Московский государственный технический университет
имени Н. Э. Баумана

Факультет: Информатика и системы управления
Кафедра: Программное обеспечение ЭВМ и информационные технологии
Дисциплина: Функциональное и логическое программирование

Лабораторная работа №7

Выполнили: Никичкин А. С., Фокеев А. И.
Группа: ИУ7–61

Москва, 2015 г.

1 Итеративный вариант функции `memberp`, которая возвращает `T`, или `NIL` в зависимости от того, принадлежит ли первый аргумент второму, как элемент

```
1 (defun memberp (x lst)
2   (dolist (item lst)
3     (when (equal x item)
4       (return t))))
```

2 Итеративный вариант функции `assoc`

```
1 (defun it-assoc (key table)
2   (dolist (entry table)
3     (when (equal key (first entry))
4       (return entry))))
```

3 Итеративный вариант функции `length`

3.1 с использованием `dolist`

```
1 (defun it-length-dolist (lst &aux (count 0))
2   (dolist (item lst count)
3     (incf count)))
```

3.2 с использованием `do`

```
1 (defun it-length-do (lst)
2   (do ((sub-list lst (rest sub-list))
3        (count 0 (1+ count)))
4       ((null sub-list) count)))
```

4 Итеративный вариант функции `nth`

```
1 (defun it-nth (n lst)
2   (declare (fixnum n))
3   (if (>= n 0)
4       (do ((sub-list lst (rest sub-list))
5            (index 0 (1+ index)))
6           ((= index n) (first sub-list)))))
```

5 Итеративный вариант функции `reverse`

```
1 (defun it-reverse (lst &aux (result nil))
2   (dolist (item lst result)
3     (push item result))))
```

6 Итеративные варианты функций, вычисляющие объединение, разность и симметрическую разность двух множеств

```
(defun it-union (x y &aux (result (copy-list x)))
  (dolist (item y result)
    (if (not (member item result))
        (push item result))))

(defun it-complement (x y &aux (result nil))
  (dolist (item x result)
    (when (not (member item y))
      (push item result))))

(defun it-symmetric-difference (x y)
  (it-union (it-complement x y) (it-complement y x)))
```

7 Функция возвращающая наибольший элемент из списка чисел

7.1 с помощью `dolist`

```
(defun dolist-max (lst &aux (result (first lst)))
  (dolist (item lst result)
    (when (> item result)
      (setf result item))))
```

7.2 с помощью `do*`

```
(defun do-max (lst)
  (do* ((result (first lst))
        (sub-list (rest lst) (rest sub-list))
        (item (first sub-list) (first sub-list)))
       ((null sub-list) result)
    (when (> item result)
      (setf result item))))
```

8 Функция возвращающая первый нечисловой элемент из списка

8.1 с помощью `dolist`

```
(defun dolist-first-not-number (lst)
  (dolist (item lst)
    (when (not (numberp item))
      (return item))))
```

8.2 с помощью `do*`

```
(defun do-first-not-number (lst)
  (do* ((sub-list lst (rest sub-list))
        (item (first sub-list) (first sub-list)))
       ((or (null sub-list)
            (not (numberp item)))
        item)))
```

8.3 с помощью рекурсии

```
(defun recursive-first-not-number (lst &aux (item (first lst)))
  (cond ((null lst) nil)
        ((not (numberp item)) item)
        (t (recursive-first-not-number (rest lst)))))
```

9 Функция сортирующая список из чисел по возрастанию

9.1 итеративный способ

```
(defun bubble-sort-list (lst &optional (predicate #'<))
  (do ((swapped t)) ((not swapped) lst)
    (setf swapped nil)
    (do ((lst lst (rest lst))) ((endp (rest lst)))
      (when (funcall predicate (second lst) (first lst))
        (rotatef (first lst) (second lst))
        (setf swapped t)))))
```

9.2 рекурсивный способ

```
(defun quicksort (lst &aux (pivot (first lst)))
  (if (rest lst)
      (nconc (quicksort (remove-if-not #'(lambda (x) (< x pivot)) lst))
             (remove-if-not #'(lambda (x) (= x pivot)) lst)
             (quicksort (remove-if-not #'(lambda (x) (> x pivot)) lst)))
      lst))
```