|  |  |
| --- | --- |
| 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 № 7**

|  |  |
| --- | --- |
| **По дисциплине**: Функциональное и логическое программирование  **Тема: «**Использование управляющих структур, работа со списками**»**  **Студент** Козлов М.А.  **Группа** ИУ7-65  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Толпинская Н.Б., Строганов Ю.В. |  |

Москва.

2021 г.

1. Написать функцию, которая по своему списку-аргументу lst определяет, является ли он палиндромом (то есть равны ли lst и (reverse lst)).

|  |
| --- |
| (defun reverse\_m (lst)  (reduce #'(lambda (res el)  (cons el res))  lst :initial-value nil)  )  (defun polindrom(lst)  (equalp lst (reverse\_m lst))  ) |

1. Написать предикат set-equal, который возвращает t, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения.

|  |
| --- |
| (defun subsetp\_m (sset subset)  (or (null subset)  (if (member (car subset) sset)  (subsetp\_m sset (cdr subset))  )  )  )  (defun set\_equalp (lst1 lst2)  (and (subsetp lst1 lst2) (subsetp lst2 lst1))  ) |

1. Напишите необходимые функции, которые обрабатывают таблицу и точечных пар:(страна, столица), и возвращают по стране - столицу, по столице – страну

|  |
| --- |
| (defun find\_by\_func(val lst)  (find-if #'(lambda (pair)  (cond  ((equal (car pair) val) (cdr pair))  ((equal (cdr pair) val) (car pair))  )  ) lst)  ) |

|  |
| --- |
| (defun find\_by\_func\_r (val lst)  (cond ((null lst) nil)  ((equal (caar lst) val) (cdar lst))  ((equal (cdar lst) val) (caar lst))  (t (find\_by\_func\_r val (cdr lst)))  )  ) |

1. Напишите функцию swap-first-last, которая переставляет в списке-аргументе первый и последний элементы.

|  |
| --- |
| (defun last\_m(lst)  (reduce #'(lambda (a x) x) lst))  (defun rem\_last\_m(lst)  (mapcar #'(lambda (x y) x) lst (cdr lst)))  (defun swap-first-last (lst)  (nconc (cons (last\_m lst) (rem\_last\_m (cdr lst))) (cons (car lst) nil))) |

1. Напишите функцию swap-two-ellement, которая переставляет в списке-аргументе два указанных своими порядковыми номерами элемента в этом списке.

|  |
| --- |
| (defun set\_and\_get\_i (lst i x)  (cond ((null lst) nil)  ((= i 0) (let ((y (car lst))) (setf (car lst) x) y) )  (t (set\_and\_get\_i (cdr lst) (- i 1) x))  )  )  (defun swap\_two\_element (lst i j)  (cond ((null lst) nil)  ((= i 0) (setf (car lst) (set\_and\_get\_i lst j (car lst))))  ((= j 0) (setf (car lst) (set\_and\_get\_i lst i (car lst))))  (t (swap\_two\_element (cdr lst) (- i 1) (- j 1)))  )  lst  ) |

1. Напишите две функции, swap-to-left и swap-to-right, которые производят круговую перестановку в списке-аргументе влево и вправо, соответственно.

|  |
| --- |
| (defun add\_rec (el lst)  (if (null lst)  (cons el nil)  (cons (car lst) (add\_rec el (cdr lst)))  )  )  (defun swap-to-left (lst k)  (if (< k 1)  lst  (swap-to-left (add\_rec (car lst) (cdr lst)) (- k 1))  )  )  (defun swap-to-right (lst k)  (if (< k 1)  lst  (swap-to-right (cons (last\_m lst) (rem\_last\_m lst)) (- k 1))  )  ) |

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)) |

б) элементы списка - любые объекты

|  |
| --- |
| (defun multiply-all (lst mul)  (mapcar #'(lambda (x)  (cond ((numbperp x) (\* x mul))  ((listp x) (multiply-all x mul))  (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 select-between (lst left right)  (remove-if #'(lambda (x) (or (< x left) (> x right)))  lst)  ) |

**Ответы на теоретические вопросы:**

**1.Способы определения функций**

Для создания функции служит специальная встроенная функция DEFUN. Вызов этой функции требует трех аргументов.

Общий вид: (defun <имя функции> (<формальные параметры>) (<тело функции>))

Пример: (defun sum2 (a b) (+ a b))

Если вызов функции DEFUN завершился удачно, то в качестве результата возвращается атом, заданный первым параметром, а в системе появляется новая функция, требующая при вызове столько аргументов, сколько элементов

содержалось в списке формальных параметров. При вычислении этой новой функции значения аргументов будут вычисляться.

Например: (defun sum(arg1, arg2) (+ arg1 arg2)) .

Также функций можно определять через лямбда выражения. Лямбда выражение— это список, содержащий в себе слово lambda и список аргументов и следующие за ним тело функции, состоящее из 0 или более выражений.

Общий вид лямбда-выражения:

(lambda (<список формальных параметров>) (<тело функции>))

(lambda (<список формальных параметров >) (<тело функции >) <фактические параметры> )

Пример: (lambda (x y) (+ (\* x x) (\* y y)))

(lambda (x y) (+ (\* x x) (\* y y)) 2 3) => 13

Вычисление лямбда функций происходит в два этапа: сначала вычисляются фактические параметры и связываются с формальными, а затем уже происходит выполнение функции. Плюс использования лямбда функции в том, что мы можем возвращать не только целые числа и так далее, но и функции.

Функцию, определенную с помощью lambda-выражения можно вызвать с помощью funcall или apply:

(funcall <lambda-выражение> <фактические параметры>)

(apply <lambda-выражение> (<список фактических параметров>))

Например:

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

**2. Варианты и методы модификации элементов списка.**

Лисп позволяет применять функцию к каждому из элементов списка (или списков) с помощью отображающих функционалов.

**Функционал 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.

Общий вид: (mapcar #’func ‘(x1 x2 … xn)) -> (list (func x1) (func x2)… (func x3))

Пример: (mapcar (lambda (x) (\* x 2)) ‘(1 2 3)) => (2 4 6)

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.

Общий вид: (mapcar #’func lst1 lst2 … lstn)

Пример: (mapcar (lambda (x y z) (+ x y z)) ‘(1 2 3) ‘(2 3 4) ‘(3 4 5)) => (6 9 12)

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

Сравнение работы:

\* (mapcar (lambda (x y z) (list x y z)) ‘(1 2 3) ‘(3 4 5) ‘(5 6 7)) ((1 3 5) (2 4 6) (3 5 7))

\* (mapcan (lambda (x y z) (list x y z)) ‘(1 2 3) ‘(3 4 5) ‘(5 6 7)) (1 3 5 2 4 6 3 5 7)

**Функционал 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.

Общий вид: (maplist #’func lst)

Пример: (maplist (lambda (lst\_i) (apply ‘+ lst\_i)) ‘(1 2 3)) => (6 5 3)

**Функционал mapcon** работает аналогично, но формирует результирующий список с помощью nconc.

Сравнение работы:

\* (mapcon 'list '(1 2 3)) ((1 2 3) (2 3) (3))

\* (maplist 'list '(1 2 3)) (((1 2 3)) ((2 3)) ((3)))