Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №2 по курсу "Функциональное и логическое программирование"

Тема Определение функций пользователя

Студент Гурова Н.А.

Группа ИУ7-64Б

Оценка (баллы) _____

Преподаватель Толпинская Н.Б., Строганов Ю.В.

Москва — 2023 г.

1 Теоретические вопросы

1.1 Базис языка Lisp. Ядро языка.

Базис языка — минимальный набор инструментов и структур данных, который позволяет решать любые задачи.

```
1  Базис Lisp = атомы + структуры +
2                базовые функции + базовые функционалы
```

Функция — правило, по которому каждому значению одного или нескольких аргументов ставится в соответствие конкретное значение результата.

Примеры функций:

```
1  CAR          ; возвращает головную часть списка
2  CDR          ; возвращает хвостовую часть списка
3  CONS         ; включить новый элемент в начало списка
4  ATOM         ; проверить, является ли аргумент атомом
5  EQ           ; проверить тождественность двух символов
```

Функционал (функция высшего порядка) — функция, аргументом или результатом которой является другая функция.

Примеры функционалов:

```
1  APPLY        ; применить функцию к списку аргументов
2  FUNCALL      ; вызвать функцию с аргументами
```

Ядро — основные действия, которые наиболее часто используются. Такие функции системы обычно реализовываны в виде машинных подпрограмм.

1.2 Классификация функций

1. **Чистые математические функции** — имеют фиксированное количество аргументов, сначала вычисляются все аргументы, а потом к ним применяется функция.

2. **Рекурсивные функции** — основной способ повторения повторных вычислений.

3. **Специальные функции (формы)** — могут принимать произвольное количество элементов, элементы могут обрабатываться по разному.

4. **Псевдофункции** — создают "эффект например, вывод на экран.

5. **Функции с вариантами значений**, из которых выбирается одно.

6. **Функции высших порядков (функционалы)** — функции, аргументом или результатом которых является другая функция.

1.3 Классификация базисных функций и функций ядра

1. Селекторы

```
1    CAR          ; возвращает первый элемент списка
2    CDR          ; возвращает хвостовую часть списка
```

2. Конструкторы

```
1    CONS         ; включить новый элемент в начало списка
2    LIST         ; составить список из своих аргументов
```

3. Предикаты — логические функции, позволяющие определить структуру элемента.

```
1    ATOM         ; проверить, является ли аргумент атомом
2    NULL         ; проверить, является ли аргумент пустым списком
3    LISTP        ; проверить, является ли аргумент списком
4    CONSP        ; проверить, является ли аргумент структурой,
5                   представленной в виде списковой ячейки
```

4. Функции сравнения (перечислены по мере роста "тщательности"проверки)

```
1    EQ           ; сравнивает два символьных атома (= указатели)
2    EQL          ; сравнивает атомы и числа одинакового типа
3    =            ; сравнивает только числа, могут быть разных типов
4    EQUAL        ; EQL + сравнивает списки
5    EQUALP       ; сравнивает любые S–выражения
```

1.4 Способы задания функций

Определение функций пользователя в Лиспе возможно двумя способами.

1.4.1 Базисный способ определения функции

Предполагает использование λ-выражения (λ-нотации). Так создаются функции без имени.

Способ задания функции: **λ-выражение: (lambda λ-список форма)**, где λ-список — формальные параметры функции, форма — тело функции.

Вызов такой функции осуществляется следующим образом: **(λ-выражение формы)**, где формы — это фактические параметры.

Вычисление функций без имени может быть также выполнено с помощью функционала **apply**: **(apply λ-выражение формы)**. Функционал apply является обычной функцией с двумя вычисляемыми аргументами, обращение к ней имеет вид

```
1  apply F L          ;   F — функциональный аргумент
2                     ;   L — список фактических параметров
```

Значение функционала — результат применения F к этим фактическим параметрам.

Вычисление функций без имени может быть также выполнено с помощью функционала **funcall**: **(funcall λ-выражение формы)**. Функционал funcall — особая функция с вычисляемыми аргументами, обращение к ней

```
1  funcall  F e1 .. en          ; n >= 0
```

Действие аналогично apply, отличие в том, что аргументы передаются не в виде списка, а по отдельности.

1.4.2 Использование макро-определения defun

Задание функции: **(defun имя-функции λ-выражение)** или в облегченной форме **(defun имя-функции (x_1, x_2, .. x_k) форма)**.

Вызов именованной функции: **(имя-функции последовательность-форм)**.

Для вызова можно также воспользоваться функционалами funcall и apply.

```
1  (foo 1 2 3)
2  (funcall #'foo 1 2 3)
3  (apply #'foo (1 2 3))
```

1.5 Назначения и отличия в работе CONS и LIST

CONS принимает два указателя на любые S-выражения и возвращает новую списковую ячейку, содержащую два значения. Если второе значение не NIL и не другая списковая ячейка, то получается точечная пара. По сути функция включает значение первого аргумента в начало списка, являющегося значением второго аргумента.

LIST составляет список из значений своих аргументов. Она создаст столько списковых ячеек, сколько ей было передано. Относится к особым функциям, так как у нее может быть произвольное число параметров.

Отличия:

- CONS является базисной, LIST — нет;
- CONS имеет фиксированное количество аргментов, LIST — произволное;
- CONS создает точечную пару или список, LIST — список;
- результат функции LIST симметричен относительно своих аргументов, CONS — нет;
- CONS эффективнее LIST.

2 Практические задания

2.1 Составить диаграмму вычисления следующих выражений

1. (equal 3 (abs -3))

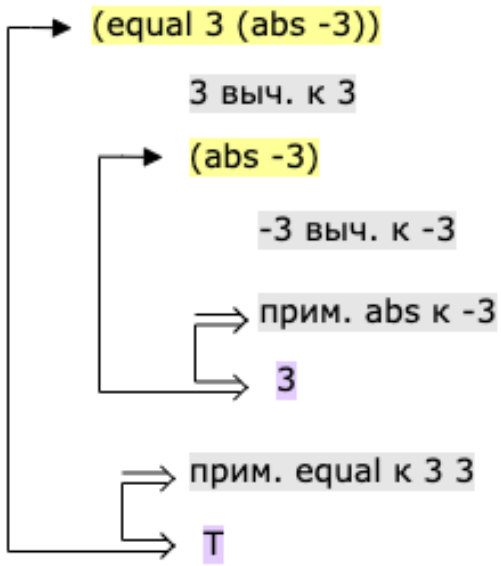

Рисунок 2.1

2. (equal (+ 1 2) 3)

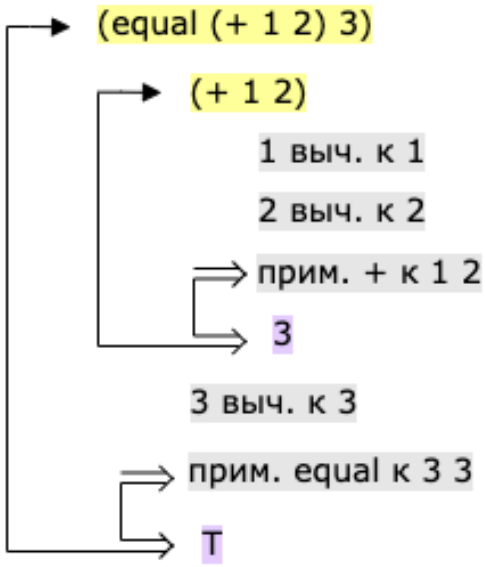

Рисунок 2.2

3. (equal (*4 7) 21)

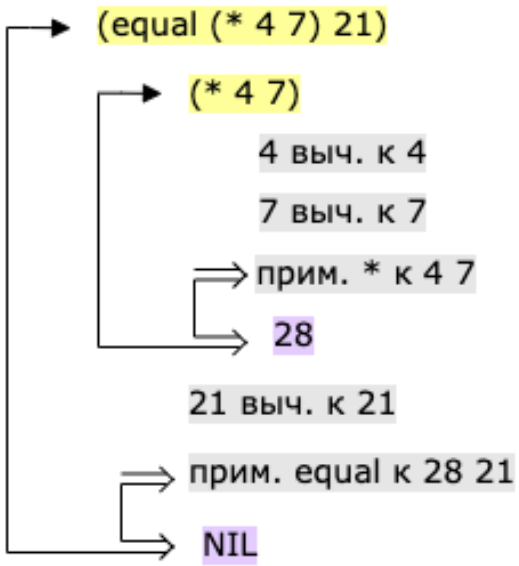

Рисунок 2.3

4. (equal (* 2 3) (+ 7 2))

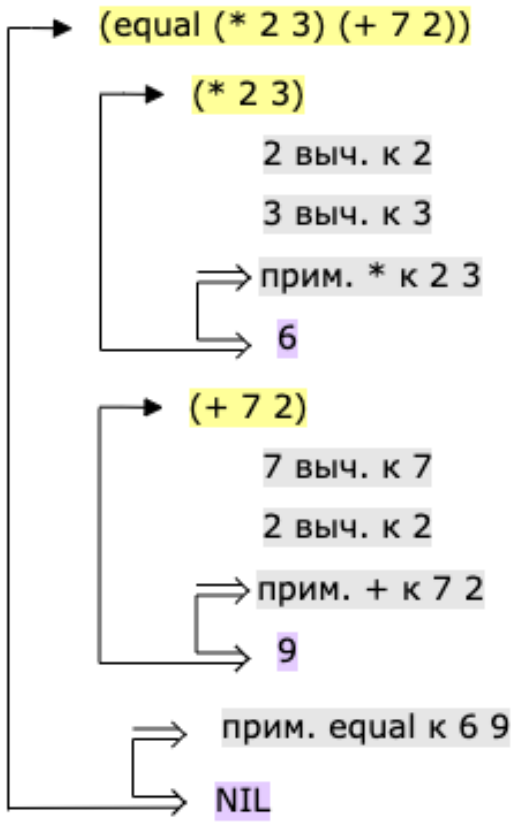

Рисунок 2.4

5. (equal (- 7 3) (* 3 2))

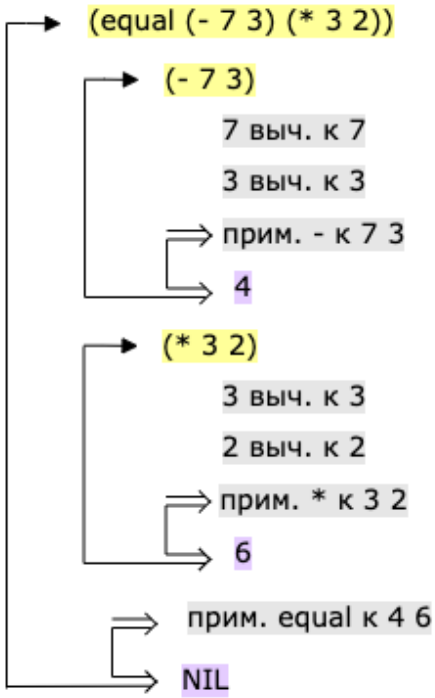

Рисунок 2.5

6. (equal (abs (- 2 4)) 3)

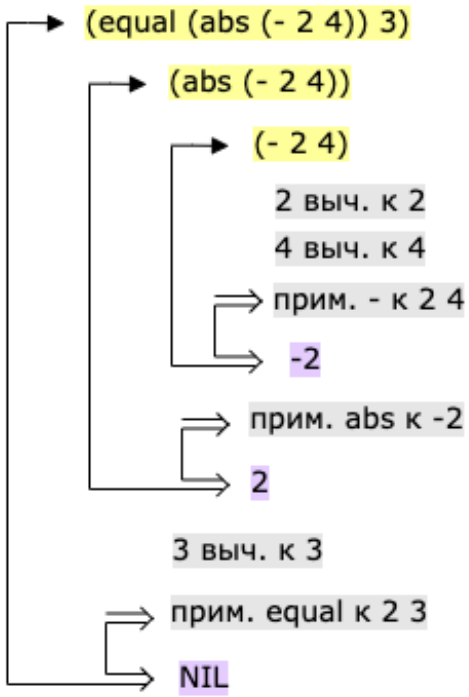

Рисунок 2.6

2.2 Написать функцию, вычисляющую гипотенузу прямоугольного треугольника по заданным катетам и составить диаграмму ее вычисления

```
1    (
2    defun
3        hypotenuse
4        (a, b)
5        (sqrt (+ (* a a) (* b b) ) )
6    )
7
8    (hypotenuse 3 4)          ; 5
```

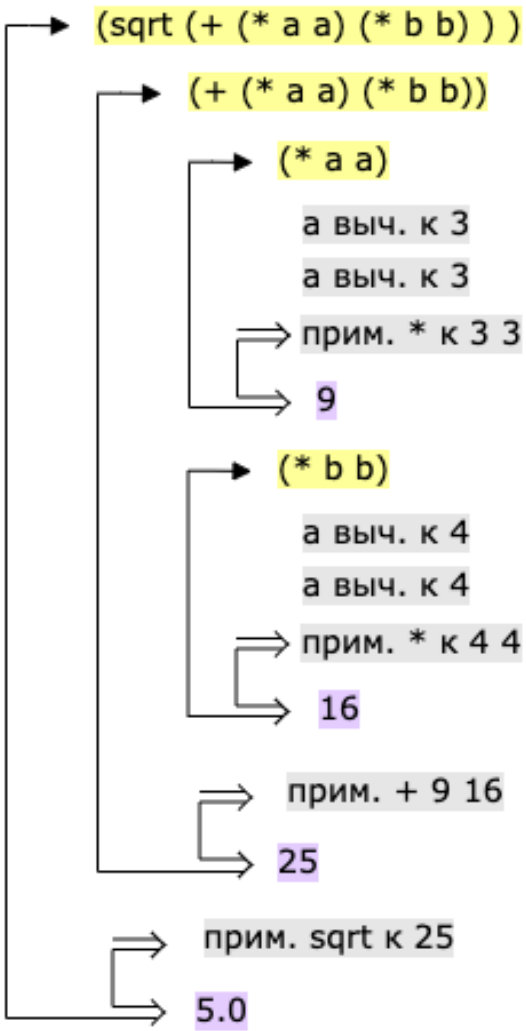

Рисунок 2.7

2.3 Каковы результаты вычисления следующих выражений?

```
(list 'a c)    =>    ERROR: Variable C is unbound.

Можно предложить два решения:
1. (list 'a 'c)                ; сделать с символом
2. (let ((c 3)) (list 'a c))    ; задать с некоторое значение
```

```
(cons 'a (b c))    =>    ERROR: Variable C is unbound.
(cons 'a (b 'c))   =>    ERROR: Function 'B' undefined.

Можно предложить два решения:
1. (cons 'a '(b c))                ; сделать (b c) списком из двух
                                     символов
2. (defun b (c) (+ c 3))           ; задать функцию b
   (let ((c 5)) (cons 'a (b c)))   ; задать с некоторое значение
```

```
(cons 'a '(b c))      =>    (A B C)
```

```
(caddr (1 2 3 4 5))   =>   ERROR: Bad function designator `1'

Решение:
1. (caddr '(1 2 3 4 5))   ; заблокировать вычисление (1 2 3 4 5)
```

```
(cons 'a 'b 'c)    =>   ERROR: too many arguments

Можно предложить два решения:
1. (cons 'a '(b c))      ; передать вторым аргументом список
2. (list 'a 'b 'c)       ; использовать функцию list
```

```
(list 'a (b c))    =>   ERROR: Variable C is unbound.

Можно предложить два решения:
1. (list 'a '(b c))                ; заблокировать вычисление
                                     второго аргумента
2. (defun b (c) (cons c Nil))      ; определить функцию b
   (let ((c 5)) (list 'a (b c)))   ; определить аргумент c
```

```
(list a '(b c))    =>   ERROR: Variable A is unbound.

Можно предложить два решения:
1. (list 'a '(b c))                ; заблокировать вычисление
                                     первого аргумента
2. (let ((a 5)) (list a '(b c)))   ; определить аргумент a
```

```
(list (+ 1 '(length '(1 2 3))))    =>   ERROR: Not a number!

Решение:
1. (list (+ 1 (length '(1 2 3)))) ; не блокировать вычисление
                                    второго аргумента
```

2.4 Написать функцию longer-then от двух списков аргументов, которая возвращает T, если первый аргумент имеет большую длину

```
(
defun
    longer_then
    (list_1 list_2)
    (> (length list_1) (length list_2))
)

(LONGER_THEN '(1 2 3) '(2 3))       ; T
(LONGER_THEN '(1 2 3) '(2 3))       ; NIL
(LONGER_THEN '(2 3 1) '(1 2 3))     ; NIL
```

2.5 Каковы результаты вычисления следующих выражений?

```
(cons 3 (list 5 6))                       ; (3 5 6)
(cons 3 '(list 5 6))                      ; (3 list 5 6)
(list 3 'from 9 'lives (- 9 3))           ; (3 from 9 lives 6)
(+ (length for 2 two))(car '(21 22 23)) ; error
(cdr '(cons is short for ans))            ; (is short for ans)
```

```
(car (list one two))                    ; error
(car (list 'one 'two))                  ; one
```

2.6 Дана функция. Какие будут результаты вычисления выражений

```
(
defun
   f1
   (x)
   (list (second x)(first x))
)
```

```
(f1 (one two))                  ; error
(f1 (last one two))             ; error
(f1 free)                       ; error
(f1 one 'two)                   ; error
```

2.7 Написать функцию, которая переводит температуру в системе Фаренгейта в температуру по Цельсию

```
(
defun
   f_to_c
   (t)
   (* (/ 5 9) (- t 32.0))
)
(f_to_c 451) ; 232.777
```

```
(
defun
   c_to_f
   (t)
   (+ (* (/ 9 5) t) 32.0)
)
(c_to_f 232.777) ; 450.999
```

2.8 Что получится при вычислении каждого из выражений

```
1      (list 'cons t NIL)                          ; (cons t ())
2      (eval (list 'cons t NIL))                   ; (t)
3      (eval (eval (list 'cons t NIL)))            ; error
4      (eval NIL)                                  ; NIL
5      (apply #cons ''(t nill)))                   ; error
6      (list 'eval NIL)                            ; (eval NIL)
7      (eval (list 'eval NIL))                     ; NIL
```