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 дисциплине «Функциональное и логическое программирование»

Тема Работа интерпретатора Lisp

Студент Завойских Е.В.

Группа ИУ7-63Б

Оценка (баллы) _____

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

Москва — 2023 г.

1 Теоретические вопросы

1.1 Базис Lisp

Базис — это минимальный набор конструкций языка, на основе которого могут быть построены остальные.

Базис Lisp:

- атомы;
- структуры;
- базовые функции и базовые функционалы.

1.2 Классификация функций

- базисные функции;
- функции ядра;
- пользовательские функции.

Классификация функций по аргументам и поведению:

- чистые функции (фиксированное количество аргументов, для определенного набора аргументов один фиксированный результат);
- формы (переменное количество аргументов или аргументы обрабатываются по-разному);
- функционалы (принимают функцию в качестве аргумента или возвращают функцию).

Классификация функций по назначению:

- селекторы (car, cdr);
- конструкторы (cons, list);
- предикаты (atom, numberp);
- функции сравнения (eq, eql, equal).

1.3 Способы создания функций

- с использованием λ-нотации (функции без имени)

  λ-выражение: (lambda λ-список тело_функции), где λ-список — формальные параметры функции.

  Вызов такой функции осуществляется следующим способом: (λ-выражение фактические параметры).

  Вычисление функций без имени может быть выполнено с использованием функционала apply: (apply λ-выражение список_фактических_параметров); или с использованием функционала funcall: (funcall λ-выражение фактические_параметры).

- с использованием макро-определения defun:

  (defun имя_функции λ-выражение),

  или в облегченной форме:

  (defun имя_функции $(x_1, x_2, ..., x_k)$ тело_функции), где $(x_1, x_2, ..., x_k)$ — список аргументов.

  В качестве имени функции выступает символьный атом. Вызов именованной функции осуществляется следующим образом: (имя_функции фактические параметры).

1.4 Работа функций Cond, if, and/or

1.4.1 Функция cond

Синтаксис:

```
(cond
    (test1 body1)
    (test2 body2)
    ...
    (testN bodyN)
    [(T else-body)])
```

По порядку вычисляются и проверяются на равенство с Nil выражения test_i. Для первого из них, которое не равно Nil, вычисляется находящееся

с ним в списке выражение и возвращается его значение. Если все выражения test_i вернут Nil, то и cond вернет Nil. Ветка «else» организуется явным указанием в качестве test — T.

1.4.2 Функция if

Синтаксис:

```
(if test t-body f-body)
```

Если test не Nil, то выполняется t-body, иначе — f-body.

1.4.3 Функция and

Синтаксис:

```
(and arg1 arg2 ... argN)
```

Функция возвращает Nil при встрече первого (при вычислении слева направо) аргумента со значением Nil. Если все не Nil, то возвращается результат вычисления последнего аргумента.

1.4.4 Функция or

Синтаксис:

```
(or arg1 arg2 ... argN)
```

Функция возвращает первый arg_i, результат вычисления которого не Nil. Если все Nil, то возвращается Nil.

2 Практические задания

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

```
1 (defun f1 (num) (if (evenp num) num (+ num 1)))
```

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

```
1 (defun f2 (num) (if (< num 0) (- num 1) (+ num 1)))
```

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

```
1 (defun f3 (num1 num2) (if (< num1 num2) (list num1 num2) (list num2
    num1)))
```

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

```
1 (defun f4 (num1 num2 num3) (or
2                              (and (< num1 num2) (< num3 num1))
3                              (and (< num1 num3) (< num2 num1)) ) )
```

2.5 Каков результат вычисления следующих выражений?

1. (and 'fee 'fie 'foe) -> foe
2. (or 'fee 'fie 'foe) -> fee
3. (or nil 'fie 'foe) -> fie
4. (and nil 'fie 'foe) -> nil
5. (and (equal 'abc 'abc) 'yes) -> yes
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 pred (num1 num2) (>= num1 num2))
```

2.7 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)))
```

Корректным является первый вариант предиката. Первым будет вычислено значение выражения (numberp x), которое проверит, является ли переданный аргумент числовым атомом. Если это не так, то (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 выражения (plusp x). Это выражение проверит, является ли переданный числовой атом большим нуля, и станет результатом всего предиката pred1.

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

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

```
; if
(defun f4 (num1 num2 num3) (if (< num1 num2) (< num3 num1)
                               (if (< num2 num1) (< num1 num3)) ) )

; cond
(defun f4 (num1 num2 num3) (cond ((< num1 num2) (< num3 num1))
                                 ((< num2 num1) (< num1 num3)) ) )

; and/or
(defun f4 (num1 num2 num3) (or (and (< num1 num2) (< num3 num1))
                               (and (< num1 num3) (< num2 num1)) ) )
```

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

```
;
(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) ) )

; if
(defun how-alike1 (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 'difference)
                    ) ) )

; cond
(defun how_alike2 (x y)
       (cond ((= x y) 'the_same)
             ((equal x y) 'the_same)
             ((cond ((oddp x) (oddp y)) ) 'both_odd)
             ((cond ((evenp x) (evenp y)) ) 'both_even)
             (t 'difference) ) )

; and/or
(defun how_alike3 (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) )
```