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 курсу «Функциональное и логическое программирование»

Тема Использование функционалов

Студент Кононенко С.С.

Группа ИУ7-63Б

Оценка (баллы) _____

Преподаватели 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 аргумента-списка, когда: а) все элементы списка – числа, б) элементы списка – любые объекты.

Решение.

Листинг 1 – Решение задания 1

```
(defun make-mult-lst-nums (lst init)
    (mapcar #'(lambda (el) (* el init)) lst))
(defun make-mult-lst-els (lst init)
    (mapcar #'(lambda (el) (cond ((listp el) (make-mult-lst-nums el init))
                          ((numberp el) (* el init))
                          (T el))) lst))
```

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.

Решение.

Листинг 2 – Решение задания 2

```
(defun make-select-between (lst l r)
    (sort (reduce #'(lambda (acc el) (if (and (> el l) (< el r))
                                (append acc (cons el Nil))
                                acc))
                                lst :initial-value ())
    #'<))
```

Задание 3

Постановка задачи. Что будет результатом?

Решение.

Листинг 3 – Решение задания 3

```
(mapcar 'bektop '(570-40-8)) ; The function COMMON-LISP-USER::BEKTOP is undefined.
```

Задание 4

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

Решение.

Листинг 4 – Решение задания 4

```
(defun make-minus-10 (lst)
    (mapcar #'(lambda (el) (cond ((numberp el) (- el 10))
                          (T el))) lst))
```

Задание 5

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

Решение.

Листинг 5 – Решение задания 5

```
(defun make-first-from-sublst (lst)
    (cond ((null lst) Nil)
    ((listp (car lst)) (caar lst))
    (T (make-first-from-sublst (cdr lst)))))
```

Задание 6

Постановка задачи. Найти сумму числовых элементов смешанного структурированного списка.

Решение.

Листинг 6 – Решение задания 6

```
(defun make-sum-lst-nums (lst)
    (reduce #'(lambda (acc el) (cond ((listp el) (+ acc (make-sum-lst-nums el)))
```

```
3                    ((numberp el) (+ acc el))
4                    (T acc))) (cons 0 lst)))
```

Ответы на контрольные вопросы

Вопрос 1. Порядок работы и варианты использования функционалов.

Ответ. Функционалы – функции, которые в качестве одного из аргументов используют другую функцию (специальным образом).

Применяющие функционалы

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

Виды:

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, некоторый аналог цикла из императивных языков.

Данные функции берут аргумент, являющийся функциональным объектом (функцией), и многократно применяет эту фукнцию к элементам переданного в качестве аргумента списка.

1. `mapcar`;

   Сначала 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) '(6 5 4)) -> (list (+ 1 6) (+ 2 5) (+ 2 4))`

2. `maplist`;

   Работает похожим на `mapcar` образом, но 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. `find-if`;

Возвращает первый элемент списка, для которого функция-предикат возвращает не `Nil`.

Синтаксис: `(find-if #'predicat lst)`

Пример: `(find-if #'oddp '(2 4 1)) -> 1`

6. `remove-if`, `remove-if-not`;

   Данные функции возвращают список, в котором находятся только те элементы, для которых функция-предикат вернула не `Nil` (для `remove-if-not` вернула `Nil`).

   Синтаксис: `(remove-if #'predicat lst)`

   Пример: `(remove-if #'oddp '(1 2 3 4 5 6)) -> (2 4 6)`

7. `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)`

8. `every`;

   Возвращает `T`, если функция-предикат возвращает не `Nil`, для всех элементов списка-аргумента.

   Синтаксис: `(every #'predicat lst)`

   Пример: `(every #'oddp '(1 3 5 7)) -> T`

9. `some`;

   Возвращает `T`, если функция-предикат возвращает не `Nil`, хотя бы для одного элемента списка-аргумента.

   Синтаксис: `(some #'predicat lst)`

   Пример: `(some #'oddp '(1 2 3 4 5)) -> T`