

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 24, 2015, 9:30-12:30

Professor: Jochen Lang

Page 1 of 13

Family Name: _____

Other Names: _____

Student Number: _____

Signature _____

You are allowed one 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		4
2		3
3		5
4		3
5		3
6a		3
6b		3
7		6
8		4
9		4
Total		38

Question 1 Prolog Rules and Facts[4 marks]

The curriculum committee is working on new pre-requisites for co-op. The committee is considering a student eligible for co-op if

- 1) Student is at least in second year,
- 2) Student has passed ITI1121 with at least D+,
- 3) Student has passed CSI2120 with at least B, and
- 4) Student is fulltime.

Design a rule eligible given the following facts and rules:

% Name, Year, full-time/part-time, [(Course, Letter Grade), ...]

```
student( jane, 2, ft, [ (iti1120, 'B'), (iti1121, 'B+'),
  (csi2120, 'A+'), (csi2372, 'A-') ] ).
```

```
student( joe, 3, pt, [ (iti1120, 'A'), (iti1121, 'B'), (csi2120, 'C'),
  (csi2372, 'C'), (csi3105, 'F') ] ).
```

```
student( mary, 1, ft, [ (iti1100, 'A+'), (iti1120, 'A+'),
  (iti1121, 'A+') ] ).
```

% true if Course is listed in L with at least MinGrade

```
grade(Course, MinGrade, [ (Course, Grade) | _ ] ) :- Grade @=< MinGrade.
```

```
grade(Course, MinGrade, [_ | L] ) :- grade(Course, MinGrade, L).
```

```
eligible( Name ) :-
```

Question 2 Prolog List Processing [3 marks]

The predicate `deleteBack` is to delete the last occurrence of a queried element `R` in a list. Complete the predicate below.

Example:

```
?- deleteBack( 6, [3,6,5,6,7], L ).
```

```
L = [3,6,5,7]
```

% boundary case

```
deleteBack(_____, _____,
            _____, _____ ) .
```

```
deleteBack( R, [R|LI], [R|A], L0 ) :-
    \+member(R, LI), !,
    deleteBack( R, LI, A, L0 ).
```

```
deleteBack(_____, _____,
            _____, _____ ) :-
    deleteBack(_____,
                _____,
                _____,
                _____) .
```

% `R` is element to be removed, `LI` is input list, `LO` is result

```
deleteBack( R, LI, LO ) :- deleteBack( R, LI, _, LO ).
```

Question 3 Scheme Syntactic Forms [5 marks]

What is the return of the following calls?

```
((lambda (x y) (+ x y)) 1 2)
```

=>

```
(let ((x 1) (y 2)) (+ x y))
```

=>

```
((lambda (x y) (+ x y) (- x y)) 2 1)
```

=>

```
(let* ((x 1) (y (+ x 1))) (- y x))
```

=>

```
(let ((f (lambda (x y) (- (+ x y) 1))) (x 1) (z 2)) (f 3 z))
```

=>

Question 4 Scheme Lists [3 marks]

Complete the following function calls with a single function.

Example

```
(define L '(1 2))  
(cadr L)  
=> 2
```

```
(define L '( w x y z ))
```

```
(_____ L)
```

=> 'x

```
(define L '((w x) (y z)))
```

```
(_____ L)
```

=> '(x)

```
(define L '((w (x (y z))) a))
```

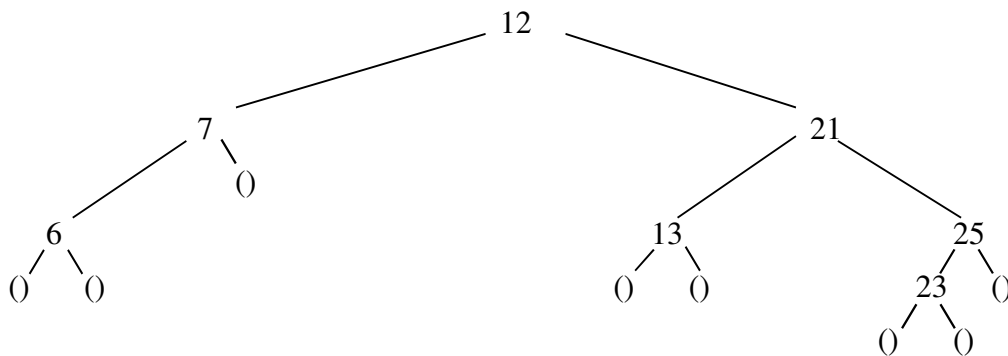
```
(_____ L)
```

=> x

Question 6 Scheme BST Tree

a) Below a binary search tree is drawn. Give its scheme representation as discussed in class. The `tree?` predicate is shown as a reminder. [3 marks]

```
(define tree?
  (lambda (t)
    (cond
      ((not (list? t)) #f)
      ((null? t) #t)
      ((not (= (length t) 3)) #f)
      ((not (tree? (cadr t))) #f)
      ((not (tree? (caddr t))) #f)
      (else #t)
    )
  ))
```



Define the tree.

```
(define T '(_____
               _____
               _____))
```

b) Complete the implementation of the function `sumPath` which sums the numbers, from the root to a given node including the node. [3 marks]

Example:

> (sumPath 13 T) ; 12 + 21 + 13

46

> (sumPath 7 T) ; 12 + 7

19

```
(define sumPath
  (lambda (x t)
    (define search
      (lambda (x t s)
        (cond
          ((null? t) #f)
          ((equal? x (car t))

           (_____)))
          ((precedes? x (car t))

           (_____)))
          ((precedes? (car t) x)

           (_____)))
        (else #f)
      )))
    (if
      (not (tree? t))
      (list 'not-a-tree t)
      (search x t 0)
    )))
```


Question 7 Python Data [6 marks]

Give the output of the following python commands

```
>>> a = [ 2*x for x in range(10) if x % 3 == 0]
>>> a
```

```
>>> d = { 3:'hello', 4:'world', 5:'from', 12:'uOttawa'}
>>> d[4]
```

```
>>> t = [-1,-2,-3,-4]
>>> t[-1]
```

```
>>> k = ['a', 'b', [1, 2, 3], 'c']
>>> k[1:3]
```

continued on next page ...

```
>>> L = [ x*y for x in range(1,3) for y in 'abc' ]  
>>> print(L)
```

```
>>> r1 = [ 1, 2, 3]  
>>> r2 = r1  
>>> r2[1] = 5  
>>> r1
```

Question 8 Go methods [4 marks]

Complete the program to produce the output

Price: \$ 32.00

```
package main

import "fmt"

type Flower struct {
    name  string
    color string
    price float64
}

type Tree struct {
    name          string
    height        float64
    pricePerMeter float64
}

type Item interface {
    getPrice()
}

func main() {
    gardenStore := [2]Plant{Tree{"maple", 1.5, 20.0},
                           Flower{"tulip", "red", 2.0}}

    price := 0.0
    for _, p := range( gardenStore ) {
        price += p.getPrice()
    }
    fmt.Printf("Price: $%6.2f", price)
}
```

Please answer on **next** page.

Partial method definition (to be completed)

```
func (_____) getPrice() _____ {  
  
    _____  
  
}
```

Second method definition (to be completed **only** if needed)

```
func (_____) getPrice() _____ {  
  
    _____  
  
}
```

Question 9 Go Routines and Channels [4 points]

Complete `main` function and the function `sendString` for the main program to receive and print the animal strings and then exit. The `animals` slice has here a size of 4 but your solution must work with any size of slice.

```
package main
```

```
import     "fmt"
```

```
func main() {  
    animals := []string{"coyote", "fox", "hare", "deer"}  
    ch := sendString( animals)  
    for {  
  
        _____  
  
        if !ok {  
            break;  
        }  
        fmt.Printf("%s ", str)  
    }  
}
```

```
func sendString(strArr []string) chan string {  
    ch := make(chan string)  
    go func() {  
  
        _____  
  
        _____  
  
        _____  
  
        _____  
  
    }()  
    return ch  
}
```