|  |  |
| --- | --- |
| 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 № 10**

|  |  |
| --- | --- |
| **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.

8. Написать рекурсивную версию (с именем reg-add) вычисления суммы чисел заданного списка.

Например: (reg-add (2 4 6)) → 12

|  |
| --- |
| (defun rec\_add\_h (lst res)  (cond ((null lst) res)  (t (rec\_add\_h (cdr lst)  (if (numberp (car lst)) (+ res (car lst)) res)))  )  )  (defun rec\_add (lst)  (rec\_add\_h lst 0)) |

9. Написать рекурсивную версию с именем recnth функции nth.

|  |
| --- |
| (defun rec\_nth (n lst)  (cond ((= n 0) (car lst))  ((null lst) nil)  (t (rec\_nth (- n 1) (cdr lst)))  )  ) |

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

|  |
| --- |
| (defun alloddr\_rec (lst state)  (cond ((null lst) state)  ((evenp (car lst)) nil)  (t (alloddr\_rec (cdr lst) t))  )  )  (defun alloddr (lst)  (alloddr\_rec lst nil)) |

\* (alloddr '(5 6 7 0)) =>NIL

\* (alloddr '(5 9)) => T

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

|  |
| --- |
| (defun mylast (lst)  (cond ((null (cdr lst)) lst)  (t (mylast (cdr lst)))  )  ) |

\* (mylast '(1 2 3)) => (3)

\* (mylast '(1 (2 3))) => ((2 3))

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

|  |
| --- |
| (defun get\_n\_sum(lst n)  ( cond  ((or (equal n -1) (null lst) ) 0)  (t (+ (car lst) (get\_n\_sum (cdr lst) (- n 1) )))  )  ) |

(get\_n\_sum '(2 4 5 7) 3) => 18

(get\_n\_sum '(2 4 5 7) 2) => 11

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

|  |
| --- |
| (defun lastodd\_help(lst res)  ( cond ((null lst) res)  ((oddp (car lst)) (lastodd\_help (cdr lst) (car lst)))  (t (lastodd\_help (cdr lst) res))  )  )  (defun lastodd(lst)  (lastodd\_help lst nil)) |

(lastodd '(4 5 6)) => 5

(lastodd '(4 6 8)) => NIL

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.

|  |
| --- |
| (defun squaring (lst)  (cond ((null lst) lst)  (t (cons (\* (car lst) (car lst)) (squaring (cdr lst))))  )  ) |

(squaring '(1 3 5 0)) => (1 9 25 0)

15. Написать функцию с именем select-odd, которая из заданного списка выбирает все нечетные числа.

|  |
| --- |
| (defun select-odd-recur (lst)  (cond  ((null lst) nil)  ((evenp (car lst)) (select-odd-recur (cdr lst)))  (t (cons (car lst) (select-odd-recur (cdr lst))))  )  )  (defun select-odd (lst)  (select-odd-recur lst)) |

(select-odd '(1 2 4 5 7)) => (1 5 7)

(select-odd '(2 4)) => NIL

16. Создать и обработать смешанный структурированный список с информацией:

ФИО, зарплата, возраст, категория(квалификация). Изменить зарплату, в зависимости от заданного условия, и подсчитать суммарную зарплату. Использовать композиции функций.

|  |
| --- |
| (defun create-info (lst lastname firstname middlename salary age skill)  (cons (list lastname firstname middlename salary age skill) lst)  )  (defun create-table (lst1 lst2)  (list lst1 lst2)  )  (defun update-salary-all(lst new\_salary)  (mapcar  #'(lambda (x)  (setf (cadddr x) new\_salary)  ) lst  )  )  (defun update-salary-person (lst new\_salary pred)  (mapcar  #'(lambda (x)  (if (funcall pred x)  (setf (caddr x) new\_salary)  )  ) lst  )  )  (defun sum-salary (lst)  (cond  ((null lst) 0)  (t (+ (car (cdddar lst)) (sum-salary (cdr lst))) )  )  ) |

(update-salary-all (create-info '((Unt Ar Ol 4000 20 2)) 'Vetr 'Kate 'Ig 5500 22 1) 12000)

* (12000 12000)

(update-salary-person (create-info '((Unt Ar Ol 4000 23 2)) 'Vetr 'Kate 'Ig 5500 24 6) 12000 #'(lambda (x) (and (equal 'Vetr (car x)))))

* (12000 NIL)

(sum-salary (create-info '((Unt Ar Ol 4000 23 2)) 'Vetr 'Kate 'Ig 5500 24 6))

* 9500