**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

*к лабораторной работе №5*

*По курсу: «Функциональное и логическое программирование»*

**Тема: «**Использование управляющих структур, работа со списками».

Студент: Якуба Д.В.
Группа: ИУ7-63Б
Преподаватели: Толпинская Н. Б.,
Строганов Ю. В.

Москва, 2021 г.

# Практическая часть

Задание 1. Написать функцию, которая принимает целое число и возвращает первое чётное число, не меньшее аргумента.

Решение:

```
(defun returnFirstBiggerEven (num)
    (cond
    ((evenp (+ num 1)) (+ num 1))
    (t num)))
```

Задание 2. Написать функцию, которая принимает число и возвращает число того же знака, но с модулем на 1 больше модуля аргумента.

Решение:

```
(defun incByAbs (num) (
    cond ((< num 0) (- num 1))
    (t (+ num 1))
))
```

Задание 3. Написать функцию, которая принимает два числа и возвращает список из этих чисел, расположенный по возрастанию.

Решение:

```
(defun getSortedPair (fNum sNum) (
    cond ((<= fNum sNum) (cons fNum (cons sNum Nil)))
    (t (cons sNum (cons fNum Nil)))
))
```

Задание 4. Написать функцию, которая принимает три числа и возвращает T только тогда, когда первое число расположено между вторым и третьим.

Решение:

```
(defun isBetween (fNum sNum tNum) (
    and (> fNum sNum) (< fNum tNum)
))
```

Задание 5. Каков результат вычисления следующих выражений?

(and `fee `fie `foe);

(or `fee fie foe);

(and (equal `abc `abc) `yes);

(or nil `fie `foe);

(and nil `fie `foe);

(or (equal `abc `abc) `yes);

Решение:

(and `fee `fie `foe) -> FOE;

(or `fee fie foe) – переменные fie и foe не заданы. Для разрешения проблемы потребуется установить запрет на вычисление fie и foe: (or `fee 'fie 'foe) -> FEE;

(and (equal `abc `abc) `yes) -> YES;

(or nil `fie `foe) -> FIE;

(and nil `fie `foe) -> NIL;

(or (equal `abc `abc) `yes) -> T;

Все полученные результаты объясняются особенностями работы функций and и or. Две эти функции вычисляют переданные им аргументы до того момента, пока не станет ясен конечный результат работы функции, при этом функцией будет возвращён последний результат вычисления аргумента, на котором результат работы стал известен.

Задание 6. Написать предикат, который принимает два числа-аргумента и возвращает T, если первое число не меньше второго.

Решение:

```
(defun isBiggerOrEq (fNum sNum) (
    >= fNum sNum
))
```

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)))

Ответ: правильным будет вариант номер 1. Это связано с особенностью работы функции and, она последовательно слева направо вычисляет переданные аргументы, пока не встретит nil или вычислимое к nil s-

выражение. Таким образом, правильнее будет сначала определить, является ли переданный аргумент вычисляемым к числовому значению, а после проверить это значение на положительность.

Задание 8. Решить задачу 4, используя для её решения конструкции IF, COND, AND/OR.

Решение:

```
;ifs
(defun isBetweenIf (fNum sNum tNum) (
    if (> fNum sNum) (if (< fNum tNum) t)
))

;conds
(defun isBetweenCond (fNum sNum tNum) (
    cond ((> fNum sNum) (cond ((< fNum tNum) t)))
))

;or
(defun isBetweenOr (fNum sNum tNum) (
    and (> fNum sNum) (< fNum tNum)
))
```

Задание 9. Переписать функцию how-alike, приведенную в лекции и использующую COND, используя конструкции IF, AND/OR.

Решение:

```
; 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
    `difference))))

; and/or
(defun how_alike (x y)
    (or
    (and (or (= x y) (equal x y)) `the_same)
    (and (oddp x) (oddp y) `both_odd)
    (and (evenp x) (evenp y) `both_even)
    `difference))
```

# Теоретическая часть

1. Классификация функций.

1) чистые математические функции (имеют фиксированное количество аргументов и в качестве возврата – единственное значение);
2) рекурсивные функции;
3) специальные функции – формы (имеют произвольное количество аргументов, либо эти аргументы обрабатываются не все одинаково);
4) псевдофункции – функции, эффект которых виден на внешних устройствах;
5) функции с вариантными значениями, из которых выбирается одно;
6) функции высших порядков – функционалы (используются для построения синтаксически-управляемых программ, в качестве одного из аргументов принимают описание функции).

Классификация базисных функций:

1) селекторы: car и cdr;
2) конструкторы: cons, list;
3) предикаты: atom, null, consp, listp;
4) сравнения: eq, eql, equal, equalp.

2. Работа функций and, or, if, cond.

Функция and.

$$(and\ arg1\ arg2\ \dots\ argN)$$

Функция последовательно слева-направо вычисляет переданные ей аргументы и возвращает первый аргумент, результат вычисления которого Nil. В случае, если вычисленных значений отсутствует Nil, возвращается результат последнего вычисленного значения.

Функция or.

$$(or\ arg1\ arg2 \dots argN)$$

Функция последовательно слева-направо вычисляет переданные ей аргументы и возвращает первый аргумент, результат вычисления которого не Nil. В случае, если все вычисленные значение Nil, возвращается Nil.

Функция if.

$$(if\ test\ T_{body}\ F_{body})$$

Функция вычисляет переданный предикат test и в случае, если он не вычисляется к Nil, выполняется $T_{body}$, иначе - $F_{body}$.

Функция cond.

$$(cond\ (test1\ value\ 1)$$

$$(test2\ value2)$$

$$...$$

$$(testN\ valueN))$$

Функция вычисляет предикаты $test_i$, где $i \in [1, n]$, и для первого предиката, который не вычисляется к Nil, вычисляется $value_i$ и возвращается его значение. В случае, если все вычисленные значение Nil, возвращается Nil.

3. Способы определения функции.

Первый способ:

```
(defun *имя функции* (*список параметров*) (
      *тело функции*
      )
)
```

Пример:

```
(defun findCat (gip cat)
    (sqrt ( - (* gip gip) (* cat cat))))

(findCat 5 4) -> 3.0;
```

Второй способ:

```
(lambda (*список аргументов*)) (*тело функции*))
```

Пример:

```
((lambda (a) (* a 3)) 4) -> 12
```

lamda-функции называются «безымянными». Суть такой функции состоит в том, что задается алгоритм вычисления, но не задается имени функции. Подобную функцию можно применить к списку аргументов и сразу получить результат.