**Лабораторная работа № 4. Тема: Рекурсия. Функционалы**

Целью работы является изучение и использование при написании программ различных видов рекурсии и функционалов.

1. Рекурсия. Различные формы рекурсии.

Основная идея рекурсивного определения заключается в том, что функцию можно с помощью рекуррентных формул свести к некоторым начальным значениям, к ранее определенным функциям или к самой определяемой функции, но с более «простыми» аргументами. Вычисление такой функции заканчивается в тот момент, когда оно сводится к известным начальным значениям.

Рекурсивная функция, во-первых, содержит всегда хотя бы одну рекурсивную ветвь и условие окончания. Во-вторых, когда функция доходит до рекурсивной ветви, то функционирующий процесс приостанавливается, и новый такой же процесс запускается сначала, но уже на новом уровне. Прерванный процесс запоминается и начнет исполняться лишь после окончания нового процесса. В свою очередь, новый процесс может приостановиться, ожидать и т. д.

Различают следующие формы рекурсии:

1. простая рекурсия;
2. параллельная рекурсия;
3. взаимная рекурсия.

Рекурсия называется простой, если вызов функции встречается в некоторой ветви лишь один раз. Простой рекурсии в процедурном программировании соответствует обыкновенный цикл.

Для примера напишем функцию вычисления чисел Фибоначчи (F(1)=1; F(2)=1; F(n)=F(n-1)+F(n-2) при n>2):

(DEFUN FIB (N)

(IF (> N 0)

(IF (OR N=1 N=2) 1

(+ (FIB (- N 1)) (FIB (- N 2))))

‘ОШИБКА\_ВВОДА))

Рекурсию называют параллельной, если она встречается одновременно в нескольких аргументах функции:

(DEFUN f ...

...(g ... (f ...) (f ...) ...)

...)

Рассмотрим использование параллельной рекурсии на примере преобразования списочной структуры в одноуровневый список:

(DEFUN PREOBR (L)

(COND

((NULL L) NIL)

((ATOM L) (CONS (CAR L) NIL))

(T (APPEND

(PREOBR (CAR L))

(PREOBR (CDR L))))))

Рекурсия является взаимной между двумя и более функциями, если они вызывают друг друга:

(DEFUN f ...

...(g ...) ...)

(DEFUN g ...

...(f ...) ...)

Для примера напишем функцию обращения или зеркального отражения списка в виде двух взаимно рекурсивных функций следующим образом:

(DEFUN obr (l)

(COND ((ATOM l) l)

(T (per l nil))))

(DEFUN per (l res)

(COND ((NULL l) res)

(T (per (CDR l)

(CONS (obr (CAR l)) res)))))

**Применяющие функционалы.**

Функции, которые позволяют вызывать другие функции, т. е. применять функциональный аргумент к его параметрам называют применяющими функционалами. Они дают возможность интерпретировать и преобразовывать данные в программу и применять ее в вычислениях.

APPLY

APPLY является функцией двух аргументов, из которых первый аргумент представляет собой функцию, которая применяется к элементам списка, составляющим второй аргумент функции APPLY:

(APPLY fn список)

\_(SETQ a ‘+) ⇨ +

\_(APPLY a ‘(1 2 3)) ⇨ 6

\_(APPLY ‘+ ‘(4 5 6)) ⇨ 15

FUNCALL.

Функционал FUNCALL по своему действию аналогичен APPLY, но аргументы для вызываемой функциион принимает не списком, а по отдельности:

(FUNCALL fn x1 x2 ... xn)

\_(FUNCALL ‘+ 4 5 6) ⇨ 15

FUNCALL и APPLY позволяют задавать вычисления (функцию) произвольной формой, например, как в вызове функции, или символом, значением которого является функциональный объект. Таким образом, появляется возможность использовать синонимы имени функции. С другой стороны, имя функции можно использовать как обыкновенную переменную, например для хранения другой функции (имени или лямбда-выражения), и эти два смысла (значение и определение) не будут мешать друг другу:

\_(SETQ list ‘+) ⇨ +

\_(FUNCALL list 1 2) ⇨ 3

\_(LIST 1 2) ⇨ (1 2)

**Отображающие функционалы.**

Отображающие или MAP-функционалы являются функциями, которые являются функциями, которые некоторым образом отображают список (последовательность) в новую последовательность или порождают побочный эффект, связанный с этой последовательностью. Каждая из них имеет более двух аргументов, значением первого должно быть имя определенной ранее или базовой функции, или лямбда-выражение, вызываемое MAP-функцией итерационно, а остальные аргументы служат для задания аргументов на каждой итерации. Количество аргументов в обращении к MAP-функции должно быть согласовано с предусмотренным количеством аргументов у аргумента-функции. Различие между всеми MAP-функциями состоит в правилах формирования возвращаемого значения и механизме выбора аргументов итерирующей функции на каждом шаге.

Рассмотрим основные типы MAP-функций.

### MAPCAR.

Значение этой функции вычисляется путем применения функции fn к последовательным элементам xi списка, являющегося вторым аргументом функции. Например, в случае одного списка получается следующее выражение:

(MAPCAR fn ‘(x1 x2 ... xn))

В качестве значения функционала возвращается список, построенный из результатов вызовов функционального аргумента MAPCAR.

\_(MAPCAR ‘LISTP ‘((f) h k (i u)) ⇨ (T NIL NIL T)

### MAPLIST.

MAPLIST действует подобно MAPCAR, но действия осуществляет не над элементами списка, а над последовательными CDR этого списка.

\_(MAPLIST ‘LIST ‘((f) h k (i u)) ⇨ (T T T T)

\_(MAPLIST ‘CONS ‘(a b c) ‘(1 2 3)) ⇨ (((a b c) 1 2 3) ((b c) 2 3) ((c ) 3))

Функционалы MAPCAR и MAPLIST используются для программирования циклов специального вида и в определении других функций, поскольку с их помощью можно сократить запись повторяющихся вычислений.

Функции MAPCAN и MAPCON являются аналогами функций MAPCAR и MAPLIST. Отличие состоит в том, что MAPCAN и MAPCON не строят, используя LIST, новый список из результатов, а объединяют списки, являющиеся результатами, в один список.

Задание 1.

С помощью базовых функций и рекурсии определите следующие функции:

Варианты 1-10

1. Возвращение последнего элемента списка.
2. Удаление последнего элемента списка
3. Проверка одноуровневости списка (возвращает Т, если ни один элемент списка не является списком, и NIL в противном случае).
4. Возвращение первого элемента списка, который является атомом. Например: (ПЕРВЫЙ\_АТОМ '(((1 2) 3) 4)) 🡪 4

Варианты 11-27

1. Возвращение первого атома списка с учетом многоуровневости списка.

(ПЕРВЫЙ\_АТОМ1 '(((a b)) c d)) 🡪 A

1. Удаление из списка первого вхождения данного элемента на верхнем уровне.

Например:

(УДАЛЕНИЕ '(1 2 (1 2) 3 4) '(1 2)) 🡪 (1 2 3 4)

1. Обращение списка и разбиение его на уровни.

Например: (ОБРАЩ\_РАЗБ '(1 2 3 4)) 🡪 ((((4) 3) 2) 1)

1. Преобразование списка (А В С) к виду (А (В (С))).

Например: (РАЗБИВКА '(a b c)) -🡪 (A (B (C)))

Задание 2

Определите функционалы.

Варианты 1-10

1. Определите функциональный предикат (**КАЖДЫЙ *пред******список***), который истинен в том и только в том случае, когда являющийся функциональным аргументом предикат ***пред*** истинен для всех элементов списка ***список***.
2. Определите функциональный предикат (НЕКОТОРЫЙ пред список), который истинен, когда предикат пред истинен хотя бы для одного элемента списка список.

Варианты 11-27

1. Определите функционал (**MAPLIST** ***fn*** ***список***) для одного списочного аргумента.
2. С помощью отображающих функционалов определите функцию (**УДАЛИТЬ-ЕСЛИ *пред******список***), удаляющую из списка элементы, которые обладают свойством, наличие которого проверяет предикат.

**Отчет по лабораторной работе**

Отчет по лабораторной работе должен содержать:

-задание на лабораторную работу;

-протоколы работы в LispWorks