

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

## CSI2120 Programming Paradigms

### FINAL EXAM

**Length of Examination: 3 hrs**

**April 26, 2018, 9:30-12:30**

**Professor: Jochen Lang**

**Page 1 of 19**

Family Name: \_\_\_\_\_

Given Names: \_\_\_\_\_

Student Number: \_\_\_\_\_

Signature \_\_\_\_\_

You are allowed one hand-written double-sided letter-sized sheet of notes.

At the end of the exam, when time is up: Stop working and close your exam booklet. Remain silent.

Question	Marks	Out of
1		6
2		5
3		3
4		5
5		4
6		5
7		5
8		5
Total		38

---

**Question 1 Database [6 marks]**

Consider the following Prolog database:

```
% name, game, score
score( 'Emma', 'FIFA18', 3 ).
score( 'Benjamin', 'Minecraft', 387 ).
score( 'Liam', 'The Legend of Zelda', 2200 ).
score( 'Ethan', 'Super Mario Odyssey', 15100 ).
score( 'Ava', 'Minecraft', 410 ).
score( 'Liam', 'Minecraft', 222 ).
score( 'Ava', 'The Legend of Zelda', 1900 ).
```

a) Use `setof/3` to find a sorted list of all highscores in the game `Minecraft`

?- \_\_\_\_\_  
\_\_\_\_\_ .

```
L = [222, 387, 410].
```

b) Use `findall/3` to find a list of all games played by `Liam`

?- \_\_\_\_\_  
\_\_\_\_\_ .

```
L = ['The Legend of Zelda', 'Minecraft'].
```

- c) Complete the predicate `countGames` to determine the number of the occurrence of a game in a list of games.

```
?- countGames( ['FIFA18', 'Minecraft', 'The Legend of Zelda',  
               'Super Mario Odyssey', 'Minecraft', 'Minecraft',  
               'The Legend of Zelda'], 'Minecraft', N).  
N = 3.
```

```
countGames( [], _, 0 ) :- _____ .
```

```
countGames( [O|T], G, C ) :- _____
```

```
_____ .
```

```
countGames( [G|T], G, C ) :- _____
```

```
_____ .
```

---

d) Consider the following predicates?

```
popular( P ) :- findall( G, highscore( _, G, _ ), L ),  
                setof( G, N^S^highscore( N, G, S ), LL ),  
                findMax( L, LL, P, _ ).
```

```
max( MG, MC, _, CC, MG, MC ) :- MC > CC, !.
```

```
max( _, _, G, C, G, C ) :- !.
```

```
findMax( _, [], 'None', 0 ) :- !.
```

```
findMax( L, [G|OG], M, C ) :- findMax( L, OG, MG, MC ),  
                              countGames( L, G, CC ),  
                              max( MG, MC, G, CC, M, C ).
```

What does the following query produce?

```
?- popular(P).
```

---

---

**Question 2 Scheme List Processing [5 marks]**

Consider the following example output:

```
(contain '(a 7 9 10) '(5 a c 7 10))
```

```
⇒ ((a 7 10) (9))
```

Complete the auxiliary procedure `containAux` for `contain` that accept two lists as inputs and produces a list of two lists as outputs. The first input list are elements to be tested if they are contained in the second input list. The first list in the output list is a list of elements that are contained in the second input list while the second list in the output list are the elements that are not contained in the second input list.

```
(define (contain S L)
  (cond
    ((or (not (list? S)) (not (list? L))) error)
    ((or (null? S) (null? L) ) (list '() '()))
    (#t (containAux S L '() '()))))
```

```
(define (containAux S L FS FO)
```

```
  (cond
```

```
    ((null? S) _____ )
```

```
    ((member (car S) L) _____
```

```
    _____
```

```
    _____) )
```

```
  (#t _____
```

```
    _____
```

```
    _____)))
```

---

**Question 3 Scheme Calculation [3 marks]**

The area of a rectangle with the lower-left corner A and the upper right corner B is given by

$$area = (B_x - A_x) (B_y - A_y)$$

The corners A, B can be represented in Scheme as pairs, e.g.,  $(-2 \ . \ 2)$   $(1 \ . \ 5)$

And then the following procedure call will calculate the area.

```
(area (cons -2 2) (cons 1 5))
```

⇒ 9

Complete the procedure `area` to calculate the area:

```
(define (area A B)
```

---

---

---

---

---

---

---

---

---

---

---

**Question 4 Scheme let, let\*, letrec, named let [5 points]**

Consider the following definition for the function `fct`

```
(define (fct x y)
  (let ((diff (- x y)))
    (let ((diff-squared (* diff diff)))
      (let ((diff-cubed (* diff-squared diff)))
        diff-cubed))))
```

a) What will be the result of the following call?

```
(fct 5 7)
```

---

b) Rewrite this function by replacing the `let` calls by a call to the `let*` function

```
(define (fct x y)
```

---

---

---

---

---

---

---

---

Consider the following definition for the function `abc`

```
(define (abc x y)
  (let loop ((i x) (s 0))
    (if (< i y)
        (loop (+ i 1) (+ s i)) s)))
```

c) What will be the result of the following call?

```
(abc 3 7)
```

---



---

Consider the following definitions for the function `len` that calculates the length of an input list.

```
(define (len L)
  (if (list? L)
      (lengthAux L)
      'error ))

(define (lengthAux L)
  (if (null? L)
      0
      (+ 1 (lengthAux (cdr L))))))
```

d) Rewrite `len` without an auxiliary function (`lengthAux`) but using a `letrec` instead (note that you will have to define the binding `lenlet`):

```
(define (len L)
  (if (list? L)
      (letrec (_____
                _____
                _____
                _____
                _____)
        (lenlet L))
      'error))
```

**Question 5 Go Routines [4 marks]**

Consider the following go main routine which applies the functions `fourier` in `go routines` and collects their calculation.

```
package main

import "fmt"
import "runtime"
import "math"
import "math/rand"

type Series struct {
    a, b float64
}

func main() {
    runtime.GOMAXPROCS(3)

    data := make(chan float64)
    defer close(data)
    var c [32]Series
    TP := 4

    for t := 0; t < TP; t++ {
        for k := 0; k < 32; k++ {
            c[k].a = rand.Float64()/32.0
            c[k].b = rand.Float64()/32.0
        }
        go fourier(c, t, TP, data)
    }
    // Below the results from all of the go routines need to be
    // received and printed to the console. The program is to exit
    // when all data is received.
```

```
}

```

The function `fourier` currently accepts a fixed size array `c` of size 32.

```
func fourier(c [32]Series, t, TP int, out chan float64) {
    res := c[0].a
    for n := 1; n < 32; n++ {
        res += c[n].a*math.Sin(2.0*math.Pi*float64(t)/float64(TP))
        + c[n].b*math.Cos(2.0*math.Pi*float64(t)/float64(TP))
    }
    out <- res
    return
}
```

Change the function to work correctly with an arbitrary-sized slice.

```
func fourier(c _____,
             t, TP int, out chan float64) {
    res := c[0].a

    for _____{
        res += c[n].a*math.Sin(2.0*math.Pi*float64(t)/float64(TP))
        + c[n].b*math.Cos(2.0*math.Pi*float64(t)/float64(TP))
    }
    out <- res
    return
}
```

---

**Question 6 N-ary Tree Prolog [5 marks]**

Consider the following n-ary tree definition:

```
t(2,-3,[t(5,1,[t(7,2,[[]])]),
        t(-3,4,[[]]),
        t(2,4,[t(-1,1,[[]]),
               t(-2,3,[[]])])])
)
```

An in-order traversal is defined as follows:

```
traverse(t(X,Y,[[]])) :- write(X), tab(1), write(Y), nl.

traverse(t(X,Y,[H|T])) :- traverse(H),
    write(X), tab(1), write(Y), nl,
    traverseTail(T).

traverseTail([]) :- !.

traverseTail([H|T]) :- traverse(H),
    traverseTail(T).
```

*Please see next page!*

---

What does the following traversal print?

```
?- traverse(t(2,-3,
            [t(5,1,[t(7,2,[[]])]),
             t(-3,4,[[]]),
             t(2,4,[t(-1,1,[[]]),
                   t(-2,3,[[]])])])
           ).
```

---

---

---

---

---

---

---

---

---

The predicate `findPoint` is true if a point is in the tree.

```
findPoint(t(U,V,_),U,V) :- !.  
findPoint(t(_,[H|_]),U,V) :- findPoint(H,U,V).  
findPoint(t(_,[_|T]),U,V) :- findPointTail(T,U,V).
```

Complete the auxiliary predicate `findPointTail` below:

```
findPointTail([H|_],U,V) :-
```

```
_____  
_____.  
_____.
```

```
findPointTail([_|T],U,V) :-
```

```
_____  
_____.  
_____.
```

**Question 7 N-ary Tree Scheme [5 marks]**

Consider the following n-ary tree definition:

```
(define pTree (list (cons 2 -3)
                    (list (list (cons 5 1) (list (list (cons 7 2) '()))
                                (list (cons -3 4) '()))
                          (list (cons 2 4) (list (list (cons -1 1) '())
                                                  (list (cons -2 3) '()))))))))
```

A tree traversal is defined below:

```
(define (traverse T)
  (cond
    ((null? T) '())
    ((null? (cdr T)) (car T))
    (#t (cons (car T) (traverseTail (cdr T))))))

(define (traverseTail T)
  (if (null? T)
      '()
      (append (traverse (car T)) (traverseTail (cdr T)))))
```

What does the function traverse evaluate to:

```
(traverse pTree)
```

⇒ \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

---

The function `findPoint?` evaluates to true if a point is in the tree.

```
(define (findPoint? T U V)
  (cond
    ((null? T) #f)
    ((and (equal? (caar T) U) (equal? (cadr T) V)) #t)
    ((null? (cdr T)) #f)
    (#t (findPointTail? (cadr T) U V))))
```

Complete the auxiliary function `findPointTail?` below:

```
(define (findPointTail? TL U V)
  (if (null? TL)
```

```
    _____
    _____
    _____
    _____
    _____) )
```



---

**Question 8 N-ary Tree Go [9 marks]**

Consider the following main program:

```
package main

import "fmt"

type nTree struct {
    x, y      int
    children []nTree
}

func main() {
    tree := nTree{2, -3,
        []nTree{{5, 1, []nTree{{7, 2, nil}}},
            {-3, 4, nil},
            {2, 4, []nTree{{-1, 1, nil},
                {-2, 3, nil}}}}}

    tree.traverse()
    u, v := -1, 1
    if tree.findPoint(u, v) {
        fmt.Printf("Found: %d %d \n", u, v)
    }
}
```

---

Complete the method `traverse` below such that it prints the points in-order to console.

```
func (t *nTree) traverse() {  
    if t.children == nil || len(t.children) == 0 {  
        fmt.Printf("%d %d \n", t.x, t.y)  
    } else {
```

```
        for _____ {
```

```
            _____  
            _____  
            _____  
            _____  
            _____
```

```
        }
```

```
    }
```

```
    return
```

```
}
```

