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»

Отчёт по лабораторной работе №3 по курсу «Функциональное и логическое программирование»

Тема Работа интерпретатора Lisp

Студент Динь Вьет Ань

Группа ИУ7И-64Б

Оценка (баллы) _____

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

Теоретические вопросы

1. Базис Lisp

Базис – это минимальный набор инструментов языка и стркутур данных, который позволяет решить любые задачи.

Базис Lisp :

– атомы и структуры (представляющиеся бинарными узлами);

– базовые (несколько) функций и функционалов: встроенные — примитивные функции (atom, eq, cons, car, cdr); специальные функции и функционалы (quote, cond, lambda, eval, apply, funcall).

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

Функционалом, или функцией высшего порядка называется функция, аргументом или результатом которой является другая функция.

2. Классификация функций

Функции в языке `Lisp`:

- Чистые математические функции (имеют фиксированное количество аргументов, сначала выяисляются все аргументы, а только потом к ним применяется функция);
- Рекурсивные функции (основной способ выполнения повторных вычислений);
- Специальные функции, или формы (могут принимать произвольное количество аргументов, или аргументы могут обрабатываться по-разному);
- Псевдофункции (создают «эффект», например, вывод на экран);
- Функции с вариантами значений, из которых выбирается одно;
- Функции высших порядков, или функционалы – функции, аргументом или результатом которых является другая функция (используются для построения синтаксически управляемых программ)

- Базисные функции — минимальный набор функций, позволяющих решить любую задачу.

Также базисные и функции ядра можно классифицировать с точки зрения действий.

1. Селекторы — переходят по соответствующему указателю списковой ячейки.
2. Конструкторы — создают структуры данных.
3. Предикаты — позволяют классифицировать или сравнивать структуры.

3. Способы создание функций

- С помощью lambda. После ключевого слова указывается лямбда-список и тело функции.

```
(lambda (x y) (+ x y))
```

Для применения используются лямбда-выражения.

```
((lambda (x y) (+ x y)) 1 2)
```

- С помощью defun. Используется для неоднократного применения функции (в том числе рекурсивного вызова).

```
(defun sum (x y) (+ x y))
(sum 1 2)
```

4. Работа функций Cond, if, and/or

- Функция `cond`

Синтаксис:

```
(cond
    (condition-1 expression-1)
    (condition-2 expression-2)
    ...
    (condition-n expression-n))
```

По порядку вычисляются и проверяются на равенство с `Nil` предикаты. Для первого предиката, который не равен `Nil`, вычисляется

находящееся с ним в списке выражение и возвращается его значение. Если все предикаты вернут `Nil`, то и `cond` вернет `Nil`.

Примеры:

```
(cond (Nil 1) (2 3)) -> 3
(cond (Nil 1) (Nil 2)) -> %\texttt{Nil}%
```

- Функция `if`

  Синтаксис: (if condition t-expression f-expression)

  Если вычисленный предикат не `Nil`, то выполняется `t-expression`, иначе - `f-expression`.

  Примеры:

```
(if Nil 2 3) -> 3
(if 0 2 3) -> 2
```

- Функция `and`

  Синтаксис: (and expression-1 expression-2 ... expression-n)

  Функция возвращает первое `expression`, результат вычисления которого = `Nil`. Если все не `Nil`, то возвращается результат вычисления последнего выражения.

  Примеры:

```
(and 1 Nil 2) -> %\texttt{Nil}%
(and 1 2 3) -> 3
```

- Функция `or`

  Синтаксис: (or expression-1 expression-2 ... expression-n)

  Функция возвращает первое `expression`, результат вычисления которого не `Nil`. Если все `Nil`, то возвращается `Nil`.

  Примеры:

```
(or Nil Nil 2) -> 2
(or 1 2 3) -> 1
```

Практические задания

1. Задание 1

Написать функцию, которая принимает целое число и возвращает первое четное число, не меньшее аргумента.

```
(defun f1 (x) (if (oddp x) (+ x 1) x))
(f1 0) => 0
(f1 1) => 2
(f1 -3) => -2
```

2. Задание 2

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

```
(defun f2 (x) (+ x (if (< x 0) -1 1)))
(f2 -5) => -6
(f2 0.0) => 1.0
(f2 2/3) => 5/3
```

3. Задание 3

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

```
(defun f3 (x1 x2) (if (> x1 x2) (list x2 x1) (list x1 x2)))
(f3 -1 2) => (-1 2)
(f3 3 1) => (1 3)
(f3 2 2.0) => (2 2.0)
```

4. Задание 4

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

```
(defun f4 (x1 x2 x3) (and (< x2 x1) (< x1 x3)))
(f4 2 1 3) => T
(f4 1 2 3) => NIL
(f4 3 1 2) => NIL
```

5. Задание 5

Каков результат вычисления следующих выражений?

```
(and 'fee 'fie 'foe) => FOE
(or 'fee 'fie 'foe) => FEE
(or nil 'fie 'foe) => FIE
(and nil 'fie 'foe) => NIL
(and (equal 'abc 'abc) 'yes) => YES
(or (equal 'abc 'abc) 'yes) => T
```

6. Задание 6

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

```
(defun gep (x1 x2) (>= x1 x2))
(gep 2 2.0) => T
(gep -1 2/3) => NIL
```

7. Задание 7

Какой из следующих двух вариантов предиката ошибочен и почему?

```
 (defun pred1 (x) (and (numberp x) (plusp x)))
 (defun pred2 (x) (and (plusp x) (numberp x)))
```

Предикат numberp вырабатывает T, если значение его аргумента – числовой атом, и NIL в противном случае. plusp проверяет, является ли одиночное вещественное число большим чем ноль.

Таким образом, идея приведенных вариантов предиката заключается в том, чтобы проверить, является ли переданный ему аргумент числом, большим нуля.

При вычислении функционального обращения (and e1 e2 . . . en) последовательно слева направо вычисляются аргументы функции ei – до тех пор, пока не встретится значение, равное NIL. В этом случае вычисление прерывается и значение функции равно NIL. Если же были вычислены все значения ei и оказалось, что все они отличны от NIL, то результирующим значением функции and будет значение последнего выражения en.

Таким образом, корректным является первый вариант предиката. Первым будет вычислено значение аргумента e1 = (numberp x), которое проверит, является ли переданный аргумент числовым атомом. Если это не так, то e1 = (numberp x) вернет Nil, на чем вычисление функции and прервется,

и результатом всего предиката pred1 будет NIL. Если же переданный аргумент является числовым атомом, то следующим будет вычислено значение аргумента e2 = (plusp x). Это выражение проверит, является ли переданный числовой атом (здесь мы уже уверены, что переданный аргумент – числовой атом) больше нуля, и станет результатом всего предиката pred1.

Второй вариант же является ошибочным. В нем первым будет вычислено значение аргумента e1 = (plusp x), но plusp принимает только числовой атом, и если x не является числовым атомом, то вычисление всего предиката pred2 завершится с ошибкой "неверный тип".

8. Задание 8

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

(and x y) можно представить как (cond (x y)); (or x y) можно представить как (cond (x) (y)).

Все приведенные ниже функции на тестах из предыдущего номера выдают те же результаты.

```
    ; cond
    (defun f4_1 (x1  x2 x3) (cond ((< x2 x1) (< x1 x3))))
    ; if
    (defun f4_2 (x1  x2 x3) (if (< x2 x1) (< x1 x3)))
    ; or
    (defun f4_3 (x1 x2 x3) (not (or (>= x2 x1) (>= x1 x3))))
```

9. Задание 9

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

Исходная функция:

```
(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 'diff)
    )
)
(how-alike 1 1) => THE_SAME
(how-alike 2.5 2.5) => THE_SAME
(how-alike -2/3 -4/6) => THE_SAME
(how-alike 3 5) => BOTH_ODD
(how-alike -4 6) => BOTH_EVEN
(how-alike 1 2) => DIFF
```

(= работает только с числами, причем они могут быть различных типов; oddp и evenp работают только с целыми числами)

Переписанная функция:

```
; if, and, or
(defun how-alike (x y)
    (if (or (= x y) (equal x y)) 'the_same
        (if (and (oddp x) (oddp y)) 'both_odd
            (if (and (evenp x) (evenp y)) 'both_even
                'diff
            )
        )
    )
)
; and, or
(defun how-alike (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
        )
        'diff
    ))

; if
(defun how-alike (x y)
    (if
        (if (= x y) T (equal x y))
        'the_same
        (if
            (if (oddp x) (oddp y))
            'both_odd
            (if
                (if (evenp x) (evenp y))
                'both_even
                'diff
            )
        )
    )
)
```