Université d'Ottawa
Faculté de génie

School of Electrical Engineering and Computer Science

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

École de science informatique et de génie électrique

CSI2520 Paradigmes de programmation
EXAMEN FINAL

Durée: 3 heures **Avril 2017**

Professeur: Robert Laganière

Nom: _____

Prénom: _____

Numéro d'étudiant: _____

Signature _____

Une feuille de notes premise (recto-verso).

Question	Résultat	Total
1		2
2		2
3		3
4		5
5		3
6		3
7		3
8		9
9		4
10		4
Total		38

Question 1 Règles Prolog [2 points]

Soit une base de faits Prolog dans le format suivant :

```
% name, studentId, course list
student(name(blake, [ann]), 33333, ['CSI2120'] ).

% course, type, name as list of text, maximum marks
evaluation('CSI2120', assignment(1),  ['Prolog', database ],  5).

% course, studentId, evaluation, mark
mark('CSI2120', 33333, assignment(1), 3.5 ).
```

La convention est que si un étudiant est absent à une évaluation, une note négative est attribuée. Par exemple :

```
mark('CSI2120', 12333, assignment(1), -1.0 ).
```

Écrire une requête permettant d'obtenir la liste des étudiants absents à une évaluation (Par exemple l'évaluation `assignment(1)` du cours csi2510).

```
?- _____________________________________________________


   _____________________________________________________

L = [12333, 45456].
```

Question 2 Liste Prolog [2 points]

Le prédicat ci-dessous est une redéfinition du prédicat `append` qui permet de concaténer deux listes.

```
q2append([],L,L).
q2append([H|T],L,[H|LL])  :-  q2append(T,L,LL).
```

Voici un exemple de son application :

```
?- q2append([1,2,3],[4,5],L).
L = [1, 2, 3, 4, 5].
```

Énumérer toutes les solutions qui seront trouvées par la requête suivante.

```
?- q2append(U,V,[1,2,3]).
```

Question 3 Liste Prolog [3 points]

Soit le prédicat suivant (voir la question précédente pour la définition de `q2append`):

```
q3([],[]).
q3(L,[H|T]) :- q2append(V,[H|U],L), q2append(V,U,W), q3(W,T).
```

Quelles seront toutes les solutions obtenues par la requête ci-dessous?

```
?- q3([a,b,c],L).
```

Question 4 Scheme [5 points]

Quel sera le résultat affiché par chacun de ces appels de fonction?

```
(let loop ((n 4) (i 2)) (if (zero? n) i (loop (- n 1) (+ i 1))))
```

```
 =>
```

```
(let* ((L1 '(1 2 3 4)) (L2 (car L1))
       (F (lambda (L) L))) (cons (F L2) L1))
```

```
=>
```

```
((lambda (x y) (* (* x y) (* x y))) 2 3)
```

```
=>
```

```
(let ((A '(1 2))) (cond ((null? (cdr A)) '(1))
                        ((list? (cadr A)) '(2))
                        ((list? (cddr A)) '(3))))
```

```
=>
```

```
(cons (car '((1))) '(((8))))
```

```
=>
```

Question 5 fonction Scheme [3 points]

Écrire une fonction Scheme permettant d'échanger le premier et second élément d'une liste.

Exemple

```
> (q5 '(1 2 3 4))
'(2 1 3 4)
> (q5 '(1 2))
'(2 1)
> (q5 '(4))
'(4)
> (q5 '())
'()
```

```
(define (q5 L)

      (cond

        ((null? L) _____________________________________________)


        ((null? (cdr L)) _____________________________________)


        (else ________________________________________________

              ________________________________________________

              ________________________________________________)
      ))
```

Question 6 fonction Scheme [3 points]

La fonction `two-last` permet de retourner les deux derniers éléments d'une liste.
Compléter la définition de cette fonction.

Exemples:

```
> (two-last '(1 2 3 4 5))
'(4 5)
> (two-last '())
'()
> (two-last '(1))
'(1)
```

```
(define (two-last L)

)
```

Question 7 Arbre binaire de recherche Scheme [3 points]

Soit la fonction suivante permettant d'effectuer une recherche dans un arbre binaire de recherche.

```
(define (tree-search x t)
      (cond
        ((null? t) #f)
        ((equal? x (car t)) #t)
        ((< x (car t)) (search x (cadr t)))
        ((> x (car t)) (search x (caddr t)))
        (else #f)))
```

Cette fonction retourne vrai ou faux selon que l'élément spécifié se trouve dans l'arbre ou non.

```
> (tree-search 101
       '(101 (31 (5 () ()) ()) (101 (83 () (97 () ())) ())))
#t
> (tree-search 85
       '(73 (31 (5 () ()) ()) (101 (83 () (97 () ())) ())))
#f
```

Modifier cette fonction de façon à ce que la fonction retourne le sous-arbre dont la racine est l'élément recherché.

```
> (tree-search2 101
       '(101 (31 (5 () ()) ()) (101 (83 () (97 () ())) ())))
'(101 (83 () (97 () ())) ())

> (tree-search2 85
       '(73 (31 (5 () ()) ()) (101 (83 () (97 () ())) ())))
'()
```

Répondre sur la page suivante…

```
(define (tree-search2 x t)

      (cond

        ((null? t) _____________________________________________)


        ((equal? x (car t)) ____________________________________)


        ((< x (car t)) _________________________________________)


        ((> x (car t)) _________________________________________)


        (else ___________________________________________________)
      ))
```

Question 8 Prolog et Scheme [9 points]

a) Concevoir un prédicat Prolog retournant vrai lorsqu'une liste d'entiers est triée en ordre croissant. Pour ce faire vous devez vérifier que chaque élément de la liste est plus petit ou égal à l'élément qui le suit. Par exemple :

```
?- q8([ 2, 5, 8, 8, 9]).
true.
?- q8([ 8, 5, 2, 8, 9]).
false.
```

```
q8([]) :-
```

```
q8([X]) :-
```

```
q8([H|T]) :-
```

b) Concevoir une fonction Scheme retournant vrai lorsqu'une liste d'entiers est triée en ordre croissant. Pour ce faire vous devez vérifier que chaque élément de la liste est plus petit ou égal à l'élément qui le suit. Par exemple :

```
> (q8 '(2 5 8 8 9))
#t
> (q8 '(8 5 2 8 9))
#f


(define (q8 L)

      (cond

        ((null? L) _____________________________________________)



        ((null? (cdr L)) _____________________________________)



        (else ____________________________________________________

                ____________________________________________________

                ____________________________________________________)
      ))
```

c) Concevoir une fonction Go retournant vrai lorsqu'un *slice* d'entiers est trié en ordre croissant. Pour ce faire vous devez vérifier que chaque élément de la liste est plus petit ou égal à l'élément qui le suit.

```
func q8(si []int) bool {
```

Question 9 les méthodes Go [4 points]

Soit le programme Go suivant :

```
package main
import "fmt"

type Reader struct {
     name string
     id int
     books []Book
}

type Book struct {
     title string
     author string
     year int
}

func main() {

     jack:= Reader{"Jack", 123, make([]Book,0,5)}

     nBooks:= jack.Add(Book{"Lord of the rings","JRR Tolkien",1937})
     fmt.Printf("Number of books read= %d\n", nBooks)
     nBooks= jack.Add(
                  Book{"The name of the rose","Umberto Eco",1980})
     fmt.Printf("Number of books read= %d\n", nBooks)

     fmt.Println(jack.books)
}
```

Définir la méthode `Add` permettant d'ajouter un livre à la liste des livres d'un lecteur. Le programme de la page précédente devrait alors afficher le résultat suivant :

```
Number of books read= 1
Number of books read= 2
[{Lord of the rings JRR Tolkien 1937} {The name of the rose Umberto Eco 1980}]
```

Question 10 Go Channels [4 points]

Voici une fonction Go permettant de distribuer un calcul à effectuer sur les éléments d'un tableau en parallélisant la boucle en deux fils concurrents.

```
package main

import "fmt"

func calcul(tab []int, ch chan int) {
     for i, v := range tab {
          ch <- i + v
     }
     close(ch)
}

func main() {
     ch := make(chan int)
     x := []int{1, 2, 3, 4, 5, 6}
     go calcul(x[:3], ch)
     go calcul(x[3:], ch)
     for {
          val, ok := <-ch
          if !ok {
               break
          }
          fmt.Printf("%d ", val)
     }
}
```

Malheureusement, tel qu'écrit ce programme s'arrête avant que les calculs soient terminés. Par exemple :

Essai 1:
```
4 6 8 Success: process exited with code 0.
```

Essai 2:
```
4 6 1 3 5 Success: process exited with code 0.
```

En utilisant la notion de *Channel* en Go, synchroniser tous ces fils de façon à ce que le programme attende la fin des calculs pour se terminer.

Répondre sur la page suivante…

```
func calcul(tab []int, ch chan int) {
     for i, v := range tab {
          ch <- i + v
     }
     close(ch)
}

func main() {
     chA := make(chan int)
     chB := make(chan int)
     x := []int{1, 2, 3, 4, 5, 6}
     go calcul(x[:3], chA)
     go calcul(x[3:], chB)

     for {

          _____________________________________________________

          _____________________________________________________

          _____________________________________________________

          _____________________________________________________

          _____________________________________________________

          _____________________________________________________

          _____________________________________________________

          _____________________________________________________

     }
}
```