (somme_pairs n)
   ; n nécessairement pair !
  (if (zero? n)
      n
      (+ n (somme_pairs (1- (1- n))))))
; méthode indirecte
(define (somme_pairs2 n)
   ; n nécessairement pair !
  (mul2 (somme_entiers (quotient n 2))))
; discuter de l'efficacité relative des deux fonctions ???
```

```
; la fonction de Fibonnaci
(define (fib n)
   (if (zero? n)
       0
       (if (= 1 n)
           1
           (+ (fib (1- n))
              (fib(1- (1- n)))))))
; > (fib 25)
; 75025
; > (fib 30)
; 832040
; au delà le temps de calcul est prohibitif d'où nécessité d'une
version terminale

(define (fib2 n)  ;l'obtention d'une version terminale n'était pas
demandée !
  (fib_bosse 0 1 n))
(define (fib_bosse acc1 acc2 n)
  (if (zero? n)
      acc1
      (fib_bosse acc2 (+ acc2 acc1) (1- n))))

; > (fib2 100)
; 354224848179261915075
; > (fib2 1000) ; avec la version terminale, on peut calculer fib de
1000
;
; somme des puissances de 2 jusque 2^k
; argument k : entier positif
; on peut remarquer que si  Sk-1 = 1 + 2 + 2^2 +..... 2^(k-1)
;alors                      Sk  = 1+  Sk-1 * 2
; avec comme condition d'arret  S0 = 1

(define (somme_puis2 k)
   ; k entier
  (if (zero? k)
      1
      (1+  (mul2 (somme_puis2 (1- k))))))

; évidemment une méthode plus "brutale" !
; car utilisant la fonction puissance
                                        define (puis x n) (expt x n))

(define (somme_puis2b k)
   ; k entier
  (if (zero? k)
      1
      (+ (puis 2 k) (somme_puis2b (1- k)))))

; on peut montrer que
; somme des puissances de 2 jusque 2^k
; argument k : entier positif
; S_puis2 = 2^(k+1) - 1[(]/(2 - 1)]
```

```
; un petit complément
La même fonction mais avec des puissances de 2 négatives

   ; k entier  soit donc 1 + 1/2 + 1/4 + 1/8, etc
  ; S_puis2_neg = 2 [1 - (1/2)^(k+1)]
  ; le resultat tend vers 2 quand k est grand !

  ; tester (exact->inexact (somme_puis2_neg k))  avec k = 5, 6,.. 10

; avec fonction précision à partir de k = 6  < 1%

(define (somme_puis2_neg k)
  (if (zero? k)
      1
      (+ (puis 2 (* -1 k)) (somme_puis2_neg (1- k)))))

; chercher l'approximation à x% près de 2
(define (precision x x_ref)
  (* 100 (/(abs (- x x_ref)) x_ref)))
```