|  |  |
| --- | --- |
| 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 №9*

*По предмету: «Функциональное и логическое программирование»*

Преподаватель: Толпинская Н.Б.,

Студент: Мирзоян С.А.,

Группа: ИУ7-65Б

Москва, 2020 г.

В языке Lisp возможна организация повторных вычислений. Ее можно организовать с помощью циклов (do) и рекурсий

Макрос do – основной итерационный оператор в Common Lisp.

(do ((variable1 value1 updated-value1)

(variable2 value2 updated-value2)

(variable3 value3 updated-value3)

...)

(test return-value)

(s-expressions)

)

Пример использования:

(do ((x 0 (+ 2 x))

(y 20 (- y 2))

((= x y) (- x y))

(format t “~% x = ~d y - ~d” x y)

Результат:

x = 0 y = 20

x = 2 y = 18

x = 4 y = 16

x = 6 y = 14

x = 8 y = 12

Рекурсия — это ссылка на определяемый объект во время его определения.

Классификация рекурсивных функций:

1. простая рекурсия - один рекурсивный вызов в теле
2. рекурсия первого порядка - рекурсивный вызов встречается несколько раз
3. взаимная рекурсия - используется несколько функций, рекурсивно вызывающих друг друга.

Типы рекурсивных функций:

* 1. хвостовая,
  2. дополняемая,
  3. множественная,
  4. взаимная рекурсия,
  5. рекурсия более высокого порядка.

При организации рекурсии можно использовать:

* + 1. Функции с именем
    2. Локально определенные с помощью лямбда выражений функции
    3. Функционалы
    4. Рекурсивную функцию внутри функционала

Важным моментом в организации рекурсии является выход из нее. Из рекурсии всегда должен быть выход. Некачественный выход из рекурсии может привести к переполнению памяти из-за "лишних" рекурсивных вызовов. Кроме того возможна потеря аргумента - кажется что функция возвращает результат и он используется, но на деле результат теряется и ответ неверен.

Способы повышения эффективности:

* + - 1. **Хвостовая рекурсия.** В целях повышения эффективности рекурсивных функций рекомендуется формировать результат не на выходе из рекурсии, а на входе в рекурсию, все действия выполняя до ухода на следующий шаг рекурсии. Это и есть хвостовая рекурсия.
      2. **Дополняемая рекурсия** — при обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова , а вне его  
         В качестве дополнительной функции в дополняемой рекурсии очень часто используется cons , которая формирует результат.
      3. **Выделяют группу функций множественной рекурсии**. На одной ветке происходит сразу несколько рекурсивных вызовов. Количество условий выхода также может зависеть от задачи.

Для превращения не хвостовой рекурсии в хвостовую и в целях формирования результата (результирующего списка) на входе в рекурсию, рекомендуется использовать дополнительные (рабочие) параметры. При этом становится необходимым создать функцию — оболочку для реализации очевидного обращения к функции:

(defun into\_one\_level (lst)

( into\_one lst () )

)

(defun into\_one (lst rst)

(cond ( (null lst) rst )

( (atom lst) (cons lst rst) )

( t (into\_one (car lst)

(into\_one (cdr lst) rst)

)

)

)

)

**Задачи**

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

(defun set-equal (x y) (and (subsetp x y) (subsetp y x)))

**3. Напишите необходимые функции, которые обрабатывают таблицу из точечных пар (страна.столица), и возвращают по стране - столицу, а по столице - страну.**

(defun check (pair val) *;возвращает противоположный элемент из точечной пары если проходит совпадение, иначе - Nil*

(cond ((equal (car pair) val) (cdr pair))

((equal (cdr pair) val) (car pair))

)

)

(defun generate-check (val) *;возвращает функцию, которая будет сравнивать свой параметр с этим самым валом*

(lambda (pair) (check pair val))

)

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

**а) все элеметны списка - числа,**

(defun multiply-all (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

)

)

**2. Напишите функцию, которая уменьшает на 10 все числа из списка-аргумента этой функции.**

(defun all\_minus\_10 (lst);

(mapcar #'(lambda (x)

(cond ((numberp x) (- x 10))

((listp x) (all\_minus\_10 x))

(t x)

)

)

lst

)

)

**3. Написать функцию, которая возвращает первый аргумент списка-аргумента, который сам является непустым списком.**

(defun get\_first\_element (lst)

(if (and (listp (car lst))

(not (null (car lst)))

)

(car lst)

(get\_first\_element (cdr lst))

)

)

**4. Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10.**

**(Вариант: между двумя заданными границами.)**

(defun select\_between\_inner (lst left right result)

(mapcar #'(lambda (x)

(cond ((listp x) (select\_between\_inner x left right result)) ((and (numberp x) (> x left) (< x right))

(nconc result (cons x nil))

)

)

)

lst

)

(cdr result)

)

(defun select\_between (lst left right);

(select\_between\_inner lst left right (cons nil nil))

)

**5. Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов. A x B - это множество всевозможных пар (a b), где a принадлежит A, b принадлежит B.**

(defun decart (X Y)

(mapcan #'

(lambda (x)

(mapcar #'

(lambda (y) (list x y))

Y

)

)

X

)

)

**6. Почему так реализовано reduce, в чем причина?**

**(reduce #'+ ()) -> 0**

**(reduce #'\* ()) -> 1**

Сначала функция проверяет список-аргумент. Если он пуст, возвращается значение функции при отсутствии аргументов.

Также reduce использует аргумент :initial-value. Этот аргумент определяет значение, к которому будет применена функция при обработке первого элемента списка-аргумента. Если список-аргумент пуст, то будет возвращено значение initial-value.