МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)
Кафедра информационных систем и технологий

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«___» _____ 2017 г.

Определение и использование функций. Нисходящая и восходящая рекурсия

Методические указания к практической работе №4 по дисциплине «Рекурсивно-логическое и функциональное программирование» для студентов, обучающихся по направлению 09.03.02 «Информационные системы и технологии»

Курск 2017

УДК 004
Составитель И.В. Зотов

Рецензент
Кандидат технических наук Ю.А. Халин

Определение и использование функций. Нисходящая и восходящая рекурсия: методические указания к практической работе / Юго-Зап. гос. ун-т; сост. И.В. Зотов. Курск, 2017. 12 с. Библиогр.: с. 12.

Приводится описание базовых конструкций языка Лисп, необходимых для определения и вызова функций, включая функции с нисходящей и восходящей рекурсией. Даются практические примеры использования этих конструкций с иллюстрациями и варианты заданий.

Методические указания предназначены для студентов, обучающихся по направлению 09.03.02 «Информационные системы и технологии». Могут использоваться также студентами, обучающимися по направлениям, связанным с вычислительной техникой и интеллектуальными информационными системами.

Текст печатается в авторской редакции.

Подписано в печать 10.11.2017. Формат 60x84 1/16.
Усл.печ. л. 0,6. Уч.-изд. л. 0,5. Тираж 100 экз. Заказ 1754. Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

1. Цель работы 4
2. Краткие теоретические сведения 4
3. Формулировка задания 9
4. Варианты заданий 9
5. Пример решения задачи 10
6. Контрольные вопросы 11
7. Содержание отчета 12
Библиографический список 12

1. Цель работы

Целью данной работы является приобретение умений в составлении и отладке Lisp-программ, содержащих определения пользовательских функций, включая рекурсивные функции с нисходящей и восходящей рекурсией.

2. Краткие теоретические сведения

Программа на языке Лисп представляет собой набор определений функций в любом порядке. Программа запускается вызовом одной из определенных функций. Вызов должен содержать фактические параметры. При вызове функций значения формальных параметров запоминаются в стеке. Им передается значение фактических параметров (значение формальных параметров восстанавливается после выполнения вызываемой функции). Далее происходит вызов функции с фактическими аргументами. Возникает последовательность вызовов, заканчивающаяся вызовом базовой функции. Вычисленное значение функции возвращается формальному параметру, на месте которого располагается вызов функции.

Программист может использовать встроенные функции Лиспа, специфические функции среды HomeLisp, а также определять собственные функции. Определить новую функцию можно с помощью функции DEFUN, которая в случае успеха возвращает имя вновь определенной функции:

```
(DEFUN имя_функции (список_параметров) <последовательность_форм>)
```

Вызвать определенную функцию можно по ее имени, указав требуемое число аргументов.

В нижеследующем примере определяется функция CCONS, которая создает список из трех параметров – X, Y и Z. Далее эта функция вызывается при X=1, Y=(2 3), Z=(4).

Пример:

```
(DEFUN CCONS (X Y Z) (CONS (CONS X Y) Z))

(CCONS 1 '(2 3) '(4))
==> ((1 2 3) 4)
```

Особое место в функциональном программировании занимают рекурсивные функции. Рекурсивной будет любая функция, которая явно либо косвенно (через другую или другие функции) вызывает сама себя. Косвенная рекурсия используется довольно редко, поэтому оставим ее за рамками рассмотрения; остановимся только на явной рекурсии.

Использование рекурсивных функций позволяет писать очень компактные программы, которые решают довольно сложные задачи. Часто к рекурсивному построению программы приводит характер объекта обработки. Например, работа с деревьями естественным образом порождает рекурсивные алгоритмы. Однако и не рекурсивные объекты можно эффективно обрабатывать рекурсивными алгоритмами и функциями.

Чтобы рекурсивная функция работала корректно, нужно правильно определить начальные условия, условия продолжения и условия завершения рекурсии. Ситуации, в которых рекурсия должна завершиться, обычно называют терминальными случаями. Напротив, те ситуации, в которых рекурсия будет продолжена, принято называть нетерминальными случаями.

Различают два вида рекурсии: нисходящую и восходящую. При нисходящей рекурсии вычисление значения функции происходит уже после наступления терминального случая, когда осуществляется выход из серии рекурсивных вызовов функции. Иными словами, вычисления производятся на «спуске» из рекурсии. При восходящей рекурсии, напротив, значение функции вычисляется до наступления терминального случая, перед «погружением» в рекурсивный вызов. Другими словами, вычисления производятся на «подъеме» в рекурсию.

Остановимся более подробно на нисходящей рекурсии. В качестве иллюстрации будем рассматривать задачу суммирования элементов одноуровневого списка.

Пусть требуется найти сумму элементов списка (A_1 A_2 A_3 … A_{i-1} A_i A_{i+1} … A_{n-1} A_n), т.е. значение $S = A_1 + A_2 + A_3 + \ldots + A_{i-1} + A_i + A_{i+1} + \ldots + A_{n-1} + A_n$. При нисходящей рекурсии сумма будет вычисляться в следующем порядке: $S = A_1 + (A_2 + (A_3 + \ldots + (A_{i-1} + (A_i + (A_{i-1} + \ldots + (A_{n-1} + (A_n + (0))))\ldots)$. Вначале она будет нулевой, затем к ней добавится голова подсписка (A_n), далее – голова подсписка (A_{n-1} A_n), голова подсписка (A_{n-2} A_{n-1} A_n), и т.д. К моменту рассмотрения подсписка (A_i A_{i+1} … A_{n-1} A_n) уже будет получена сумма

элементов его хвоста, т.е. подсписка (A_{i+1} … A_{n-1} A_n): $S = A_{i+1} + … + A_{n-1} + A_n$. Таким образом, на каждом шаге функция суммирования будет добавлять голову текущего подсписка к сумме элементов его хвоста. Это и есть нетерминальный случай рекурсии. Терминальный случай наступит тогда, когда текущий подсписок станет пустым. Значение суммы будет равно нулю.

Все сказанное выше можно представить в компактной форме:

```
Сумма = 0, если список пуст
Сумма = голова списка + сумма элементов хвоста списка
```

Здесь первая строчка описывает терминальный случай, а вторая – нетерминальный. Отметим, что терминальных случаев в определении функции может быть более одного, что зависит от особенностей решаемой задачи.

Обозначая функцию суммирования через SUM, а исходный список через LST, непосредственно по приведенной выше компактной записи можно получить определение функции на Лиспе:

```
(DEFUN SUM (LST)
(COND
      ((NULL LST) 0)
      (T (+ (CAR LST) (SUM (CDR LST)))))
))
```

Пример вызова функции:

```
(SUM '(1 2 3 4))
==> 10
```

Рисунок 1 иллюстрирует порядок работы функции SUM для приведенного вызова.

Алгоритм разработки функции, использующей нисходящую рекурсию, включает следующие шаги:

a) составить определение функции для терминальных случаев (т.е. задать определение функции, когда список станет или будет пустым, когда число станет или будет равным 0 и т.д.);
b) для нетерминального случая предположить, что уже имеется определение разрабатываемой функции. Используя вызов этого определения, примененный к остатку списка, и элемент, предшествующий этому остатку, составить выражение, которое вычисляет значение функции для этого случая;

с) объединить пункты 1 и 2 в одно условное выражение, записав в нем также условие наступления терминального случая.

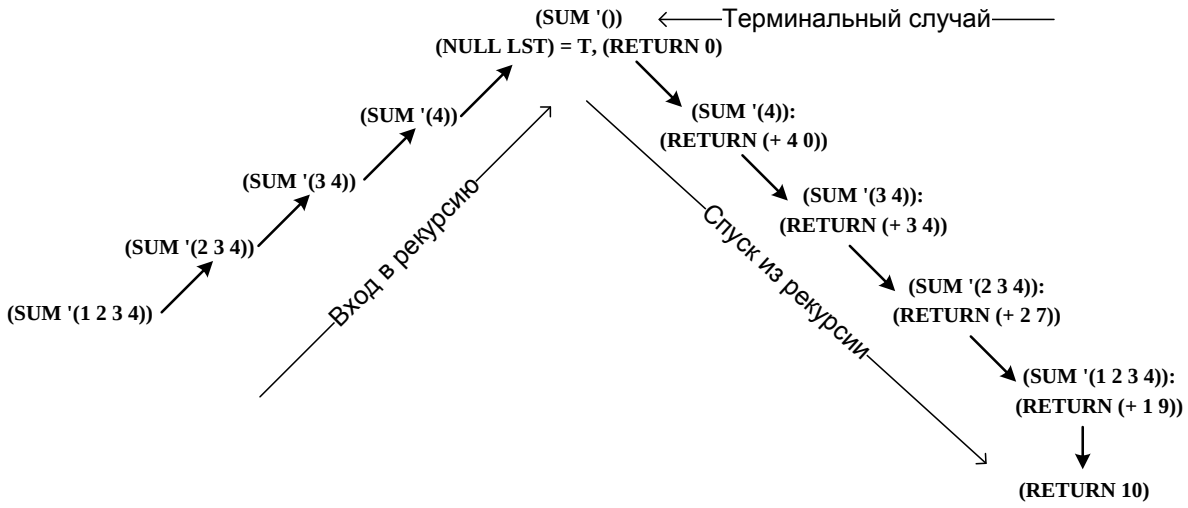

Рисунок 1. Порядок работы функции SUM
при нисходящей рекурсии

Теперь рассмотрим особенности восходящей рекурсии. В качестве иллюстрации вновь будем использовать задачу суммирования элементов одноуровневого списка.

При восходящей рекурсии сумма будет вычисляться в следующем порядке: $S = ((\ldots(((\ldots(((((0) + A_1) + A_2) + A_3) + \ldots + A_{i-1}) + A_i) + A_{i-1}) + \ldots + A_{n-1}) + A_n)$. Вначале она будет нулевой, затем к ней добавится голова исходного списка, далее – голова хвоста исходного списка, голова хвоста хвоста этого списка, и т.д. К моменту рассмотрения подсписка $(A_i\ A_{i+1}\ \ldots\ A_{n-1}\ A_n)$ уже будет получена сумма предшествующих элементов, т.е. подсписка $(A_1\ A_2\ \ldots\ A_{i-1})$: $S = A_1 + A_2 + \ldots + A_{i-1}$. Таким образом, на каждом шаге функция суммирования будет добавлять голову текущего хвоста к сумме всех предшествующих элементов. Это и будет описывать нетерминальный случай рекурсии. Терминальный случай наступит тогда, когда рассматриваемый хвост станет пустым. Значение суммы к этому моменту будет сформировано.

Все сказанное выше можно записать в компактной форме:

```
Сумма = 0
Если список пуст, вернуть Сумму
Сумма = Сумма + голова списка, список = хвост списка
```

Здесь первая строчка – инициализация, которая выполняется за пределами рекурсивной функции. Вторая строчка описывает терминальный случай, а третья – нетерминальный.

Из приведенной записи видно, что для формирования суммы необходима дополнительная переменная, внешняя по отношению к рекурсивной функции. Эту переменную можно задать во вспомогательной функции-обертке, в задачу которой будет входить запуск основной рекурсивной функции и получение результата. Именно такой вариант реализован в нижеследующем примере:

```
;; SUM – функция-обертка
;; MAKE_SUM – основная рекурсивная функция

(DEFUN MAKE_SUM (S LST)
(COND
      ((NULL LST) S)
      ( T (MAKE_SUM (+ S (CAR LST)) (CDR LST)))
))

(DEFUN SUM (LST) (MAKE_SUM 0 LST))
```

Пример вызова функции:

```
(SUM '(1 2 3 4))
==> 10
```

Рисунок 2 иллюстрирует порядок работы функции SUM для приведенного вызова.

Алгоритм разработки функции, использующей восходящую рекурсию, включает следующие шаги:

a) составляется список параметров основной рекурсивной функции. В этот список помимо основных параметров вводятся дополнительные параметры, в которых будет формироваться результат;
b) записывается определение основной рекурсивной функции для терминальных случаев (количество случаев зависит от количества используемых списков). В этом определении значение функции равно значению дополнительного параметра, в котором формируется результат;
c) для нетерминального случая записывается вызов определяемой основной функции, в котором значение параметра определяется на основании предыдущих значений и значений очередных элементов списка;
d) пункт b) и c) объединяются в одном условном выражении;

e) составляется определение функции-обертки, в которой осуществляется вызов основной рекурсивной функции с начальными значениями вспомогательных параметров.

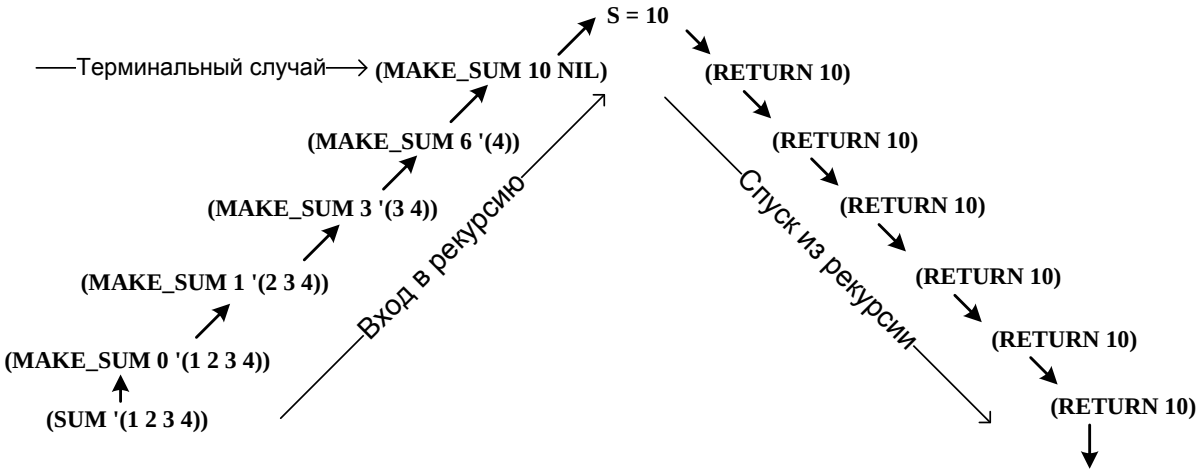

Рисунок 2. Порядок работы функции SUM
при восходящей рекурсии

3. Формулировка задания

Разработать рекурсивную функцию для решения задачи в соответствии с вариантом задания. Показать несколько вариантов вызова функции. Правильность работы программы подтвердить снимками экрана.

Примечание. Выбор вида рекурсии (нисходящая или восходящая) определяется студентом.

4. Варианты заданий

a) Написать функцию, которая из данного одноуровневого списка строит список списков его элементов, например: (a b) -> ((a) (b)).

b) Определить функцию, которая преобразует исходный список в список с одним уровнем скобок, например: (a (b c) a b (c (c)))) -> (a b c a b c c).

c) Определить функцию CDRN такую, что (CDRN N L) будет эквивалентна функции (CDD..DR L), в имени которой имеется N букв D.

d) Написать функцию CARN такую, что (CARN N L) будет эквивалентна функции (CAA..AR L), в имени которой имеется N букв A.

e) Определить предикат (NOSUBS L), который принимает значение T, если список L не имеет подсписков, и NIL-в противном случае.

f) Написать функцию НОД (X Y), позволяющую найти наибольший общий делитель чисел X и Y.

g) Определить функцию на Lisp, осуществляющую циклическую перестановку элементов в списке со сдвигом влево, т.е. (f g h j) -> (g h j f).

h) Написать функцию на Lisp, осуществляющую циклическую перестановку элементов в списке со сдвигом вправо, т.е. (f g h j) -> (j f g h).

i) Определить функцию (COUNTSUBS L), которая возвращает число подсписков в заданном списке L. Рассматривать подсписки всех уровней вложенности.

5. Пример решения задачи

Написать программу, которая переворачивает список со всеми его подсписками.

Текст программы:

```
(DEFUN REVERSE (lst)
(COND ((NULL lst) NIL)
       ((ATOM (CAR LST)) (APPEND (REVERSE (CDR lst)) (LIST (CAR LST))) )
       ( T (APPEND (REVERSE (CDR lst)) (LIST (REVERSE (CAR LST)))) )
))
```

В приведенной программе использовалась функция APPEND, обеспечивающая добавление элемента (атома или списка) в конец списка. Функция LIST применялась для преобразования атома в список. Предикат ATOM использовался для проверки аргумента на атом.

Результат работы программы в HomeLisp дан ниже.

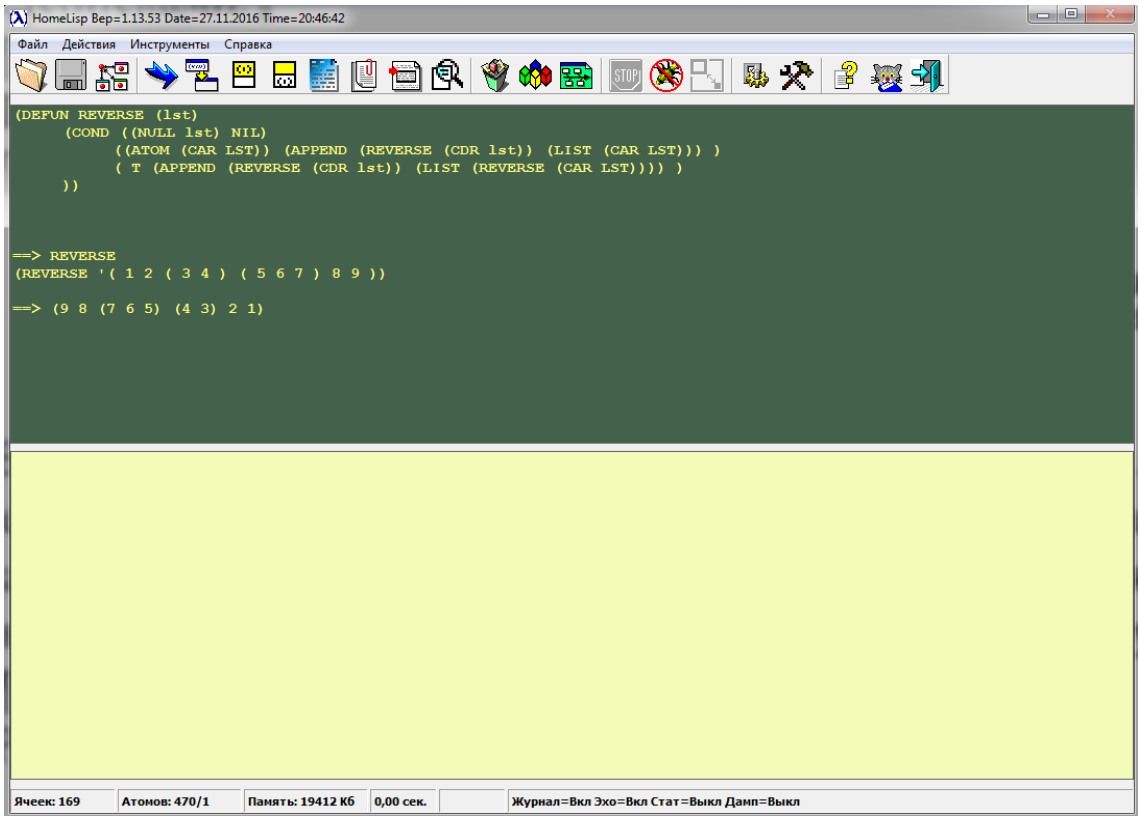

Рисунок 3. Результат работы программы

6. Контрольные вопросы

1. Что понимается под рекурсией?
2. Какие виды рекурсии поддерживаются в Лиспе? Чем они отличаются друг от друга?
3. Что такое «терминальный случай»?
4. Сколько терминальных случаев может быть в определении функции?
5. Верно ли утверждение, что функция не может содержать более одного нетерминального случая?
6. Сформулируйте алгоритм написания функции при нисходящей рекурсии.
7. Что такое функция-обертка?
8. Сколько раз функция может рекурсивно вызывать сама себя в одном выражении?
9. Что понимается под «спуском» из рекурсии?
10. Назовите преимущества и недостатки использования рекурсивных функций.

7. Содержание отчета

Результаты выполнения практической работы оформляются в виде отчета, который должен содержать следующие разделы:

- тема работы;
- цель работы;
- индивидуальный вариант задания;
- текст программы;
- снимки экрана, демонстрирующие выполнение задания;
- выводы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Сергиевский, Георгий Максимович. Функциональное и логическое программирование [Текст] : учебное пособие / Г. М. Сергиевский, Н. Г. Волченков. - М. : Академия, 2010. - 320 с.

2. Шалимов П. Ю. Функциональное программирование [Текст] : учебное пособие. - Издательство БГТУ, 2003. - 160 с.

3. Городняя Л. В. Основы функционального программирования [Электронный ресурс] : учебное пособие. - М. : Интернет-Университет Информационных Технологий, 2004. – 217 с. - Режим доступа: http://biblioclub.ru/index.php?page=book_red&id=233773