|  |  |
| --- | --- |
| 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

**Отчет**

**по лабораторной работе № 10**

**Дисциплина: «Функциональное и логическое программирование»**

Выполнила: Овчинникова А.П.

Группа: ИУ7-65Б

Преподаватель: Толпинская Н.Б.

Строганов Ю.В.

Москва, 2020

**Теоретическая часть.**

**Вопрос 1.**

Повторные вычисления в Лисп могут быть организованы с помощью рекурсии или с помощью функционалов.

Функционалы бывают:

* применяющие:
  + *(apply #’ fun args)* – применяет функцию fun к аргументам args, которых должно быть не менее одного. Последний аргумент args должен быть списком. Функция fun может быть также символом, и в этом случае используется связанная с ним глобальная функция.
  + *(funcall #’fun args)* – применяет функцию fun к аргументам args. Функция может быть задана как символ, и в этом случае используется определение глобальной функции, связанной с этим символом.
* отображающие:
  + *(mapar #’fun lst)* – ко всем элементам списка *lst* применяется функция *fun*. Из результатов применения этой функции к элементам списка формируется результирующий список. Функция *fun* должна быть одноаргументной.
  + *(mapcar #’fun lst1 … lstN)* – применяет функцию *fun* сначала ко всем первым элементам списков *lst1 … lstN*, затем ко всем последовательным элементов каждого списка *lst1 … lstN*. Прекращает работу, когда заканчиваются элементы самого короткого из списков *lst1 … lstN*. В результате получается список списков результатов каждого вызова функции *fun*. Функция *fun* должна иметь n аргументов.
  + *(maplist #’fun lst)* – вызывает функцию *fun* n раз (n – длина списка *lst*)для *lst* целиком, затем для всех последовательных cdr списка *lst*, заканчивая (n-1)-м. Возвращает список значений, полученных функцией *fun.* Функция *fun* должна быть одноаргументной.
  + *(maplist #’fun lst1 … lstN )* – вызывает функцию *fun* n раз (n – длина кратчайшего из списков *lst1 … lstN*) для каждого списка *lst1 … lstN* целиком, затем для всех последовательных cdr каждого *lst1 … lstN*, заканчивая (n-1)-м. Возвращает список значений, полученных функцией *fun.* Функция *fun* должна иметь n аргументов.

Во всех случаях функция *fun* может быть задана именем функции или лямбда-определением. Здесь лямбда-определение будет более эффективным, так как нет необходимости искать функцию по имени среди атомов.

**Вопрос 2.**

Использование функционалов mapcar и maplist описано в вопросе 1. Кроме этого, существуют также следующие функционалы:

1. *(mapcan #’fun prolist)* – эквивалент применения nconc к результату вызова mapcar с теми же аргументами.
2. *(mapcon #’fun prelist)* – эквивалент применения nconc к результату вызова maplist с теми же аргументами.
3. *(find-if #’predicate proseq)* – возвращает первый соответствующий предикату элемент proseq.
4. *(remove-if #’predicate proseq)* – возвращает последовательность, похожую на proseq, но без всех элементов, для которых справедлив предикат predicate.
5. *(reduce #fun lst)* – применяет fun к элементам lst каскадным образом. Если lst пуст, то fun вызывается без аргументов.
6. *(some #’predicate proseq)* – возвращает истину, если предикат predicate, который должен быть функцией стольких аргументов, сколько задано последовательностей proseq, возвращает истину, будучи примененным к первым элементам всех последовательностей, или ко вторым, или к n-м, где n – длина кратчайшей последовательности proseq. Проход по последовательностям останавливается тогда, когда предикат вернет истину.
7. *(every #’predicate preseq)* – возвращает истину, если предикат predicate, который должен быть функцией стольких аргументов, сколько последовательностей передано, истинен для всех первых элементов последовательностей proseq, затем для всех вторых элементов, и так до n-го элемента, где n – длина кратчайшей последовательности proseq. Проход по последовательностям завершается, когда предикат возвратит nil. В этом случае в результате всего вызова возвращается nil.

**Вопрос 3.**

Рекурсия — это ссылка на определяемый объект во время его определения. Т. к. в Lisp используются рекурсивно определенные структуры (списки), то рекурсия — это естественный принцип обработки таких структур.

**Классификация рекурсивных функций.**

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. **Рекурсия по нескольким параметрам.**
3. **Дополняемая рекурсия** — при обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова, а вне его.
4. **Функции множественной рекурсии**. На одной ветке происходит сразу несколько рекурсивных вызовов. Количество условий выхода также может зависеть от задачи.

**Вопрос 4.**

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, то есть использовать хвостовую рекурсию.

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.

Функция *(nconc lists)* возвращает список с элементами из всех *lists* по порядку. Принцип работы: устанавливает cdr последней ячейки каждого списка в начало следующего списка. Последний аргумент может быть объектом любого типа. Вызванная без аргументов, вовзрващает nil.

Функция *(mapcan function prolist)* – эквивалент применения nconc к результату вызова mapcar с теми же аргументами.

Функция *(mapcon function prolist)* – эквивалент применения nconc к результату вызова maplist с теми же аргументами.

Функция *(butlast list n)* возвращает копию списка list без последних n элементов или nil, если список list имеет менее n элементов. Возвращает ошибку, если n – отрицательное число.

Функция *(rassoc key alist)* возвращает первый элемент (списковую ячейку) alist, cdr которого совпал с key.

Функция *(assoc key alist)* возвращает первый элемент (списковую ячейку) alist, car которого совпадает с key.

**Практическая часть.**

**Задание 7.**

Пусть list-of-lists список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-lists.

Функции sum\_length и sum\_length2 принимают один параметр – список. Функция sum\_length использует два функционала: mapcar и reduce. Функция лямбда-функция, которую mapcar применяет к каждому элементу исходного списка, заменяет каждый элемент в этом списке на его длину, т. е. если это не список, то он заменяется на 1, иначе с помощью функции sum\_length вычисляется длина этого элемента, и он заменяется на его длину. reduce складывает длины элементов, которые были записаны.

Функция sum\_length2 рекурсивная. Она по очереди (рекурсивно) проверяет каждый элемент списка lst. Если это не список, то к длине cdr lst прибавляется 1, иначе к длине cdr lst прибавляется длина car lst, которая вычисляется с помощью этой же функции sum\_length2.

*(defun sum\_length (lst)*

*(reduce #'+ (mapcar #'(lambda (elem)*

*(if (listp elem) (sum\_length elem) 1)*

*)*

*lst*

*)*

*)*

*)*

*(defun sum\_length2 (lst)*

*(cond*

*( (null lst) 0 )*

*( (listp (car lst)) (+ (sum\_length2 (car lst)) (sum\_length2 (cdr lst))) )*

*( t (+ 1 (sum\_length2 (cdr lst))) )*

*)*

*)*

**Задание 8.**

Написать рекурсивную версию вычисления суммы чисел заданного списка.

Функция rec\_sum принимают один аргумент-список lst. Функция rec\_sum рекурсивная. Она проверяет каждый элемент lst. Если он является списком, то сумма его элементов, вычисляемая с помощью rec\_sum, складывается с суммой элементов cdr lst. Если он является числом, то это число складывается с суммой элементов cdr lst. Если он не является ни числом ни списком, то сумма элементов cdr lst складывается с 0.

*(defun rec\_sum (lst)*

*(cond*

*( (null lst) 0)*

*( (listp (car lst)) (+ (rec\_sum (car lst)) (rec\_sum (cdr lst))) )*

*( (numberp (car lst)) (+ (car lst) (rec\_sum (cdr lst))) )*

*( (not (numberp (car lst))) (+ (rec\_sum (cdr lst))) )*

*)*

*)*

**Задание 9.**

Написать рекурсивную версию функции nth.

Функция rec\_nth принимает два параметра: параметр-список lst и целое положительное число count. Выход из рекурсии происходит, когда count равен 0.

*(defun rec\_nth (count lst)*

*(cond*

*( (or (not (integerp count)) (<= count 0)) `(,count не является положительным целым числом) )*

*( (= count 1) (car lst) )*

*( t (rec\_nth (- count 1) (cdr lst)) )*

*)*

*)*

**Задание 10.**

Написать рекурсивную функцию, которая возвращает t, когда все элементы списка нечетные.

Функция alloddr принимает один аргумент-список lst. Выход из рекурсии происходит, если список lst пуст, в этом случае возвращается t. Если car lst является списком, то alloddr вызывается и для car lst, и для cdr lst. Если car lst не является списком, то alloddr вызывается для cdr lst.

*(defun alloddr (lst)*

*(cond*

*( (null lst) t)*

*( (listp (car lst)) (and (alloddr (car lst)) (alloddr (cdr lst))) )*

*( t (and (numberp (car lst)) (oddp (car lst)) (alloddr (cdr lst))) )*

*)*

*)*

**Задание 11.**

Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка-аргумента.

*(defun rec\_get\_last (lst)*

*(cond*

*( (null (cdr lst)) (car lst))*

*( t (rec\_get\_last (cdr lst)) )*

*)*

*)*

*(rec\_get\_last '(1 2 3 4))*

**Задание 12.**

Написать рекурсивную функцию, относящуюся к дополняемой рекурсии с одним тестом завершения, которая вычисляет сумму всех чисел от 0 до n-ого аргумента функции.

Функция sum\_n принимает один аргумент – число n – и вычисляет сумму чисел от до n. Все вычисления выполняет рекурсивная функция sum\_inner\_start, которая принимает два аргумента: числа start и n. sum\_inner\_start вычисляет сумму всех чисел от start до n с шагом 1. Ни в одной из этих двух функций не проводится проверок аргументов: являются ли они числами, или нет. Поэтому, если в качестве аргументов передаются не числа, возникает ошибка.

Функция sum\_n2 принимает три аргумента: числа n – нижняя граница, m – верхняя граница и d – шаг. Проверка аргументов (являются ли они числами) не проводится. Проводится только проверка аргумента d – он должен быть положительным. Все вычисления выполняет рекурсивная функция sum\_n2\_inner. Она принимает три аргумента – числа n, m, d. Выход из рекурсии происходит тогда, когда текущая нижняя граница + шаг больше верхней границы.

*(defun sum\_n\_inner (start n)*

*(cond*

*( (= n start) start)*

*(t (+ n (sum\_n\_inner start (- n 1))))*

*)*

*)*

*(defun sum\_n (n)*

*(sum\_n\_inner 0 n)*

*)*

*(defun sum\_n2\_inner (n m d)*

*(cond*

*( (> (+ n d) m) n)*

*(t (+ n (sum\_n2 (+ n d) m d)))*

*)*

*)*

*(defun sum\_n2 (n m d)*

*(cond*

*( (<= d 0) `(,d не является положительным числом) )*

*( t (sum\_n2\_inner n m d) )*

*)*

*)*

**Задание 13.**

Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции.

Функция get\_last принимает один аргумент – числовой список. Она использует функцию get\_last\_inner, которая тоже принимает один аргумент-список (числовой) и возвращает первое нечетное число. Для того, чтобы получить последнее нечетное число, get\_last применяет reverse к исходному списку lst.

Функция get\_last2 принимает один аргумент – числовой список. Она использует функцию get\_last\_inner2. Функция get\_last\_inner2 принимает два аргумента: числовой список и значение по умолчанию (оно возвращается, если ничего не найдено). Когда get\_last\_inner2 встречает в списке нечетное число, то оно «сохраняется» в аргументе val. Когда список становится пуст (или он мог сразу быть пуст), рекурсия останавливается и возвращается val.

*(defun get\_last (lst)*

*(get\_last\_inner (reverse lst))*

*)*

*(defun get\_last\_inner (lst)*

*(cond*

*( (null lst) Nil)*

*( (oddp (car lst)) (car lst))*

*( t (get\_last\_inner (cdr lst)) )*

*)*

*)*

*(defun get\_last\_inner2 (cur val)*

*(cond*

*( (eql cur Nil) val )*

*( (oddp (car cur)) (get\_last\_inner2 (cdr cur) (car cur)) )*

*( t (get\_last\_inner2 (cdr cur) val) )*

*)*

*)*

*(defun get\_last2 (lst)*

*(get\_last\_inner2 lst Nil)*

*)*

**Задание 14.**

Используя cons-дополняемую рекурсию с одним тестом завершения, написать функцию, которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

Функция squares принимает один аргумент – список lst. Элементы этого списка не проверяются на то, являются ли они числами. Поэтому при передаче squares не числового списка, возникнет ошибка. Выход из рекурсии происходит, когда список lst пуст.

*(defun squares (lst)*

*(cond*

*( (null lst) Nil)*

*( t (cons (\* (car lst) (car lst)) (squares (cdr lst))) )*

*)*

*)*

*(squares '(1 5))*

**Задание 15.**

Написать функцию, которая из заданного списка выбирает все нечетные числа .

Функция select\_odd принимает один аргумент-список. Элементы списка проверяются на то, являются ли они числами. Функция выбирает нечетные числа только из списка верхнего уровня. Функция использует функционал mapcan.

Функция select\_odd2 также принимает один аргумент-список. Внутри select\_odd2 используется mapcan и функция select\_odd2\_inner. select\_odd2\_inner принимает один аргумент – элемент исходного списка. Она проверяет, является ли текущий элемент списком или нечетным числом. Если это нечетное число, то элемент заносится в результирующий список. Если это список, то он обрабатывается так же, как и исходный список lst. Иначе (не список и не нечетное число) никаких действий не производится.

*(defun select\_odd (lst)*

*(mapcan #'(lambda (elem)*

*(if (and (numberp elem) (oddp elem)) (list elem))*

*)*

*lst*

*)*

*)*

*(defun select\_odd2\_inner (elem)*

*(cond*

*( (listp elem) (list (mapcan #'select\_odd2\_inner elem)) )*

*( (and (numberp elem) (oddp elem)) (list elem) )*

*)*

*)*

*(defun select\_odd2 (lst)*

*(remove Nil (mapcan #'select\_odd2\_inner lst))*

*)*

*(defun select\_odd3\_inner (lst res)*

*(mapcar #'(lambda (elem)*

*(cond*

*( (listp elem) (select\_odd3\_inner elem res))*

*( (and (numberp elem) (oddp elem)) (nconc res (cons elem Nil)))*

*)*

*)*

*lst*

*)*

*(cdr res)*

*)*