1.3) Результаты вычисления выражений

```
(caadr '((blue cube)(red pyramid))) ⇔
⇔(car (car (cdr  '((blue cube)(red pyramid)) ))) ⇔
⇔(car (car '((red pyramid)) ))⇔
⇔(car '(red pyramid)) ⇔ red

(cdar '((abc)(def)(ghi))) ⇔
⇔ (cdr (car '((abc)(def)(ghi))))⇔
⇔ (cdr '(abc)) ⇔ Nil

(cadr '((abc)(def)(ghi))) ⇔
⇔ (car (cdr '((abc)(def)(ghi))))⇔
⇔ (car '((def)(ghi))) ⇔ (def)

(caddr '((abc)(def)(ghi))) ⇔
⇔ (car (cdr (cdr '((abc)(def)(ghi)))))⇔
⇔ (car (cdr '((def)(ghi)) )) ⇔
⇔ (car '((ghi)) ) ⇔ (ghi)
```

1.4) Результаты вычисления выражений

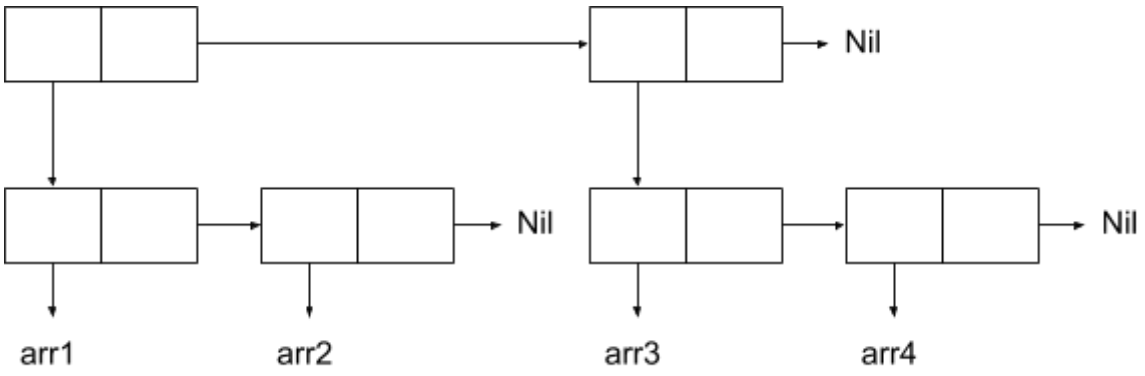
```
(list 'Fred 'and Wilma) = (Fred and Wilma's_value)
(list 'Fred '(and Wilma)) = (Fred (and Wilma))
(cons Nil Nil) = (Nil.Nil) = (Nil) = (())
(cons T Nil) = (T.Nil) = (T)
(cons Nil T) = (Nil. T)
(list Nil) = (Nil) = (())
(cons (T) Nil) = error T - неопределенная функция
(list '(one two) '(free temp)) = ((one two)(free temp))

(cons 'Fred '(and Wilma)) = (Fred. (and Wilma)) = (Fred and Wilma)
(cons 'Fred '(Wilma)) = (Fred.(Wilma)) = (Fred Wilma)
(list Nil Nil) = (Nil Nil) = (()())
(list T Nil) = (T Nil) = (T ())
(list Nil T) = (Nil T) = (() T)
(cons T (list Nil))= (T.(Nil)) = (T Nil) = (T ())
(list (T) Nil) = error T - неопределенная функция
(cons '(one two) '(free temp)) = ((one two).(free temp)) =
= ((one two) free temp)
```

1.5) Функции

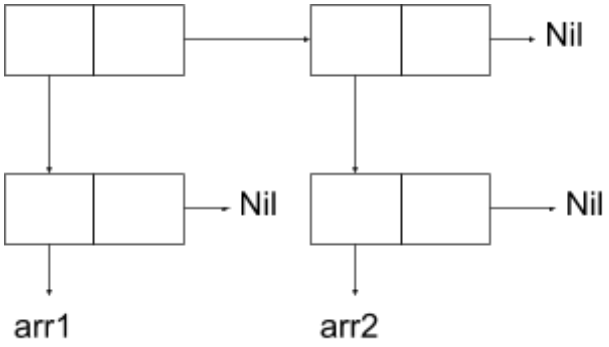
a) Функция `(f ar1 ar2 ar3 ar4)` возвращающая список `((ar1 ar2)(ar3 ar4))`

```
(defun f (ar1 ar2 ar3 ar4)
      (list (list ar1 ar2)(list ar3 ar4)))
```



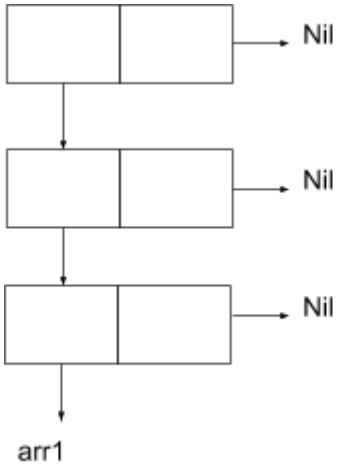

b) Функция `(f ar1 ar2)` возвращающая список `((ar1)(ar2))`

```
(defun f (ar1 ar2)
      (list (list ar1)(list ar2)))
```




c) Функция `(f ar1)` возвращающая список `(((ar1)))`

```
(defun f (ar1)
      (list (list (list ar1))))
```



Вопросы

1. **Базис языка**
    a. Атом: символ, число (самоопределимый атом).
    Символы могут обозначать числа, строки, сложные структуры, функции. T (true, истина) и NIL (false, ложь) - зарезервированные константы, встроенные функции.

    b. Точечная пара.
    <точечная пара> ::= (<атом | точечная пара>.<атом | точечная пара>)

Атомы и точечные пары объединяют под общим названием S-выражения.
Особым видом S-выражения является список
<список> ::= NIL | (<S-выражение>.<список>)

2. **Классификация функций**

- Чисто математические (фиксированное количество аргументов, результат только один)
- Формы (либо переменное число аргументов,либо они по-разному обрабатываются, либо обрабатываются не все)
- Функционал (принимают в качестве аргумента функцию)

1) Конструкторы(cons, list)
    a) cons - двухаргументная функция, создает одну списковую ячейку и расставляет указатели; (cons ’a ’b) -> (a . b); (cons ’a ’(b)) -> (a b)
    b) list- создает список. Создает и связывает столько списковых ячеек, сколько аргументов (list ’a ’b ‘c ‘d) -> (a b c d)
2) Селекторы(car, cdr)
    a) car- переходит по car указателю и возвращает голову списка
    b) cdr- переходит по cdr указателю и возвращает хвост списка
3) Предикаты (позволяют определить природу элементов)
    a) atom - возвращает T, если аргумент - атом, иначе Nil;
    b) listp - возвращает T, если аргумент - список, иначе Nil;
    c) consp - возвращает T, если аргумент представлен списковой ячейкой, иначе Nil;
    d) numberp - возвращает T, если аргумент- число, иначе Nil;
    e) symbolp - возвращает T, если аргумент- не число, иначе Nil;
4) Функции сравнения:
    a) eq - принимает два аргумента - символьные атомы. Возвращает T, если аргументы указывают на одно и то же значение (сравнивает указатели), иначе Nil.
    b) eql - сравнивает два числа (по форме их представления)
    (eql 3 3) -> T (eql 3 3.0) -> Nil

c) = - сравнивает два числа (= 3 3) -> T (= 3 3.0) -> T
d) equal - сравнивает как и eql, но также может сравнивать списки
e) equalp - сравнивает все варианты представления, но долго работает

5) Арифметические функции (- +, -, *, /)
6) Определяющие функции
   a) (defun func_name (список аргументов) (тело функции)
   b) (lambda (список аргументов) (тело функции))
7) Определяющее значение
   связывают символ со значением, предварительно вычисляя значения аргументов
   a) (set ‘a ‘b)

   В качестве значения функция SET возвращает значение второго аргумента. Если перед первым аргументом нет апострофа, то значение будет присвоено значению этого аргумента.
   b) (setq a ‘b) = (set ‘a ‘b); q - обозначает quote
   c) (setf place ‘value) - в качестве первого аргумента, в отличии от предыдущих функция, может принимать не только символьный атом

   (setq x '(1 2)) x = (1 2)
   (setf (car x) 5)) x = (5 2)
8) Функционалы (принимают функцию как аргумент)
   a) (apply #’(lambda-выражение | имя функции) арг1 арг2 арг3 ...)
   b) (fncall #’(lambda-выражение | имя функции) (арг1 арг2 арг3 ... ))

## 3. Представление списка в памяти

Пустой список представлен одним атомом - Nil. Не пустой список представлен одной или несколькими списковыми ячейками. Списковая ячейка состоит из двух полей - двух указателей: car и cdr. Car - указатель на голову, cdr - на хвост.

## 4. Car и cdr

Car - предназначена для получения первого элемента точечной пары или же головы списка. Cdr - предназначена для получения второго элемента точечной пары или же хвоста списка. Данные функции в качестве аргумента принимают точечную пару или список. Их использование возможно лишь в списочном контексте, использование для атома приведет к ошибке. Головой и хвостом пустого списка для удобства считается Nil.