|  |  |
| --- | --- |
| 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 № 6**

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

Выполнила: Овчинникова А.П.

Группа: ИУ7-65Б

Преподаватель: Толпинская Н.Б.

Москва, 2020

**Теоретическая часть.**

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

Рекурсия – это ссылка при описании объекта на этот же объект.

Функционалы бывают:

* применяющие;
* 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.

Функция *(oddp i)* возвращает истину, если *i* нечетное число.

Функция *(evenp i)* возвращает истину, если *i* четное число.

Специальный оператор *let* позволяет ввести новые локальные переменные:

*(let ((var1 value1)*

*…*

*(varN valueN)*

*body)*

*let* вычисляет свое тело, предварительно связав каждый символ с соответствующим значением или Nil в случае отсутствия значения *value*. Сначала вычисляются все значения (*value1, …, valueN*), а затем происходит связывание полученных значений с *var1, …, varN*.

Оператор

*(let\* ((var1 value1)*

*…*

*(varN valueN)*

*body)*

отличается от *let* лишь тем, что выражения *value* могут ссылаться на предыдущие переменные *var*.

Переменные, создаваемые оператором *let*, называются локальными, то есть действительными в определенной области. Есть также глобальные переменные, которые действительны везде (в пределах пакета). Глобальная переменная может быть создана с помощью оператора *(defparameter symbol expression)*. Такая переменная будет доступна везде (в пределах пакета), кроме выражений, в которых создается локальная переменная с таким же именем.

Также в глобальном окружении можно задавать константы, используя оператор *(defconstant symbol expression)*.

Функции, определенные с помощью *defun* являются глобальными. Как и глобальные переменные, они могут быть использованы везде в пределах пакета. Кроме того, есть возможность определять локальные функции, которые, как и локальные переменные, доступны лишь внутри определенного контекста. Локальные функции могут быть определены с помощью конструкции *(labels ((fname parameters . body)\*) declaration\* expression\*)*. Локальная функция в *labels* может ссылаться на любые другие функции, определенные в этой же конструкции labels, в том числе и на саму себя.

**Практическая часть.**

**Задание 1.**

Переписать функцию how-alike, используя конструкции IF, AND/OR.

*(defun how\_alike (x y)*

*(cond ((or (= x y)(equal x y)) 'the\_same)*

*((and (oddp x)(oddp y)) 'both\_odd)*

*((and (evenp x)(evenp y)) 'both\_even)*

*(t 'difference)))*

*(defun how\_alike2 (x y)*

*( or (or (and (= x y) 'the\_same) (and (equal x y) 'the\_same))*

*(and (oddp x) (oddp y) 'both\_odd)*

*(and (evenp x) (evenp y) 'both\_even)*

*'difference*

*)*

*)*

*(defun how\_alike3 (x y)*

*(if (= x y) 'the\_same*

*(if (equal x y) 'the\_same*

*(if (oddp x)*

*(if (oddp y) 'both\_odd 'difference*

*)*

*(if (evenp x)*

*(if (evenp y) 'both\_even 'difference)*

*)*

*)*

*)*

*))*

**Задание 2.**

Дано два списка: названия стран и список столиц (4 шт.). Создать список из двухэлементных списков (страна-столица). Создать список из точечных пар (страна-столица).

*'(Финляндия Германия Нидерланды Норвегия)*

*'(Хельсинки Берлин Амстердам Осло)*

Функция для создания списка двухэлементных списков:

*(defun create\_list (lst1 lst2)*

*( cons*

*( cons (car lst1) (cons (car lst2) Nil) )*

*(cons ( cons (second lst1) (cons (second lst2) Nil ))*

*( cons*

*(cons (third lst1) (cons (third lst2) Nil))*

*(cons (cons (fourth lst1) (cons (fourth lst2) Nil)) Nil)*

*)*

*)*

*)*

*)*

*(create\_list '(Финляндия Германия Нидерланды Норвегия) '(Хельсинки Берлин Амстердам Осло))*

Функция для создания списка точечных пар:

*(defun create\_point\_pair (lst1 lst2)*

*( cons*

*( cons (car lst1) (car lst2) )*

*(cons ( cons (second lst1) (second lst2))*

*( cons*

*(cons (third lst1) (third lst2))*

*(cons (cons (fourth lst1) (fourth lst2)) Nil)*

*)*

*)*

*)*

*)*

*(create\_point\_pair '(Финляндия Германия Нидерланды Норвегия) '(Хельсинки Берлин Амстердам Осло))*

**Задание 3.**

По созданным в задании 2 спискам по столице найти страну, по стране найти столицу.

*'((ФИНЛЯНДИЯ . ХЕЛЬСИНКИ) (ГЕРМАНИЯ . БЕРЛИН) (НИДЕРЛАНДЫ . АМСТЕРДАМ) (НОРВЕГИЯ . ОСЛО))*

*'((ФИНЛЯНДИЯ ХЕЛЬСИНКИ) (ГЕРМАНИЯ БЕРЛИН) (НИДЕРЛАНДЫ АМСТЕРДАМ) (НОРВЕГИЯ ОСЛО))*

Функция для поиска столицы по стране в списке двухэлементных списков с использованием AND и OR:

*(defun search\_list\_by\_contry1 (lst contry)*

*(or*

*(and (eql (car (car lst)) contry) (car (cdr (car lst)))*

*)*

*(and (eql (car (second lst)) contry) (car (cdr (second lst ) ) )*

*)*

*(and (eql (car (third lst)) contry) (car (cdr (third lst)))*

*)*

*(and (eql (car (fourth lst)) contry) (car (cdr (fourth lst)))*

*)*

*'Unknown*

*)*

*)*

*(search\_list\_by\_contry1 '((ФИНЛЯНДИЯ ХЕЛЬСИНКИ) (ГЕРМАНИЯ БЕРЛИН) (НИДЕРЛАНДЫ АМСТЕРДАМ) (НОРВЕГИЯ ОСЛО)) 'Нидерланды)*

Функция для поиска страны по столице в списке двухэлементных списков с использованием AND и OR:

*(defun search\_list\_by\_capital1 (lst capital)*

*(or*

*(and (eql (car (cdr (car lst))) capital) (car (car lst))*

*)*

*(and (eql (car (cdr (second lst))) capital) (car (second lst))*

*)*

*(and (eql (car (cdr (third lst))) capital) (car (third lst))*

*)*

*(and (eql (car (cdr (fourth lst))) capital) (car (fourth lst))*

*)*

*'Unknown*

*)*

*)*

*(search\_list\_by\_capital1 '((ФИНЛЯНДИЯ ХЕЛЬСИНКИ) (ГЕРМАНИЯ БЕРЛИН) (НИДЕРЛАНДЫ АМСТЕРДАМ) (НОРВЕГИЯ ОСЛО)) 'Хельсинки)*

Функция для поиска столицы по стране в списке двухэлементных списков с использованием IF:

*(defun search\_list\_by\_contry2 (lst contry)*

*(if (eql (car (car lst)) contry) (car (cdr (car lst)))*

*(if (eql (car (second lst)) contry) (car (cdr (second lst ) ) )*

*(if (eql (car (third lst)) contry) (car (cdr (third lst)))*

*( if (eql (car (fourth lst)) contry) (car (cdr (fourth lst))) 'Unknown*

*)*

*)*

*)*

*)*

*)*

*(search\_list\_by\_contry2 '((ФИНЛЯНДИЯ ХЕЛЬСИНКИ) (ГЕРМАНИЯ БЕРЛИН) (НИДЕРЛАНДЫ АМСТЕРДАМ) (НОРВЕГИЯ ОСЛО)) 'ФИНЛЯНДИЯ)*

Функция для поиска страны по столице в списке двухэлементных списков с использованием IF:

*(defun search\_list\_by\_capital2 (lst capital)*

*(if (eql (car (cdr (car lst))) capital) (car (car lst))*

*(if (eql (car (cdr (second lst))) capital) (car (second lst))*

*(if (eql (car (cdr (third lst))) capital) (car (third lst))*

*( if (eql (car (cdr (fourth lst))) capital) (car (fourth lst)) 'Unknown*

*)*

*)*

*)*

*)*

*)*

*(search\_list\_by\_capital2 '((ФИНЛЯНДИЯ ХЕЛЬСИНКИ) (ГЕРМАНИЯ БЕРЛИН) (НИДЕРЛАНДЫ АМСТЕРДАМ) (НОРВЕГИЯ ОСЛО)) 'осло)*

Функция для поиска столицы по стране в списке двухэлементных списков с использованием COND:

*(defun search\_list\_by\_contry3 (lst contry)*

*(cond ((eql (car (car lst)) contry) (car (cdr (car lst))))*

*((eql (car (second lst)) contry) (car (cdr (second lst ) ) ))*

*((eql (car (third lst)) contry) (car (cdr (third lst))))*

*((eql (car (fourth lst)) contry) (car (cdr (fourth lst))))*

*('UNKNOWN)*

*)*

*)*

*(search\_list\_by\_contry3 '((ФИНЛЯНДИЯ ХЕЛЬСИНКИ) (ГЕРМАНИЯ БЕРЛИН) (НИДЕРЛАНДЫ АМСТЕРДАМ) (НОРВЕГИЯ ОСЛО)) 'ФИНЛЯНДИЯ)*

Функция для поиска страны по столице в списке двухэлементных списков с использованием COND:

*(defun search\_list\_by\_capital3 (lst capital)*

*(cond ((eql (car (cdr (car lst))) capital) (car (car lst)))*

*((eql (car (cdr (second lst))) capital) (car (second lst)))*

*((eql (car (cdr (third lst))) capital) (car (third lst)))*

*((eql (car (cdr (fourth lst))) capital) (car (fourth lst)))*

*('UNKNOWN)*

*)*

*)*

*(search\_list\_by\_capital3 '((ФИНЛЯНДИЯ ХЕЛЬСИНКИ) (ГЕРМАНИЯ БЕРЛИН) (НИДЕРЛАНДЫ АМСТЕРДАМ) (НОРВЕГИЯ ОСЛО)) 'Хельсинки)*

Функция для поиска столицы по стране в списке точечных пар с использованием AND и OR:

*(defun search\_pairs\_by\_contry1 (lst contry)*

*(or*

*(and (eql (car (car lst)) contry) (cdr (car lst))*

*)*

*(and (eql (car (second lst)) contry) (cdr (second lst ) )*

*)*

*(and (eql (car (third lst)) contry) (cdr (third lst))*

*)*

*(and (eql (car (fourth lst)) contry) (cdr (fourth lst))*

*)*

*'Unknown*

*)*

*)*

*(search\_pairs\_by\_contry1 '((ФИНЛЯНДИЯ . ХЕЛЬСИНКИ) (ГЕРМАНИЯ . БЕРЛИН) (НИДЕРЛАНДЫ . АМСТЕРДАМ) (НОРВЕГИЯ . ОСЛО)) 'Нидерланды)*

Функция для поиска страны по столице в списке точечных пар с использованием AND и OR:

*(defun search\_pairs\_by\_capital1 (lst capital)*

*(or*

*(and (eql (car (cdr (car lst))) capital) (car (car lst))*

*)*

*(and (eql (car (cdr (second lst))) capital) (car (second lst))*

*)*

*(and (eql (car (cdr (third lst))) capital) (car (third lst))*

*)*

*(and (eql (car (cdr (fourth lst))) capital) (car (fourth lst))*

*)*

*'Unknown*

*)*

*)*

*(search\_pairs\_by\_capital1 '((ФИНЛЯНДИЯ ХЕЛЬСИНКИ) (ГЕРМАНИЯ БЕРЛИН) (НИДЕРЛАНДЫ АМСТЕРДАМ) (НОРВЕГИЯ ОСЛО)) 'ХЕЛЬСИНКИ)*

Функция для поиска столицы по стране в списке точечных пар с использованием IF:

*(defun search\_pairs\_by\_contry2 (lst contry)*

*(if (eql (car (car lst)) contry) (cdr (car lst))*

*(if (eql (car (second lst)) contry) (cdr (second lst ) )*

*(if (eql (car (third lst)) contry) (cdr (third lst))*

*( if (eql (car (fourth lst)) contry) (cdr (fourth lst)) 'Unknown*

*)*

*)*

*)*

*)*

*)*

*(search\_pairs\_by\_contry2 '((ФИНЛЯНДИЯ . ХЕЛЬСИНКИ) (ГЕРМАНИЯ . БЕРЛИН) (НИДЕРЛАНДЫ . АМСТЕРДАМ) (НОРВЕГИЯ . ОСЛО)) 'ФИНЛЯНДИЯ)*

Функция для поиска страны по столице в списке точечных пар с использованием IF:

*(defun search\_pairs\_by\_capital2 (lst capital)*

*(if (eql (car (cdr (car lst))) capital) (car (car lst))*

*(if (eql (car (cdr (second lst))) capital) (car (second lst))*

*(if (eql (car (cdr (third lst))) capital) (car (third lst))*

*( if (eql (car (cdr (fourth lst))) capital) (car (fourth lst)) 'Unknown*

*)*

*)*

*)*

*)*

*)*

*(search\_pairs\_by\_capital2 '((ФИНЛЯНДИЯ ХЕЛЬСИНКИ) (ГЕРМАНИЯ БЕРЛИН) (НИДЕРЛАНДЫ АМСТЕРДАМ) (НОРВЕГИЯ ОСЛО)) 'ФИНЛЯНДИЯ)*

Функция для поиска столицы по стране в списке точечных пар с использованием COND:

*(defun search\_pairs\_by\_contry3 (lst contry)*

*(cond ((eql (car (car lst)) contry) (cdr (car lst)))*

*((eql (car (second lst)) contry) (cdr (second lst ) ) )*

*((eql (car (third lst)) contry) (cdr (third lst)))*

*((eql (car (fourth lst)) contry) (cdr (fourth lst)))*

*('UNKNOWN)*

*)*

*)*

*(search\_pairs\_by\_contry3 '((ФИНЛЯНДИЯ . ХЕЛЬСИНКИ) (ГЕРМАНИЯ . БЕРЛИН) (НИДЕРЛАНДЫ . АМСТЕРДАМ) (НОРВЕГИЯ . ОСЛО)) 'ФИНЛЯНДИЯ)*

Функция для поиска страны по столице ыв списке точечных пар с использованием COND:

*(defun search\_pairs\_by\_capital3 (lst capital)*

*(cond ((eql (car (cdr (car lst))) capital) (car (car lst)))*

*((eql (car (cdr (second lst))) capital) (car (second lst)))*

*((eql (car (cdr (third lst))) capital) (car (third lst)))*

*((eql (car (cdr (fourth lst))) capital) (car (fourth lst)))*

*('UNKNOWN)*

*)*

*)*

*(search\_pairs\_by\_capital3 '((ФИНЛЯНДИЯ ХЕЛЬСИНКИ) (ГЕРМАНИЯ БЕРЛИН) (НИДЕРЛАНДЫ АМСТЕРДАМ) (НОРВЕГИЯ ОСЛО)) 'Хельсинки)*