COMP348Miderm Answers

Principles of Programming Languages (Concordia University)

COMP348 - MIDTERM ANSWERS

QUESTION 1

Use Arrays.sort(Object[] obj, Comparator<? super T> c) to compare two Cars.
(Hint: Can use ArrayUtils.indexOf(Object[] obj, Object toFind))

```
public interface Car {
   public final String[] MODELS = { ... };
   public String getModel();
   public String getMake(); // i.e. Toyota.
   public int getYear();
}
```

1a)
```
// Anonymous class to sort by make.
Array.sort(cars, new Comparator<Car>() {
                        public int compare(Car c1, Car c2) {
                                return ("" + c1.getMake().compareTo(c2.getMake()) + ""); // The
quotes prevent a NullPointerException.
                        }
                });


// Lambda expression to sort by make.
Arrays.sort(cars, (Car c1, Car c2) -> ("" + c1.getMake().compareTo(c2.getMake()) + ""));
```

1b)
```
// Lambda expression to sort by make.
Arrays.sort(cars, (Car c1, Car c2) -> (ArrayUtils.indexOf(Car.MODELS, c1.getModel()) -
ArrayUtils.indexOf(Car.MODELS, c2.getModel())));
```

1c)
```
// Same concept but compare two attributes.
```

QUESTION 2

2a)
Does chk(X, a(b, c), d(Y, [H|T])) unify with the following items?

i) X
X = chk(X, a(b, c), d(Y, [H|T]))

ii) chk(a, Y, Z)
a = X
a(b, c) = Y
Z = d(Y, [H|T])

iii) chk(X, a(b, X), d(a, b))
b cannot be an atom AND a list.

iv) chk(X, a(b, c), d(a, [b]))
X = X
a(b, c) = a(b, c)
Y = a
[b] = [H|T]

v)[X|Y]
The given clause is not a list.

2b)
Show the complete steps for unification and resolution for the following prolog code:

add(X, L, [X|L]).
?- add(a, [b|[c]], L).

X = a
L' = [b|[c]]
[X|L'] = L
L = [a, b, c]

QUESTION 3

3a) Represent the circuit in prolog.

circuit(A, B, S, C) :- or(A, B, AoB), and(A, B, C), inv(C, Cinv), and(AoB, Cinv, S).

3b) Output of the following code.

```
f(0, 1).
f(1, 1).
d(N, R) :- N1 is N-1, N2 is N-2, f(N1, R2), R is R1 + R2.
runme :- forall(member(X, [0, 1, 2, 3, 4, 5]), write(Y), nl).
```

i) 1, 1, 2, 3, 5, 8
ii) Fibonnaci sequence.

QUESTION 4

4a) Write a procedure called find_at/3 to find the K'th element of a given list.

```
find_at([H|_], 1, H) :- !.
find_at([H|T], N, X) :- find_at(T, N1, X), N is N1+1.
```

4b)

```
?-find_at([1, 2, 3], -1, K).
false

?-findall(K, find_at([a, b, c, d, e], _, K), O).
O = [1, 2, 3, 4, 5]
```

4c)

```
parent(john, sally).
parent(jim, mike).
parent(carol, john).
parent(carol, sue).
parent(sally, jim).
parent(jim, bob).
malelist([john, jim, mike, bob]).
femalelist([sally, carol, sue]).
```

```
i) mother(X, Y) :-        % X is the mother of Y
        femalelist(L), member(X, L), parent(X, Y).

ii) father(X, Y) :-       % X is the father of Y
        malelist(L), member(X, L), parent(X, Y).
```

```
iii) sibling(X, Y) :-        % X and Y share a common parent
        parent(Z, X), parent(Z, Y), X /= Y.

iv) couple(X, Y) :-        % X and Y share a child
        parent(X, Z), parent(Y, Z), X /= Y.
```

QUESTION 5

5a)

```
(car (cdr (list '(1 3) '(2 4) '(5 7 6))
```

Output: (2 4)

5b)

```
(let ((x 2)))
(let* ((x 5) (y 10)) (if (> (/ y x) (/ 8 4)) (+ x y) (* y x)))
```

Output: 50

5c)

```
(< 1 2 5 0)
```

Output: nil

5d)

```
(list '(a b) (list '(a b) (list '(c) 'd)))
```

Output: ((a b) ((a b) ((c) d)))

5e)

```
(let ((a '(a))) (append a '(a))
```

Output: (a a)

QUESTION 6

6a) Write a function that receives a list and duplicates its elements such that:

```
> (dup1 '(1 (2) (3 4)))
(1 1 (2) (2) (3 4) (3 4))

(defun dup1 (lst)
        (cond
                ((null lst)
```

```
                lst
        )
        (t
                (cons (car lst) (cons (car lst) (dup1 (cdr (lst)))))
        )
    )
)
```

6b) Write a function that receives a list and flattens it

```
> (flatten '((1) (2 (3)) (((4 5))))
(1 2 3 4 5)
```

6c) Write a function that skips a number of elements in a list.

```
> (skip '(1 2 3 4) 2)
(3 4)
> (skip '(1 2 3 4) -2)
(1 1 1 2 3 4)
```

```
(defun skip (lst n)
    (cond
        ((= n 0)
                lst
        )
        ((> n 0)
                (skip (cdr lst) (- n 1))
        )
        ((< n 0)
                (cons (cons lst) (skip lst (+ n 1)))
        )
    )
)
```

QUESTION 7

Write a function to check if a given list is a tree (empty list is a tree and all children nodes have values).

```
(defun istree (lst)
```

```
	(cond
		((atom lst)
			nil
		)
		((null lst)
			T
		)
		(t
			(istreelist (cdr lst))
		)
	)
)

(defun istreelist (lst)
	(cond
		((null lst)
			T
		)
		(t
			(and (istree (car lst)) (istreelist (cdr lst)))
		)
	)
)
```

QUESTION 8

Calculate the sum from 0 to n.

8a) Use recursion:

```
(defun sum (n)
	(cond
		((= n 0)
			0
		)
		(t
			(+ (sum (- n 1)) n)
		)
	)
)
```

8b) Use iteration:

```
(defun sum (n)
	(cond
		((= n 0)
			0
		)
		(t
			(let ((s 0))
				(dotimes (i (+ n 1) s)
					(setq s (+ s i))
				)
			)
		)
	)
)
```