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 №8 по дисциплине «Функциональное и логическое программирование»

Тема Использование функционалов

Студент Романов А.В.

Группа ИУ7-63Б

Оценка (баллы) _____

Преподаватель Толпинская Н.Б., Строганов Ю. В.

Москва — 2021 г.

Задание 1

Постановка задачи

Напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда все элементы списка — числа и все элементы списка — любые объекты.

Решение

```
(defun mult-numbers (acc lst)
  (reduce #'* lst :initial-value acc))

(defun mult-objects (mp lst)
  (reduce
    #'(lambda (acc el)
      (if (numberp el)
        (* acc el)
        acc))
  lst :initial-value mp))
```

Задание №2

Постановка задачи

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)

Решение

```
(defun rec-add-to-end (lst elem)
  (cond
    ((cdr lst) (rec-add-to-end (cdr lst) elem))
    ((listp elem) (setf (cdr lst) elem))
    (t (setf (cdr lst) (cons elem Nil))))
  lst)

(defun add-to-end (lst elem)
  (if (null lst)
    (cons elem Nil)
    (rec-add-to-end lst elem)))

(defun select-between (lst bot top)
```

```
  (reduce
    #'(lambda (acc el)
      (if (and (> el bot) (< el top))
        (add-to-end acc el)
    acc))
  lst :initial-value ()))
```

Задание №3

Постановка задачи

Что будет результатом `(mapcar 'вектор '(570-40-8))`?

Решение

Функции `вектор` не существует, программа завершится с ошибкой.

Задание №4

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

Постановка задачи

Решение

```
(defun rec-minus-n-internal (lst n acc)
  (if (car lst)
    (cond ((listp (car lst))
      (add-to-end acc (rec-minus-n (car lst) n ())))
    ((numberp (car lst))
      (rec-minus-n (cdr lst) n (add-to-end acc (- (car lst) n))))
    (t
      (add-to-end acc (car lst))))
  acc))

(defun rec-minus-n (lst n)
  (rec-minus-n-internal lst n ()))

(defun minus-n (lst n)
  (mapcar #'(lambda (el)
    (cond
      ((listp el) (minus-n el n))
      ((numberp el) (- el n))
      (t el)))
```

```
20    lst))
```

Задание №5

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

Постановка задачи

Решение

```
1 (defun first-lst (lst)
2   (if (listp (car lst))
3     (car (car lst))
4     (first-lst (cdr lst))))
```

Задание №6

Найти сумму числовых элементов смешанного структурированного списка.

Постановка задачи

Решение

```
(defun sum-list (lst)
  (reduce
    #'(lambda (acc x)
    (cond
      ((listp x) (+ acc (sum-list x)))
      ((numberp x) (+ acc x))
      (t acc)))
  lst :initial-value 0))
```

Контрольные вопросы

Вопрос 1. Порядок работы и варианты использования функционалов

Функционалы — функции, которые в качестве одного из аргументов используют другую функцию.

Ответ. Применяющие функционалы

Данные функционалы просто применить переданную в качестве аргумента функцию к переданным в качестве аргументов параметрам.

1. `funcall` – вызывает функцию-аргумент с остальными аргументами;

   Синтаксис: `(funcall #'fun arg1 arg2 ... argN)`. Пример: `(funcall #'* 1 2 3)`

2. `apply` – вызывает функцию-аргумент с аргументами из списка, переданного вторым аргументом в `apply`.

   Синтаксис: `(apply #'fun arg-lst)`. Пример: `(apply #'* '(1 2 3))`

Отображающие функционалы

Отображения множества аргументов в множество значений позволяют многократно применить функцию. Данные функции берут аргумент, являющийся функциональным объектом и многократно применяет эту фукнцию к элементам переданного в качестве аргумента списка.

1. `mapcar` — функция `fun` применяется ко всем первым элементам списков-аргументов, затем ко всем вторым аргументам и так до тех пор, пока не кончатся элементы самого короткого списка. К полученным результатам применения функции применяется функция `list`, поэтому на выходе функции всегда будет список;

Синтаксис: `(mapcar #'fun lst1 lst2 ... lstN)`. Пример: `(mapcar #'(lambda (x y) (* x y)) '(1 2 3) '(4 5 6)) -> (4 10 18)`

2. `maplist` — в качестве аргумента на каждой итерации функция `fun` получает хвост списка, который использовался на предыдущей итерации (изначально функция получает сам список-аргумент). Если функция принимает несколько аргументов и передано несколько аргументов-списков, то они передаются функции `fun` в том же порядке, в которым идут в `maplist`;

   Синтаксис: `(maplist #'fun lst1 lst2 ... lstN)`. Пример: `(maplist #'(lambda (x y) (+ (car x) (car y))) '(1 2 3 4) '(6 5 4)) -> (list (+ 1 6) (+ 2 5) (+ 2 4))`

3. `mapcan` — работает аналогично `mapcar`, только соединяет результаты функции с помощью функции `nconc`. Может использоваться как `filter-map` из некоторых современных языков (например, функция, которая оставляет только четные числа и возводит их в квадрат);

   Синтаксис: `(mapcan #'fun lst1 lst2 ... lstN)`. Пример: `(mapcan #'(lambda (x) (and (oddp x) (list (* x x)))) '(1 2 3 4 5 6 7 8 9)) -> (1 9 25 49 81)`

4. `mapcon` — работает аналогично `maplist`, только соединяет результаты функции с помощью функции `nconc`.

   Синтаксис: `(mapcon #'fun lst1 lst2 ... lstN)`

5. `reduce` – применяет функцию к элементам списка каскадно. Накапливает значение, применяя функцию-аргумент к результату предыдущей итерации и следующему элементу списка (изначально инициализирует результат первым элементом, в случае пустого списка пытается вызвать функцию-аргумент без аргументов и вернуть значение);

   Синтаксис: `(reduce #'aggregator lst)`. Пример: `(reduce #'oddp '(1 2 3 4 5 6)) -> (2 4 6)`.