COMP 348
PRINCIPLES OF PROGRAMMING LANGUAGES

Tutorial #6

Functional Programming with LISP (Continued)

CONTROL FLOW: MULTIPLE SELECTION

- Multiple selection can be formed with a **cond** expression which contains a list of clauses where each clause contains two expressions, called condition and answer. Optionally, we can have an **else**.
- Conditions are evaluated sequentially.

```
( cond (question  answer)
       ...
       (else  answer)   ; Optional.
  )
```

- We can also use t (true) in place of else.

DEFINING FUNCTION F : N → LISTS(N)

- Suppose we need to define the function f : N→lists(N) that accepts an integer argument and returns a list, such that

$$f(n) = \langle n, n-1, \dots, 0\rangle$$

- We can therefore define f recursively by

$$f(0) = \langle 0\rangle.$$
$$f(n) = cons(n, f(n-1)), \ \ for \ \ n > 0.$$

- We can unfold this definition for f(3) as =>

$$\begin{aligned} f(3) &= cons(3, f(2)) \\ &= cons(3, cons(2, f(1))) \\ &= cons(3, cons(2, cons(1, f(0)))) \\ &= cons(3, cons(2, cons(1, \langle 0\rangle))) \\ &= cons(3, cons(2, \langle 1, 0\rangle)) \\ &= cons(3, \langle 2, 1, 0\rangle) \\ &= \langle 3, 2, 1, 0\rangle. \end{aligned}$$

DETERMINING SET UNION

- Consider function **setunion** which takes as its arguments two lists t1 and t2 representing sets and returns the set union.

  Base cases:

  1. If t1 is empty, then return t2.
  2. If t2 is empty, then return t1.

  Recursive cases:

  1. If the head of t1 is a member of t2, then ignore this element and recur on the tail of t1, and t2.
  2. If the head of t1 is not a member of t2, return a list which is the concatenation of this element with the union of the tail of t1 and t2.

DETERMINING SET UNION (EXAMPLE)

```
CL-USER 1 > (defun setunion (t1 t2)
               (cond
                ((null t1) t2)
                ((null t2) t1)
                ((member (car t1) t2)(setunion (cdr t1) t2))
                (t (cons (car t1) (setunion (cdr t1) t2)))))
SETUNION

CL-USER 2 > (setunion '(1 3) '(1 2 3 4))
(1 2 3 4)
```

DETERMINING SET INTERSECTION

- Consider function **setintersection** which takes as its arguments two lists t1 and t2 representing sets, and returns a new list representing a set which forms the intersection of its arguments.

  Base cases:

  If either list is empty, then return the empty set.

  Recursive cases:

  1. If the head of t1 is a member of t2, then keep this element and recur on the tail of t1 and t2.
  2. If the head of t1 is not a member of t2, ignore this element and recur on the tail of t1 and t2.

DETERMINING SET INTERSECTION (EXAMPLE)

```
CL-USER 1 > (defun setintersection (t1 t2)
              (cond
               ((null t1) '())
               ((null t2) '())
               ((member (car t1) t2)
                (cons (car t1)(setintersection (cdr t1) t2)))
               (t (setintersection(cdr t1) t2))))
SETINTERSECTION
CL-USER 2 > (setintersection '(1 2 3) '())
NIL
CL-USER 3 > (setintersection '(1 2 3) '(2 4 5 6))
(2)
```

DETERMINING SET DIFFERENCE

- Consider function **setdifference** which takes as its arguments two lists t1 and t2 representing sets and returns the set difference.

  Base cases:

  If t1 is empty, then return the empty set. If t2 is empty, then return t1.

  Recursive cases:

  1. If the head of t1 is a member of t2, then ignore this element and recur on the tail of t1, and t2.
  2. If the head of t1 is not a member of t2, keep this element and recur on the tail of t1 and t2.

DETERMINING SET DIFFERENCE (EXAMPLE)

```
CL-USER 1 > (defun setdifference (t1 t2)
              (cond
               ((null t1) '())
               ((null t2) t1)
               ((member (car t1) t2)(setdifference (cdr t1) t2))
               (t (cons (car t1) (setdifference (cdr t1) t2)))))
SETDIFFERENCE
CL-USER 2 > (setdifference '() '(1 4 6))
NIL
CL-USER 3 > (setdifference '(1 2 3 4 5) '(2 4))
(1 3 5)
```

EXERCISE#1

1/ Consider the binary tree in Fig.1 , translate this representation into lisp

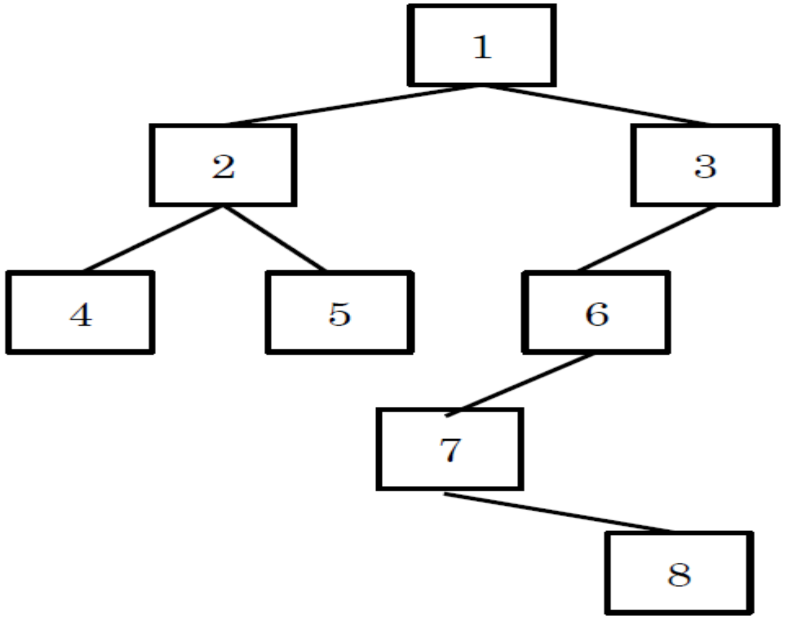

Fig.1

EXERCISE#2

2a/ Consider the binary tree in Fig.1., obtain the root of the tree

?

2b/ Consider the binary tree in Fig.1., obtain the left subtree

?

2c/ Consider the binary tree in Fig.1., obtain the right subtree

?

DETERMINING IF THE INPUT LIST IS SORTED

Write a function **is-sortedp,** which returns True or False on whether or not its list argument is sorted.

```
CL-USER 1 > (defun is-sortedp (lst)
               (cond ((or (null lst)(null (cdr lst))) t)
                ((< (car lst) (car (cdr lst)))(is-sortedp (cdr lst)))
                 (t nil)))
IS-SORTEDP
CL-USER 2 > (is-sortedp '(0 1 3 2 4 5))
?
CL-USER 3 > (is-sortedp '(0 1 3 4 7))
?
```

PLACE ONE ELEMENT IN ITS PROPER POSITION

Write a function **bubble**, which performs one iteration, thus placing one element in its proper position.

```
CL-USER 1 > (defun bubble (lst)
              (cond ((or (null lst) (null (cdr lst))) lst)
               ((< (car lst) (car (cdr lst)))
                 (cons (car lst) (bubble (cdr lst))))
               (t (cons (car (cdr lst))
                  (bubble (cons (car lst) (cdr (cdr lst))))))))
BUBBLE
CL-USER 2 > (bubble '(0 1 3 2 4))
?
CL-USER 3 > (bubble '(0 5 3 4 2 1 6))
?
```

SORT A LIST IN ASCENDING ORDER

Write a function **bubble-sort**, to sort a list in ascending order by using function **bubble** and **is-sortedp.**

```
CL-USER 1 > (defun bubble-sort (lst)
               (cond ((or (null lst) (null (cdr lst))) lst)
                 ((is-sortedp lst) lst)
                 (t (bubble-sort (bubble lst)))))
BUBBLE-SORT
CL-USER 2 > (bubble-sort '(0 4 3 2 1 5))
?
CL-USER 3 > (bubble-sort '(0 3 5 2 1 7 6))
?
```

BINARY-SEARCH FUNCTION

Provide the result of call of the following BINARY-SEARCH function:

```
CL-USER 1 > (defun binary-search (lst elt)
                 (cond ((null lst) nil)
                  ((= (car lst) elt) t)
                  ((< elt (car lst)) (binary-search (car (cdr lst)) elt))
                  ((> elt (car lst))
                     (binary-search (car (cdr (cdr lst))) elt))))
BINARY-SEARCH
CL-USER 2 > (binary-search '(7 (1 ( ) (2 ( ) ( ) ) ) (8 ( ) (9 ( ) ( ) ) ) ) 9)
?
```