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»

Лабораторная работа №6

Студент: Луговой Д.М.
Группа: ИУ7-61Б
Преподаватель: Толпинская Н.Б.

Москва, 2020 г.

Цель работы: приобрести навыки работы в Common Lisp.

Задачи работы: изучить работу интерпретатора Lisp, алгоритм работы функции eval, структуру и порядок обработки программы в Lisp.

1. **Способы определения функций**

   Новые функции можно определить с помощью оператора defun. Он принимает три или более аргументов: имя, список параметров и ноль или более выражений, которые составляют тело функции.

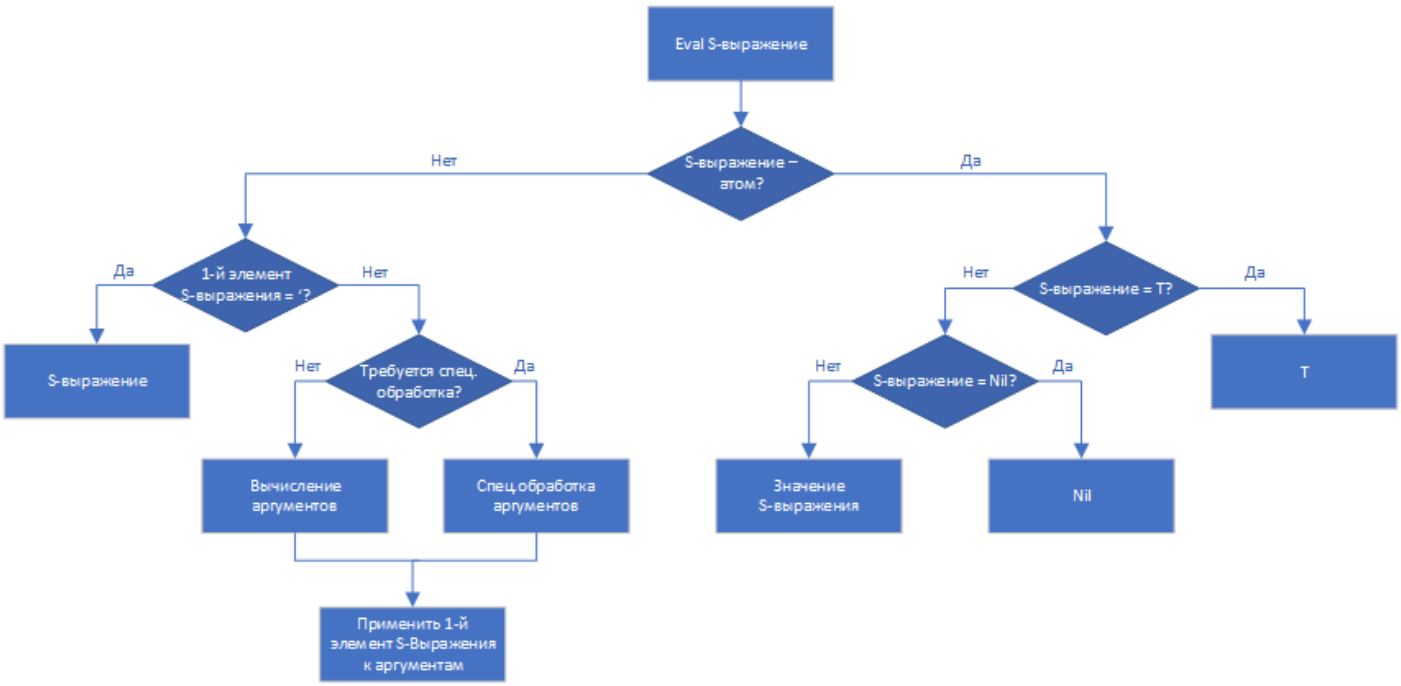
   (defun func_name (arg1 arg2 ... argN) func_body)

   Но функция не обязательно должна иметь имя, для того, чтобы определить функцию, не имеющую имени, необходимо воспользоваться лямбда-выражением. Лямбда-выражение – это список, содержащий символ lambda и следующие за ним список аргументов и тело, состоящее из нуля или более выражений.

   (lambda (arg1 arg2 .. argN) func_body)

2. **Вызовы функций, блокировка их выполнения**

   Для простейшего вызова функции используется список, первый элемент которого трактуется как имя функции, а остальные - как ее аргументы. Любое введенное S-выражение передается функции EVAL, которая вычисляет его и возвращает результат его вычисления. Схема работы функции EVAL:



Для явного вызова функции в Lisp используются функционалы APPLY и FUNCALL. APPLY принимает первым аргументом функцию, а вторым - список аргументов, которые будут переданы в функцию. FUNCALL принимает переменное число аргументов, первый из которых функция, а остальные - аргументы, которые будут переданы в функцию. Оба функционала возвращают результаты вычисления переданной им функции при переденной ей аргументах.

Для того, чтобы заблокировать вычисление S-выражения используется функция QUOTE, или ее обозначение '. Она принимает S-выражение и возвращает его же, таким образом блокируя его вычисление.

3. **Локальное и глобальное определение значения атома**

   Локальные значение атома можно определить с помощью функций let и let*. Областью видимости является тело функции, в которой определена переменная.

   Синтаксис:

   (let ((var1 value1) (var2 value2) ... (varN valueN)) body)

   Сначала вычисляются значения value1, value2, ... , valueN, а затем происходит их связывание с var1, var2, ... , varN.

   (let* ((var1 value1) (var2 value2) ... (varN valueN)) body)

   Отличие от let состоит в том, что связывание каждого значения value с символом var происходит сразу после вычисления значения.

   Глобальные значение атома устанавливается с помощью функции setf. Областью видимости является весь код, следующий после определения.

   Синтаксис:

   (setf var value)

Задание №1

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

Листинг 1: Переписанная функция how-alike

```
(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
        'difference
      )
    )
  )
)
```

Пример:

```
> (how_alike 1 1)
THE_SAME
> (how_alike 1 6)
DIFFERENCE
> (how_alike 8 4)
BOTH_EVEN
> (how_alike 5 7)
BOTH_ODD
```

Задание №2

Даны 2 списка: с названиями стран и с названиями столиц.

Написать функции для создания:

- списка из двухэлементых списков, содержащих страну и столицу;
- списка из точечных пар, содержащих страну и столицу.

Листинг 2: Функция создания списка из двухэлементных списков

```
(defun join_in_lists (countries capitals) (mapcar #'(
  lambda
    (country capital)
    (cons country
      (cons capital nil)
    )
  )
  countries capitals
))
```

Пример:

```
> (join_in_lists '(Russia England USA Ukraine) '(Moscow London Washington
    Kiev))
((RUSSIA MOSCOW) (ENGLAND LONDON) (USA WASHINGTON) (UKRAINE KIEV))
```

Листинг 3: Функция создания списка из точечных пар

```
(defun join_in_pairs (countries capitals)(mapcar #'cons countries capitals))
```

Пример:

```
> (join_in_pairs '(Russia England USA Ukraine) '(Moscow London Washington
    Kiev))
((RUSSIA . MOSCOW) (ENGLAND . LONDON) (USA . WASHINGTON) (UKRAINE . KIEV))
```

Написать функции для поиска страны по столице и столицы по стране.

Листинг 4: Функция поиска в списке двухэлементных списков

```
(defun find_in_lists_list (name lst) (cond
  ((OR
    (car
      (rassoc (cons name nil) lst :test 'equal)
    )
    (cadr
      (assoc name lst)
    )
  ))
))
```

Пример:

```
> (setf lists (join_in_lists '(Russia England USA Ukraine) '(Moscow London
    Washington Kiev))
((RUSSIA MOSCOW) (ENGLAND LONDON) (USA WASHINGTON) (UKRAINE KIEV))
> (find_in_lists_list 'London lists)
ENGLAND
> (find_in_lists_list 'USA lists)
WASHINGTON
> (find_in_lists_list 'Paris lists)
NIL
```

Листинг 5: Функция поиска в списке точечных пар

```
(defun find_in_pairs_list (name lst) (cond
  ((OR
    (car
      (rassoc name lst)
    )
    (cdr
      (assoc name lst)
    )
  ))
))
```

Пример:

```
> (setf pairs (join_in_pairs '(Russia England USA Ukraine) '(Moscow London
    Washington Kiev)))
((RUSSIA . MOSCOW) (ENGLAND . LONDON) (USA . WASHINGTON) (UKRAINE . KIEV))
> (find_in_pairs_list 'Kiev pairs)
UKRAINE
> (find_in_pairs_list 'Russia pairs)
MOSCOW
> (find_in_pairs_list 'India pairs)
NIL
```