Université d'Ottawa
Faculté de génie

École d'ingénierie et de
technologie de l'information

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

CSI2520
Paradigmes de programmation

EXAMEN de PRATIQUE

Durée: 3 heures
Professeur: Robert Laganière

14 questions

Questions à choix multiples.

Problème 1

Dans le paradigme de programmation concurrente, la concurrence peut se produire à plusieurs niveaux. Parmi les choix suivants, lequel n'est pas un niveau de concurrence acceptable:

A	Au niveau des énoncés
B	Au niveau des sous-programmes (procedures ou fonctions)
C	Au niveau des objets (et de leurs methodes)
D	Au niveau des programmes
E	Toutes ces réponses sont bonnes

Problème 2

Soit le prédicat et la requête ci-dessous :

```
p2(X, L) :- append(_ ,[_, X, _, _ | _], L).
?- p2(X, [2, 3, 5, 7, 11, 13]).
```

Si d'autres solutions sont demandées au système (en utilisant le ';'), combien de résultats seront obtenus au total? (sans compter le 'false') :

A	2 résultats
B	3 résultats
C	4 résultats
D	5 résultats

Problème 3

Soit les prédicats Prolog[1] suivants:

```
age(jean, 10).    age(bill, 23).    age(sam, 17).    age(paul, 5).
?- findall(X, age(X, Y), L).
```

Le résultat de la requête ci-dessus sera:

A	`L = [bill, jean, paul, sam].`

B	`Y = 5, L = [paul] ;`
	`Y = 10, L = [jean] ;`
	`Y = 17, L = [sam] ;`
	`Y = 23,L = [bill].`

C	`L = [jean, bill, sam, paul].`
D	`L = [10, 23, 17, 5].`

Problème 4

Un seul des prédicats ci-dessous additionne correctement les entiers compris à l'intérieur d'un intervalle donné, par exemple (sachant que 2+3+4+5=14), nous avons :

```
?- addUp(2, 5, Result).
Result = 14 .
```

La bonne définition est:

A	`addUp(N, N, N).`
	`addUp(L, H, T) :-`
	`  L < H,`
	`  LL is L + 1,`
	`  addUp(LL, H, TT),`
	`  T is L + TT.`
		B	`addUp(N, N, 0).`
			`addUp(L, H, T) :-`
			`  L < H,`
			`  LL is L + 1,`
			`  addUp(LL, H, TT),`
			`  T is L + TT.`

C	`addUp(N, N, N).`
	`addUp(L, H, T) :-`
	`  L < H,`
	`  LL is L + 1,`
	`  addUp(LL, H, TT),`
	`  TT is L + T.`
		D	`addUp(N, N, 0).`
			`addUp(L, H, T) :-`
			`  L < H,`
			`  LL is L + 1,`
			`  addUp(LL, H, TT),`
			`  TT is L + T.`

[1] **findall**(**Template, Goal, Bag**): creation d'une liste Bag des instances de Template pour lesquelles Goal est vérifié avec success.

Problème 5

Laquelle des solutions ci-dessous permet de réaliser l'énoncé conditionel suivant de la facon la plus *exacte et efficace:*:

```
if (X < 10) Y = 10;
else if (X < 100) Y = 100;
else Y = 0;
```

A	`f(X,  0) :- !.`
	`f(X, 10) :- X < 10, !.`
	`f(X, 100) :- X < 100.`
		B	`f(X, 10) :- X < 10.`
			`f(X, 100) :- X < 100.`
			`f(X, 0).`

C	`f(X, 10) :- X < 10, !.`
	`f(X, 100) :- X < 100, !.`
	`f(X, 0).`
		D	`f(X, 10) :- !, X < 10.`
			`f(X, 100) :- !, X < 100.`
			`f(X, 0).`

Problème 6

Voici la définition d'une fonction en Scheme:

```
(define sch6
  (lambda (l1 l2)
    (cond
      ((and (null? l1) (null? l2)) ())
      ((null? l1) (if (number? (car l2))
                      (cons (car l2) (sch6 l1 (cdr l2)))
                      (sch6 l1 (cdr l2))
      )           )
      ((number? (car l1)) (cons (car l1) (sch6 (cdr l1) l2)))
      (else (sch6 (cdr l1) l2))
) ) )
```

Soit l'expression suivante:

```
(sch6 '(1 5 a 7 g) '(3 b 3 c))
```

La valeur de cette expression sera:

A	`(a g b c)`
B	`(a b c g)`
C	`(1 3 3 5 7)`
D	`(1 5 7 3 3)`

Problème 7

Soit une fonction de tri en Scheme utilisant un prédicat de comparaison et une liste qui s'utiliserait, par exemple, de la façon suivante:

```
(sort char>? '(#\c #\a #\n #\t #\h #\i #\s #\b #\e))
```

Maintenant, soit l'expression suivante:

```
(let
  ( (q (lambda (x) (* (car x) (cdr x)))) )
  (sort
    (lambda (x y) (< (q y) (q x)))
    '((1 . 9)(2 . 7)(3 . 5)(4 . 3)(5 . 1))
) )
```

La valeur de cette expression est:

A	((3 . 5) (2 . 7) (4 . 3) (1 . 9) (5 . 1))
B	((5 . 1) (1 . 9) (4 . 3) (2 . 7) (3 . 5))
C	((1 . 9) (2 . 7) (3 . 5) (4 . 3) (5 . 1))
D	((5 . 1) (4 . 3) (3 . 5) (2 . 7) (1 . 9))

Problème 8

Une fonction Scheme permet d'insérer un entier dans une liste ordonnée d'entiers (sans répétitions). Voici quelques exemples d'exécution:

```
> (sch8 0 '(1 3 5 7))
(0 1 3 5 7)
> (sch8 4 '(1 3 5 7))
(1 3 4 5 7)
> (sch8 5 '(1 3 5 7))
(1 3 5 7)
> (sch8 8 '(1 3 5 7))
(1 3 5 7 8)
```

La seule définition **erronée** de cette fonction est:

A

```
(define sch8
  (lambda (x l)
    (cond
      ((null? l) (cons x ()))
      ((> x (car l)) (cons x l))
      ((< x (car l)) (cons (car l) (sch8 x (cdr l))))
      (else l)
) ) )
```

B

```
(define sch8
  (lambda (x l)
    (cond
      ((null? l) (cons x ()))
      ((< x (car l)) (cons x l))
      ((> x (car l)) (cons (car l) (sch8 x (cdr l))))
      (else (sch8 x (cdr l)))
) ) )
```

C

```
(define sch8
  (lambda (x l)
    (cond
      ((null? l) (cons x ()))
      ((< x (car l)) (cons x l))
      ((> x (car l)) (cons (car l) (sch8 x (cdr l))))
      (else l)
) ) )
```

D

```
(define sch8
  (lambda (x l)
    (cond
      ((null? l) (cons x ()))
      ((= x (car l)) l)
      ((> x (car l)) (cons (car l) (sch8 x (cdr l))))
      (else (cons x l))
) ) )
```

Problème 9

Une fonction Scheme calculant le produit scalaire de deux vecteurs **x** et **y** ($x_1y_1 + x_2y_2 + ... + x_ny_n$), représenté à l'aide d'une liste, doit être définie. Voici un exemple d'exécution:

```
> (sch9 '(1 -3 0 5 3) '(3 1 8 0 7))
21
```

La seule définition **correcte** est: (*indice*: 2 des ces définitions produiront une erreur d'exécution, une retournera le mauvais résultat):

A

```
(define sch9
  (lambda (v1 v2)
    (eval '+ (map * v1 v2)
) ) )
```

B

```
(define sch9
  (lambda (v1 v2)
    (apply + (map * v1 v2))
) )
```

C

```
(define sch9
  (lambda (v1 v2)
    (+ (map * v1 v2))
) )
```

D

```
(define sch9
  (lambda (v1 v2)
    (map + (map * v1 v2)
) ) )
```

Problème 10

Soit la definition Scheme suivante:

```
(define sch10?
  (lambda (x)
    (cond
      ((null? x) #f)
      ((null? (cdr x)) #f)
      ((< (car x) (cadr x)) (sch10? (cdr x)))
      (else #t)
) ) )
```

La fonction `sch10?` vérifie que la liste de nombre donnée en argument:

A	contient au moins 2 éléments et n'est pas en ordre croissant
B	contient au moins 2 éléments et est en ordre croissant
C	n'est pas vide et aucun éléments adjacents ne sont identiques
D	n'est pas vide et comprend deux éléments identiques

Problème 11

Soit une liste de trois éléments `#(item lowscore highscore)`. Il faut définir une fonction Scheme permettant de trouver la liste ayant l'amplitude la plus faible (`highscore-lowscore`). Par exemple:

```
> (sch11  '(#(one 4 21) #(two 5 19) #(ten 11 25)))
#(two 5 19)
```

La fonction `sch11` utilise la fonction `my-neat-extr` dont la définition est donnée au Problème 14. La définition correcte de cette fonction est:

A
```
(define sch11 (lambda (lst)
    (my-neat-extr (lambda (x y)
                    (< (- (vector-ref x 1) (vector-ref x 2))
                       (- (vector-ref y 1) (vector-ref y 2))))
                  lst
) ) )
```

B
```
(define sch11 (lambda (lst)
    (my-neat-extr (lambda (x y)
                    (< (- (vector-ref x 2) (vector-ref x 1))
                       (- (vector-ref y 2) (vector-ref y 1))))
                  lst
) ) )
```

C
```
(define sch11 (lambda (lst)
    (my-neat-extr (lambda (x y)
                    (< (- (vector-ref y 2) (vector-ref y 1))
                       (- (vector-ref x 2) (vector-ref x 1))))
                  lst
```

	)))
D	(define sch11 (lambda (lst) (my-neat-extr (lambda (x y) (< (- (vector-ref x 1) (vector-ref y 1)) (- (vector-ref x 2) (vector-ref y 2)))) lst)))

Problème 12

Soit la function Scheme suivante:

```
(define sch12
  (lambda (test l1 l2)
    (cond
      ((null? l1) l2)
      ((null? l2) l1)
      ((equal? (car l1) (car l2)) (sch12 test (cdr l1) l2))
      ((test (car l1) (car l2))
        (cons (car l1) (sch12 test (cdr l1) l2)))
      (else
        (cons (car l2) (sch12 test l1 (cdr l2))))
) ) )
```

L'expression suivante:

```
(sch12 char<? '(#\g #\c #\b #\a) '(#\g #\e #\d #\c))
```

donnera comme résultat:

A	(#\g #\c #\b #\a #\g #\e #\d #\c)
B	(#\c #\b #\a #\g #\e #\d #\c)
C	(#\g #\e #\d #\c #\b #\a)
D	(#\a #\b #\c #\d #\e #\g)

Problème 13

Soit l'expression suivante Scheme:

```
(
 (lambda (x y) (if (x y) y (cons (cdr y) (car y))))
 (lambda (z) (< (car z) (cdr z)))
 '(17 . 6)
)
```

La valeur de cette expression est:

A	(17 6)
B	(17 . 6)
C	(6 17)
D	(6 . 17)

Problème 14

Voici une fonction Scheme permettant de trouver l'extremum dans une liste. Par exemple:

```
(my-neat-extr > '(8 3 5 7))
```

trouvera le maximum, alors que

```
(my-neat-extr < '(8 3 5 7))
```

trouvera le minimum de la liste.

```
 (define my-neat-extr
  (lambda (c x)
    (define my-extr-help
      (lambda (curr-extr tail)
        (cond
          ((null? tail) curr-extr)
          ((c (car tail) curr-extr)
            (my-extr-help (car tail) (cdr tail)))
          (else (my-extr-help curr-extr (cdr tail)))
    ) ) )
    (if
      (null? x)
      x
      (my-extr-help (car x) (cdr x))
) ) )
```

Soit l'expression suivante:

```
(my-neat-extr (eval (read)) '("this" "is" "not" "what" "it" "seems"))
```

Si le résultat obtenu est `"this"`, quelle est l'expression qui a été entrée par l'utilisateur ?

A	`string<?`
B	`string>=?`
C	`equal?`
D	`(lambda (x y) (not (equal? x y)))`

Problème 15

Voici la representation Pascal d'un arbre binaire:

```
type infoType = integer;
     tree = ^treeNode;
     treeNode =
       record
         info: infoType;
         left, right: tree
       end;
```

Soit la fonction suivante:

```
procedure pas15(t: tree);
begin
  if t = nil then
    writeln
  else
  if (t^.left = nil) and (t^.right = nil) then
    write(t^.info, ' ')
  else
  begin
    pas15(t^.left);
    pas15(t^.right)
  end
end;
```

Pour un arbre **T** donné la procédure `pas15` affichera:

A	les nœuds internes de **T**
B	les feuilles de **T**
C	tous les nœuds de **T** parcourus de façon pré-ordre
D	la racine de **T**

Problème 16

Il faut définir un programme Pascal permettant de calculer la norme d'un nombre complexe. L'un des programmes suivants **ne produit pas** le bon résultat:

A

```
var n, r, i: real;
procedure norm(r, i: real;
               var n: real);
begin
  n := sqrt(sqr(r) + sqr(i))
end;
begin
  readln(r, i);
  norm(r, i, n);
  writeln(n)
end.
```

B

```
var r, i: real;
function norm(r, i: real): real;
begin
  norm := sqrt(sqr(r) + sqr(i))
end;
begin
  readln(r, i);
  writeln(norm(r, i))
end.
```

C

```
var n, r, i: real;
procedure norm(r, i: real);
var n: real;
begin
  n := sqrt(sqr(r) + sqr(i))
end;
begin
  readln(r, i);
  norm(r, i);
  writeln(n)
end.
```

D

```
type realptr = ^real;
var r, i: real;
    n: realptr;
procedure  norm(r, i: real;
    n: realptr);
begin
  n^ := sqrt(sqr(r) + sqr(i))
end;
begin
  readln(r, i);
  new(n);
  norm(r, i, n);
  writeln(n^)
end.
```

Problème 17

Soit le programme Pascal suivant:

```
const
  maxindex = 10;
type
  VEC = array [1 .. maxindex] of integer;
var
  A: VEC;
  i: integer;

procedure pas17(n: integer; var T: VEC);
var i, aux: integer;
begin
  aux := T[maxindex];
  for i:= maxindex downto 2 do T[i] := T[i - 1];
  T[1] := aux;
  if n > 1 then pas17(n - 1, T);
end;

begin
  for i := 1 to maxindex do A[i] := maxindex - i;
  pas17(6, A);
  for i:= 1 to maxindex do write(A[i])
end.
```

Ce programme affichera:

A	4567890123
B	0123456789
C	5432109876
D	9876543210

Problème 18

Définir une fonction Scheme permettant de trouver la médiane de trois. La médiane est le nombre se retrouvant au centre si les trois nombre était triés (par exemple la médiane de 8 3 9 est 8 et la médiane de 2 7 2 est 2). Vous choisissez la structure de données à utiliser pour manipuler les trois nombres.

Problème 19

Définir une fonction Pascal permettant de trouver la médiane de trois. La médiane est le nombre se retouvant au centre si les trois nombre était triés (par exemple la médiane de 8 3 9 est 8 et la médiane de 2 7 2 est 2). Vous choisissez la structure de données à utiliser pour manipuler les trois nombres.

Problème 20

Donnez 2 exemples de ce qui serait plus facile à programmer en Prolog qu'en Java. Donnez aussi deux exemples de ce qui serait plus facile à programmer en Java qu'en Scheme. Justifiez vos réponses.

Problème 21

Ordonner les langages C++, Java, Pascal, Prolog et Scheme selon votre préférence (de celui que vous préférez à celui que vous aimez le moins). Évidemment, ce n'est pas votre classement qui sera évalué, mais plutôt la justification de vos choix.