59. Понятие о функциональном стиле программирования. Язык функционального программирования Lisp: определение функций, простые и условные формы. Рекурсивная обработка списков на примере поиска элемента в списке

Идея функционального программирования заключается в том, что программа полностью состоит из функций. Процедурная программа состоит из последовательности оперторов и предложений, управляющих последовательностью их выполнения. Типичными опреаторами являются операторы присваивания и передачи управления, операторы ввода\вывода и специальные предложения для органицации циклов. Из них можно составлять фрагменты программ и подпрограммы. В основе такого программирования лежат взятие значения какой-то переменной, совершеие над ним действия и сохранение нового значения с помощью оператора присваивания. Этот процесс продолжается до тех пор пока не будет получено желаемое окончательное значение.

Функциональная программа состоит из совокупности определений функций. Функции, в свою очередь, представляют собой вызовы других функций. Функции часто вызывают сами себя. Каждый вызов возвращает некоторое значение в вызвавшую его функцию, вычисление которой после этого продолжается, процесс повторяется до тех пор, пока функция не вернет конечный результат пользователю.

Lisp – лепетать

Лисп (*LISP*, от англ. *LISt Processing language* — «обработка списков»; современное написание: *Lisp*) — семейство языков программирования, программы и данные в которых представляются системами линейных списков **символов**. Лисп является вторым в истории (после Фортрана) высокоуровневым языком программирования, который используется по сей день.

Символ - это последовательность, состоящяя из букв, цифр и специальных знаков, которая обозначает какой-нибудь объект, отношение или действие из реального мира:

APPLE RABBIT READ-LINE

Числа - могут быть целыми или с фиксированной точкой (вещественными):

-5.2 15 34.117

Символы и числа представляют собой те простейшие объекты LISPa, из которых строятся остальные структуры. Поэтому их называют атомами (atom). Список - это упорядоченная, ограниченная последовательность, элементами которой являются атомы или другие списки (подсписки). Списки заключаются в круглые скобки, элементы списка разделяются пробелами. Например,

(The quick brown fox)

((height 172) (weight 75) (eyes blue))

((рост 172) (вес 75) (глаза голубые))

(a b (c d) e)

Символы T и NIL обозначают логические значения: T (true) - истина, NIL (false) -ложь. Символом NIL обозначается также и пустой список (). Числа и логические значения T и NIL являются константами, остальные символы - переменными.

Одним из основных отличий языка LISP от традиционных языков программирования является запись в виде списков не только данных, но и программ (или функций). Например, список (+ 2 3)
можно интерпретировать в зависимости от окружения либо как действие (дающее число 5), либо как список, состоящий из трех элементов.

Чтобы предотвратить вычисление значения выражения, нужно перед этим выражением поставить апостроф ' (quote):

'(+ 2 3) или (Quote (+2 3))

ВЫРАЖЕНИЯ LISP (S-ВЫРАЖЕНИЯ)

Общие правила построения s-выражений можно определить следующим образом:
1) атом есть s-выражение;
2) список s-выражений, является s-выражением

В языке LISP как для вызова функции, так и для записи выражений принята единообразная префиксная форма записи, при которой как имя функции или действия, так и сами аргументы записываются внутри скобок:

в математике - f(x), g(x,y), h(x,g(y,z))
x+y, x-y, x*(y+z)

в LISPe - (f x),(g x y),(h x (g x y))
(+ x y),(- x y),(* x (+ y z))

Одним из основных отличий языка LISP от традиционных языков программирования является запись в виде списков не только данных, но и программ (или функций). Например, список (+ 2 3) можно интерпретировать в зависимости от окружения либо как действие (дающее число 5), либо как список, состоящий из трех элементов.

Чтобы предотвратить вычисление значения выражения, нужно перед этим выражением поставить апостроф ' (quote):

'(+ 2 3) или (Quote (+2 3))

ФУНКЦИЯ

Функция - это процедура, в результате вычисления которой вырабатывается некоторое значение (s-выражение). Программист может вычислять функцию лишь ради ее побочного эффекта (присвоения значения, операция ввода-вывода информации и т.д.), но и в этом случае она должна выдавать некоторые значения, т.е. изменение значения аргумента внутри функции не влечет за собой изменения соответствующей переменной вне функции, другими словами в функцию передается не переменная, а только ее значение.

При обращении к функции указывается ее имя и аргументы (фактические параметры - ACTUAL ARGUMENTS). Аргументами функции в общем случае могут быть s-выражения. Связь с формальными параметрами (FORMAL ARGUMENTS) осуществляется по значению.

В языке LISP предусмотрено много встроенных функций (подобно базовым), которые будут расмотрены позже. Но сколько бы встроенных функций не было в языке всегда возникает необходимость в новых функциях, определяемых пользователем.

Лямбда-выражение - функции без имени, которая пропадает тотчас после вычисления значения.

Проблема разрешима путем именования лямбда-выражений и использования в вызове лишь имени.

Дать имя и определить новую функцию можно с помощью функции DEFUN (define function).

$ (DEFUN <имя функции> ([<список формальных параметров>])
    <тело функции>)

Тело функции состоит из одного или нескольких s-выражений.

Внимание !!! Описание функции должно всегда предшествовать обращению к ней.

Примеры:

1) определить функции, возвращающие второй (SECOND) и третий (THIRD) элементы заданного списка:

```
$ (DEFUN second (LIST)          ; определение функций
      (car (cdr LIST)))
$ (DEFUN third (LIST)
      (car (cdr (cdr LIST))))

$ (second '(cat dog cow horse))  ; вызов функций
dog
$ (third '(cat dog cow horse))
cow
```

1\ Здесь и дальше квадратные скобки ([]) обозначают необязательность параметра

2) Построить функцию, возвращающую T, если аргумент пустой список и NIL - в противном случае.

ПРОСТЫЕ И УСЛОВНЫЕ ФОРМЫ

Под формой (form) понимается такое s-выражение, значение которого может быть найдено интерпретатором. Тело функции состоит из форм или предложений (clause). Предложения бывают двух типов: простыми и условными.

Простое предложение - это атом или список, если его первый элемент - атом: NIL,

    (cdr '(DOG CAT COW))

Условное предложение - это список, если его первый элемент не является атомом.

    ((cdr LIST) (car (cdr LIST)))

Первый элемент условного предложения используется как предикат.

Если значение предиката NIL, то значение предложения тоже NIL и для вычисления тела функции интерпретатор переходит к анализу следующего предложения (если оно есть). Если выражения кончились - функция возвращает NIL.

Если значение предиката не равняется NIL, значение функции определяется используя хвост условного предложения.
Примеры:
1) Предикат LISTP возвращает NIL, если аргумент - атом и возвращает T - в противном случае.

```
$ (DEFUN listp (OBJ)
        ((NULL OBJ))
        ((ATOM OBJ) NIL)
        T)

$ (listp '(a b c))
T


$ (listp 'a)
NIL
$ (listp ())
T
```

1\ Предикатом называется любая функция с областью значений {T, NIL}

Принцип работы данной LISP-программы заключается в следующем: если (NULL OBJ) - истина, то LISTP возвращает T и выполнение программы прекращается. Если (NULL OBJ) - NIL,то происходит переход к следующему предложению, где проверяется, является ли аргумент функции атомом.

2) Функция TYPE определяет тип s-выражения: пустой список, атом, список.

```
$ (DEFUN type (OBJ)
        ((NULL OBJ) 'Пустой-список)
        ((ATOM OBJ) 'Атом)
        'Список)

$ (type '(a b c))
Список
$ (type (atom '(a b c)))
Пустой-список
$ (type (atom 'atom))
Атом
```

РЕКУРСИВНОЕ ОПРЕДЕЛЕНИЕ ФУНКЦИИ.

Функция является рекурсивной, если в ее определении содержится вызов самой этой функции.

Примеры:

1) Определить, является ли NAME элементом в списке LIST.

```
$ (DEFUN member (NAME LIST)
        ((NULL LIST) NIL)
        ((EQL NAME (CAR LIST)))
        (member NAME (CDR LIST)))
```

Отсутствие проверки, ошибочное условие или неверный их порядок могут привести к бесконечной рекурсии. Это произошло бы, например, если б в предикате MEMBER значение аргумента LIST не укорачивалось бы на каждом шагу рекурсии формой (CDR LIST). То же самое случилось бы, если рекурсивная ветвь находилась в словном предложении перед условием окончания или при отсутствии условия окончания.