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 №3 по курсу "Функциональное и логическое программирование"

Тема Работа интерпретатора Lisp

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 Работа функций COND, IF, AND/OR

1.5.1 COND

Общий вид условного выражения:
(COND $(p_1\ e_{11}$ .. $e_{1m_1})$ $(p_2\ e_{21}$ .. $e_{2m_2})$ .. $(p_n\ e_{nn}$ .. $e_{nm_n}))$, $m_i \geq 0$, $n \geq 1$
Вычисление условного выражения выполняется по следующим правилам:

1. последовательно вычисляются условия $p_1..p_n$ ветвей выражения до тех пор, пока не встретится выражение p_i, отличное от NIL;
2. последовательно вычисляются выражения-формы $e_{i1}..e_{im_i}$ соответствующей ветви, и значение последнего выражения e_{im_i} возвращается в качестве значения функции cond;
3. если все условия p_i имеют значения NIL, то значением условного выражения становится NIL.

Пример:

```
1  (
2  cond
3     ((< X 5) (print "a") X)
4     ((= X 9) (print "b") X)
5     (T       (print "c") X)
6  )
7
8  ; Возвращает всегда X, при этом выведена будет одна из трех строк
```

1.5.2 IF

Макрофункция (IF C E1 E2), вычисляет значение выражения E1, если значение выражения C отлично от NIL, в ином случае она вычисляет E2.

```
1  (if c e1 e2) == (list 'cond (list c e1) (list T e2))
2
3  (if (< 3 5) (print "a") (print "b"))
4     ==
5  (list 'cond (list (< 3 5) (print "a")) (list T (print "b")))
```

1.5.3 AND/OR

Вызов функции and, реализующей конъюнкцию, имеет вид (AND e_1 .. e_n).

При выполнении этого функционального обращения последовательно слева направо вычисляются аргументы e_i, до тех пор, пока не встретится значение, равное NIL. В этом случае выполнение функции прерывается, возвращается NIL. Если все e_i отличны от NIL, то результатом будет e_n.

Вызов функции or, реализующей конъюнкцию, имеет вид (OR e_1 .. e_n).

При выполнении этого функционального обращения последовательно слева направо вычисляются аргументы e_i, до тех пор, пока не встретится значение, отличное от NIL. В этом случае выполнение функции прерывается, возвращается e_i. Если все e_i равны NIL, то результатом будет NIL.

Таким образом, значения функций and и or не обязательно равно T и NIL, а может быть произвольным атомом или списочным выражением.

2 Практические задания

2.1 Написать функцию, которая принимает целое число и возвращает первое четное число, больше или равное аргумента

```
1  (
2  defun
3     f1
4     (x)
5     (if (= (mod x 2) 0) x (+ x 1))
6  )
7
8  (f1 4)      ; 4
9  (f1 5)      ; 6
```

2.2 Написать функцию, которая принимает число и возвращает число того же знака, но с модулем на 1 больше модуля аргумента

```
1  (
2  defun
3     f2
4     (x)
5     (if (< x 0) (+ x -1) (+ x 1))
6  )
7
8  (f2 2.5)     ; 3.5
9  (f2 -1.0)    ; -2
```

2.3 Написать функцию, которая принимает два числа и возвращает список из этих чисел, расположенный по возрастанию

```
1      (
2      defun
3          f3
4          (x y)
5          (if (< x y) (list x y) (list y x))
6      )
7
8      (f3 2 1.2)       ; (1.2 2)
9      (f3 5 6.0)       ; (5 6)
```

2.4 Написать функцию, которая принимает три числа и возвращает T только тогда, когда первое число расположено между вторым и третьим

```
1      (
2      defun
3          f4
4          (x1 x2 x3)
5          (and (< x2 x1) (< x1 x3) T)
6      )
7
8      (f4 1 2 3)       ; NIL
9      (f4 2 1 3)       ; T
```

2.5 Каков результат вычисления следующих выражений?

```
1      (and 'fee 'fie 'foe)                    ; foe
2      (or nil 'fie 'foe)                      ; fie
3      (and (equal 'abc 'abc) 'yes)            ; yes
4      (or 'fee 'fie 'foe)                     ; fee
5      (and nil 'fie 'foe)                     ; nil
6      (or (equal 'abc 'abc) 'yes)             ; T
```

2.6 Написать предикат, который принимает два числа-аргумента и возвращает T, если первое число не меньше второго

```
(
defun
    my_geq
    (x1 x2)
    (>= x1 x2)
)

(my_geq 1 2)        ; nil
(my_geq 2 1)        ; T
(my_geq 2 2)        ; T
```

2.7 Какой из следующих двух вариантов предиката ошибочен и почему?

```
; NUMBERP — проверяет, является ли значение аргумента числом
; PLUSP   — проверяет, является ли число положительным

(
defun
    pred1
    (x)
    (and (numberp x) (plusp x))
)

(
defun
    pred2
    (x)
    (and (plusp x)(numberp x))
)
```

Первый вариант предиката корректен — он сначала проверят, что аргумент является числом, после, что число положительно.

Ошибочным является второй предикат, так как ему нельзя подать на вход строку (в этом случае выводится сообщение об ошибке).

```
(pred1 1)       ; T
(pred1 -1)      ; NIL
(pred1 "a")     ; NIL

(pred2 1)       ; T
(pred2 -1)      ; NIL
(pred2 "a")     ; ERROR: Not a number!
```

2.8 Решить задачу 4, используя для ее решения конструкции: только IF, только COND, только AND/OR

Написать функцию, которая принимает три числа и возвращает T только тогда, когда первое число расположено между вторым и третьим.

```
(
defun
   use_if
   (x1 x2 x3)
   (if (< x2 x1) (< x1 x3) NIL)
)

(use_if 1 2 3)     ; NIL
(use_if 2 1 3)     ; T
(use_if 2 1 2)     ; NIL
```

```
(
defun
   use_cond
   (x1 x2 x3)
   (cond ((< x2 x1) (< x1 x3)) )
)

(use_cond 1 2 3)    ; NIL
(use_cond 2 1 3)    ; T
(use_cond 2 1 2)    ; NIL
```

```
1     (
2     defun
3         use_and
4         (x1 x2 x3)
5         (and (< x2 x1) (< x1 x3))
6     )
7
8     (use_and 1 2 3)       ; NIL
9     (use_and 2 1 3)       ; T
10    (use_and 2 1 2)       ; NIL
```

2.9 Переписать функцию how-alike, приведенную в лекции и использующую COND, используя только конструкции IF, AND/OR

Функция вернет

- the-same, если числа равны;
- both-odd, если числа не равны и оба четны;
- both-even, если числа не равны и оба нечетны;
- difference, иначе.

```
1     (
2     defun
3         how_alike
4         (x y)
5         (
6         cond
7             ( (or (= x y) (equal x y)) 'the_same )
8             ( (and (oddp x) (oddp y)) 'both_odd )
9             ( (and (evenp x) (evenp y)) 'both_even )
10            ( T 'difference)
11        )
12    )
```

IF

```
(
defun
    how_alike_if
    (x y)
    (
    if
        (or (= x y) (equal x y))            ; условие1
        'the_same                           ; если выполнено
        (
        if
            (and (oddp x) (oddp y))         ; условие2
            'both_odd                       ; если выполнено
            (
            if
                (and (evenp x) (evenp y))   ; условие 3
                'both_even                  ; если выполнено
                'difference                 ; иначе
            )
        )
    )
)
```

```
(how_alike_if 1 2)      ; DIFFERENCE
(how_alike_if 2 2)      ; THE_SAME
(how_alike_if 2 4)      ; BOTH_EVEN
(how_alike_if 3 1)      ; BOTH_ODD
```

AND/OR

```
(
defun
   how_alike_and
   (x y)
   (
   or
       (
       and
           (or (= x y) (equal x y))
           'the_same
       )
       (
       and
           (and (oddp x) (oddp y))
           'both_odd
       )
       (
       and
           (and (evenp x) (evenp y))
           'both_even
       )
       'difference
   )
)
```

```
(how_alike_and 1 2)      ; DIFFERENCE
(how_alike_and 2 2)      ; THE_SAME
(how_alike_and 2 4)      ; BOTH_EVEN
(how_alike_and 3 1)      ; BOTH_ODD
```