|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 № 8**

|  |  |
| --- | --- |
| **По дисциплине**: Функциональное и логическое программирование  **Тема: «**Использование управляющих структур, работа со списками**»**  **Студент** Унтилова А.О.  **Группа** ИУ7-66  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Толпинская Н.Б., Строганов Ю.В. |  |

Москва.

2021 г.

1. Напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда

а) все элеметны списка – числа

|  |
| --- |
| (defun multall (lst mul)  (mapcar #'(lambda (x) (\* x mul)) lst)) |

|  |
| --- |
| (defun mullall-r (k lst)  (cond ((null lst) nil)  (t (cons (\* k (car lst)) (mullall-r k (cdr lst))))  )  ) |

б) элементы списка - любые объекты

|  |
| --- |
| (defun mul\_ex (lst num)  (mapcar #'(lambda (x) (cond ((numberp x) (\* x num))  ((listp x) (mul\_ex k x))  (t x)))  lst)  ) |

1. Напишите функцию, select-between, которая из списка-аргумента, содержащего только числа, выбирает только те, которые расположены между двумя указанными границами-аргументами и возвращает их в виде списка (упорядоченного по возрастанию списка чисел).

|  |
| --- |
| (defun insert\_elem (elem list)  (cond  ((null list) (cons elem nil))  ((< elem (car list)) (cons elem list))  (t (cons (car list) (insert\_elem elem (cdr list))))  )  )  (defun my\_sort (list)  (reduce  #'(lambda (sorted tmp)  (insert\_elem tmp sorted)  ) list)  )  (defun select-between (lst left right)  (my\_sort (remove-if #'(lambda (x) (or (< x left) (> x right))))  lst)  ) |

1. Что будет результатом (mapcar 'вектор '(570-40-8))?

The function COMMON-LISP-USER::ВЕКТОР is undefined. Вектор воспринимается как функция, переданная в параметры mapcar, и применяется последовательно к каждому из значений по car-указателям списковых ячеек. Но такая функция не была определена

4. Напишите функцию, которая уменьшает на 10 все числа из списка аргумента этой функции.

А) **Реализация с использованием функционалов**

|  |
| --- |
| (defun subtract\_10 (lst)  (mapcar #'(lambda (x)  (cond ((numberp x) (- x 10))  ((listp x) (subtract\_10 x))  (t x)))  lst)  ) |

Б) **Реализация с использованием рекурсии**

|  |
| --- |
| (defun subtract\_10\_r (lst)  (cond  ((null lst) nil)  ((numberp (car lst)) (cons (- (car lst) 10) (subtract\_10\_r (cdr lst)) ))  ((listp (car lst)) (cons (subtract\_10\_r (car lst)) (subtract\_10\_r (cdr lst)) ))  (t (cons (car lst) (subtract\_10\_r (cdr lst)) ))  )  ) |

5. Написать функцию, которая возвращает первый аргумент списка –аргумента, который сам является непустым списком.

**А**) **Реализация с использованием функционалов**

|  |
| --- |
| (defun first\_listarg (lst)  (car ( find-if (lambda (x) (and (listp x) x)) lst  ))  ) |

**Б) Реализация с использованием рекурсии**

|  |
| --- |
| (defun first\_list (lst)  (cond ((null lst) nil)  ((and (listp (car lst)) (car lst)) (caar lst))  (t (first\_list (cdr lst)))  )  ) |

**6. Сумма элементов смешанного структурированного списка**

С использованием рекурсии:

|  |
| --- |
| (defun my\_sum\_r(lst res)  (cond  ((null lst) res)  ((numberp (car lst)) (my\_sum\_r (cdr lst)(+ res (car lst))))  ((listp (car lst)) (my\_sum\_r (cdr lst) (my\_sum\_r (car lst) res)))  (t (my\_sum\_r (cdr lst) res))  )  ) |

С использованием функционалов:

|  |
| --- |
| (defun my\_sum (lst)  (reduce #'(lambda (res x)  (cond  ((numberp x) (+ res x))  ((listp x) (+ (my\_sum x) res))  (t res)  )) lst :initial-value 0)  ) |

**Ответы на теоретические вопросы:**

**Порядок работы и варианты использования функционалов**

**Функционалы** – функции, принимающие в качестве аргумента другую функцию (имя или лямбда-выражение).

**Функционалы классифицируются на:**

1. **Применяющие**.

a. (apply #’func (arg\_lst)) – вызывает функцию func и передает список arg\_lst в качестве списка фактических параметрова функции func

b. (funcall #’func arg1…argn) – применяет функцию func к аргументам arg1…argn

2. **Отображающие.**

a. **mapcar** принимает на вход имя функции или lambda-выражение и переменное количество списков-аргументов. Если передан только один список-аргумент: функция, переданная в параметры mapcar, применяется последовательно к каждому из значений по car-указателям списковых ячеек. Из вычисленных значений формируется список с помощью list. Если передано несколько списков-аргументов: функция, переданная в аргументы mapcar, должна иметь столько же формальных параметров, сколько было передано списков-аргументов. Функция последовательно применяется к первым элементам всех списков-аргументов, затем ко вторым и т.д. Если списки-аргументы имеют разную длину, mapcar вычисляет элементы результирующего списка, пока не закончатся элементы самого короткого из списков-аргументов.

b. **mapcan** работает аналогичным образом, однако результирующий список формируется не с помощью list, a с помощью nconc.

c. **maplist** принимает имя функции или лямбда-выражение и ровно один список-аргумент. Применяет переданную функцию ко всему списку-аргументу, а затем - последовательно к каждому последующему хвосту, переходя по cdr-указателям. Формирует результирующий список из вычисленных значений с помощью cons.

d. **mapcon** работает аналогично, но формирует результирующий список с помощью nconc. e. some проверяет удовлетворение условию хотя бы одного элемента, вычисляет до первого удовлетворяющего.

Пример: (some (lambda (x) (> x 0)) '(3 -3 var -2 2)) => T

var - переменная, значение которой не было ранее определено, тем не менее ошибки не будет, тк не будет попытки вычислить ее значение

f. **every** проверяет удовлетворение условию всех элементов списка, вычисляет значения предиката до первого не удовлетворяющего элемента списка. Пример: (every (lambda (x) (> x 0)) '(3 -3 var -2 2))=> Nil

g. **find-if** последовательно вычисляет предикат от каждого из значений до первого удовлетворяющего условию элемента, возвращает значение первого удв элемента (если он найден) и Nil (если удв элементов нет )

Примеры \* (find-if (lambda (x) (> x 0)) '(-3 3 -2 2)) => 3 \* (find-if (lambda (x) (> x 10)) '(-3 3 -2 2)) => NIL

h**. remove-if** из исходного списка формирует новый список, из которого исключены \*все\* элемент удв предикату. Исходный список не изменяется при этом.

Пример: (remove-if (lambda (x) (> x 5)) ‘(1 5 6 7)) => (1 5)

i. **remove-if-not** работает аналогичным образом, но исключает элементы не удовлетворяющие предикату.

Пример: (remove-if (lambda (x) (> x 5)) ‘(1 5 6 7)) => (6 7)

j**. reduce** принимает функцию с двумя формальными параметрами и ровно один список-аргумент. Применяет функцию к первым двум элементам списка, затем - последовательно применяет функцию к получаемому на каждом этапе результату и следующему элементу списка-аргумента.

Пример: (reduce (lambda (x y) (cons y x)) ‘(1 2 3) :initial-value Nil) => (3 2 1)