Notes lecture 8

1. **Other list constructors:**

* **REVERSE subr 1 (l): l** = reverse of the list (at superficial level)
* **LENGTH subr 1 (l): n** = number of elements at superficial level

1. **Predicates:**

* any value other than NIL has the meaning of true
* **ATOM subr 1 (e): T, NIL**
  + For NIL returns T
* **LISTP subr 1 (e): T, NIL** 
  + For NIL returns T
* **EQ subr 2 (e e): T, NIL**
  + T if the args are the **same identical object**
* **EQL subr 2 (e e): T, NIL**
  + T if
    - its arguments **are EQ**
    - they are **numbers** of the **same type** with the **same value**
    - they are **character** objects that represent the **same character**
* EQUAL subr 2 (e e): T, NIL
  + T if they have the **same structure** (case insensitive for characters)
* EQUALP subr 2 (e e): T, NIL
  + T if
    - its arguments are **EQUAL**
    - **characters** of **same value** (case insensitive)
    - **numbers** of **same value** (can be of different types)
    - have **components** that are all **EQUALP**

1. **Predicates for lists:**

* **NULL subr 1 (e): T, NIL**
  + T if **empty list** or **null atom**
* **MEMBER subr 2 (e l): l**
  + check the inclusion of an S-expression e in a list l (at superficial level)
  + returns the list from first appearance of e in l (like strstr)

1. **Predicates for numbers**

* **NUMBERP subr 1 (e): T, NIL**
  + Check if it’s a number or not.
* **ZEROP subr 1 (n): T, NIL**
  + Check if n is the number 0 or not. If the argument is not evaluated by number the result is undefined or error.
* **PLUSP subr 1 (n): T, NIL**
  + Check if n is a strictly positive number.
* **MINUSP subr 1 (n): T, NIL**
  + Check if n is a strictly negative number.

1. **Arithmetic operations**

* **+ - \* /** 🡪 variable no. of params
* **1+ subr 1 (n): n** = inc
* **1- subr 1 (n): n** = dec
* **MIN/MAX** 🡪 variable no. of params

1. **Logical operations**

* **(NOT e)** = **T** if e is evaluated at NIL
* **AND fsubr 0, ... (... e ...): e**
  + **NIL** if NIL found throughout args
  + **e** = value of the last arg
* **OR fsubr 0, ... (... e ...): e**
  + **E** = first value other than NIL
  + **NIL** otherwise
* Obs.: AND & OR 🡪 lazy evaluation

1. **Relational operators for numbers**

* **= < <= > >=**
* All of them with 2 args

1. **Defining user functions. DEFUN function**

* **(DEFUN fname (p1 p2 ... pn)**

**(form1)**

**…**

**(formM)**

**)**

* **(fname arg1 arg2 ... argn)**
* the evaluation of the call proceeds as follows:

1. **args** are evaluated
2. formal params are **bound** to the values obtained at 1
3. each **form** is evaluated in order
4. the **returned value** is the value of the last form
5. formal params are **unbound**