ФИЛП 18.02

(defun average (x y) (/ (+ x y) 2.0))

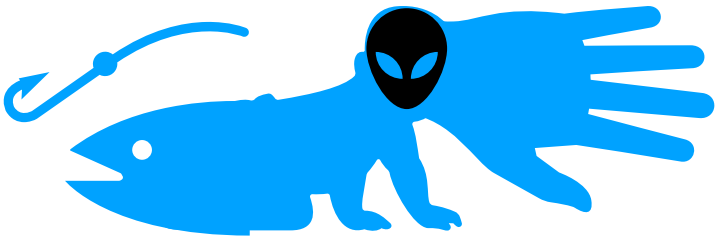
Eval начинает обрабатывать выражение которое попадает ей на вход

3 Лаба - символьные атомы.

Представление в памяти символьных атомов

Символьный атом представляется в памяти 5 указателями. У каждого символьного атома есть окружение и время работы.

Name
Value
Function
Property
Package



В системе есть пакет - область памяти куда вносится информация о тех символьных атомах которые система знает(распознает). В пакете хранится информация про все символьные атомы которые имеют смысл. В качестве символьного атома может быть и другой символьный атом и s-выр.

Символьный атом A может быть связан с определением функции, но может быть связан и со значением. В зависимости от контекста, будет работать вычисление по формуле.

Свойство - список состоящий их двухэлементных списков - имя свойства, значение свойства.
((Name1 value1)...(nameN valueN))

Специальные функции

Базисная функция **cond** имеет в составе произвольное количество двухэлементных списков. Если все тесты возвращают nil то cond вернет nil.
(**if** test T_body F_body)
(**not** s-выр)
(**and** arg1 arg2 arg3 .... argN) работает до тех пор пока не будет очевиден результат. Если какой-то аргумент не nil а true то функция закончит работу.
(**or** arg1 arg2 arg3 .... argN) работает до тех пор пока не будет очевиден результат. Если какой-то аргумент не nil а true то функция закончит работу.

```
(defun how_alike (x y)'
        (cond ((or (= x y)(equal x y )) 'the_same)
                ((and (oddp x)(oddp y) 'both_odd)
                ((and (evenp x)(evenp y) 'both_even)
                (T      'difference)))
```

Существует возможность установить переменной значение а потом ипользуяю переменную получить результат. Если это параметры которые при вызове функции меняются, то ок. Но иногда надо в течение работы функции поменять значение переменной. В lisp это называется let-формы. Это локальные(временные) переменные. У let-формы 2 аргумента, первый-список из 2элементных списков, второй-тело.

```
Let ((var1 value1)
        (var2 value2)
        (varN valueN)
                body)
```

Операции со списками

Функции можно разделить на два типа структуру разрушающие и не разрушающие. Те которые не разрушают работают медленнее.
(append list1 list2) list1 (A B C) list2 (D E)

Функция **reverse** не создает копию а работает по верхнему уровню списочных ячеек.
last - доступ к последней спусковой ячейке верхнего уровня.
nth - голову n-й спусковой ячейки.
nthcdr - хвост списка верхнего уровня начиная с n-й списковой ячейки, нумерация с нуля.
length - количество ячеек на верхнем уровне.

Чтобы удалить элемент из списка надо чтобы система его нашла, для этого надо чтобы она умела сравнивать. **Eql** не сравнивает списки.
(**remove** element lst)
(remove 'a '(b a c)) → (b c)
(remove '(a b) '((a b) c d)) → ((a b) c d)
(remove '(a b) '((a b) c d): test #'equal) → (c d)

(member element list) проходит только по верхнему уровню и сравнивает элементы с искомым. Возвращает nil либо хвост оставшегося списка. Внутри функции membrer тоже используется equal, её тоже надо дорабатывать как remove.

Есть функция которая как sed:
(rplaca lst el) car указатель соответствующей ячейки переустанавливает на него
(rplacd lst el) cdr указатель соответствующей ячейки переустанавливает на него