Функциональное программирование ориентировано на символьную обработку данных

Символ == Идентификатор

**Элементы языка/ Базовые элементы языка:**

Атомы:

1. символы(идентификаторы) - синтаксически - набор литер(букв латинского алфавита и цифр), начинающихся с буквы;

2. специальные символы - T, Nil - используются для обозначения «логических констант»;

3. само определимые атомы - натуральные числа, дробные числа, вещественные числа, строки последовательность символов, заключенных вдвойные апострофы.

Более сложные данные в Lisp выстраиваются с помощью бинарных узлов, содержащих пару указателей.

Точечная пара – структура данных, состоящая из двух символьных выражений, разделенных точкой.

Список – это структура данных(**Определение списка**). Может быть пустой и непустой. Если непустой, то состоит из двух элементов: первый - любой формы, а второй - список. Список - это частный случай S-выражения. В памяти список представляется бинарным узлом, состоящим из двух указателей: car и cdr

Списки представлены с помощью списковых ячеек. В списковой ячейке хранится два указателя: на голову и на хвост. Способ интерпретации определяется положением выражения и алгоритмом функционирования ЛИСП машины

**Представление элементов в памяти**

Атом в памяти представляется 5тью указателями.

Точечная пара - бинарным узлом.

Список - бинарный узел(тоже самое списковой ячейкой)

**Символьное выражение** это атом или точечная пара

**Что такое бинарный узел?** Бинарный узел – это структура данных. Каждый бинарный узел соответствует минимальному блоку памяти, выделяемому системой при организации и обработке структур данных.

**Базис ЛИСП –** это минимально необходимый набор конструкций с помощью которгот можно запрограммировать.

Атомы, структуры, базовые функции, базовые функционалы

CAR и CDR являются базовыми функциями доступа к данным. CAR принимает точечную пару или пустой список в качестве аргумента и возвращает  
первый элемент или nil, соответственно. CDR принимает точечную пару или пустой список и возвращает список состоящий из всех элементов, кроме первого.  
Если в списке меньше 2 элементов, то возращается nil.

LIST и CONS являются функциями создания списков (cons - базовая, list -  
нет). Функция «cons» создает списочную ячейку и устанавливает два указателя  
на аргументы. Функция list принимает переменное число аргументов и возвращает список, элементы которого - переданные в функцию аргументы.

**Как установить символьному атому значение?**

Функция **SET** класса **SUBR** требует ровно два аргумента. Значением первого аргумента должен быть атом, а значением второго - произвольное S-выражение. Это значение функция одновременно возвращает в качестве результата. **SETQ –** не вычисляет значение первого элемента. **CSETQ –** устанавливает значение которое нельзя изменить.

**Как воспринимается апостроф?**

Апостроф используется для блокировки вычисления значения. Если аргументом является самоопределяющий атом, то квонтировать не требуется.

Замена для апострофа функция QUOTE, которая принимает ровно один аргумент и возвращает выражение совпадающее с аргументом

**Как определить функцию?**

Собственную функцию можно определить через спец функцию DEFUN, которая принимает три аргумента(первый должен быть атомом(название), второй аргумент список атомов(аргументы), третий аргумент произвольной формы(тело функции))

|  |
| --- |
| Но функция не обязательно должна иметь имя, для того, чтобы определить функцию, не имеющую имени, необходимо воспользоваться лямбда-выражением. Лямбда-выражение – это список, содержащий символ lambda и следующие за ним список аргументов и тело, состоящее из нуля или более выражений. |
|  |

|  |
| --- |
|  |
|  |

(lambda (arg1 arg2 .. argN) func\\_body)

**Классификация функций**

Базовые – представляются языком, минимально необходимые для работы языка

Селекторы(car, cdr)

Конструкторы(cons)

Предикаты – логические функции(null atom numberp)

Сравнения (eq, eql, equal, =)

Чистые(математические) – фиксированное кол-во аргументов и есть результат

Рекурсивные

Специальные-формы – функции принимающие переменное кол-во аргументов и обрабатываются они тоже по разному. Пример функция +

* Функционалы – используют другие функции в качестве аргументов или вырабатывают в качестве результатов(funcall apply)

Псевдофункции – функции, которые создают эффект на внешних устройствах(на экране)

Функции с вариантными значениями из которых выбирается только 1 значение

Городняя Л В основы фп

Погрем коммон лисп - справочни

Функция **EVAL** вычисляет значение своего единственного аргумента и возвращает его в качестве результата. Ядро Лиспа, в сущности, работает следующим образом:   
  
1. Ожидает ввода S-выражения;   
  
2. Передает введенное S-выражение функции **EVAL**;   
  
3. Выводит полученный результат;   
  
4. Переходит к п.1   
  
**EVAL - это универсальная функция Лиспа, которая может вычислить любое правильно составленное s-выражение.**

Функция **EVAL** в известном смысле противоположна по своему действию функции **QUOTE** - для любого S-выражения **x** результат вызова **(EVAL (QUOTE x))** совпадает с **x**.

Вопросы????7

Как воспринимаются элементы списка? Первый элемент списка это название функции, остальные элементы аргументы(1 2 3/ 1 – без квонтирования будет считан как функция, а остальное как аргументы).

Что хранит бинарный узел? **Бинарный узел – это списковая ячейка содержащая пару указателей на голову и хвост.**

* Что такое функционалы? Функционалы – используют другие функции в качестве аргументов или вырабатывают в качестве результатов(funcall apply)

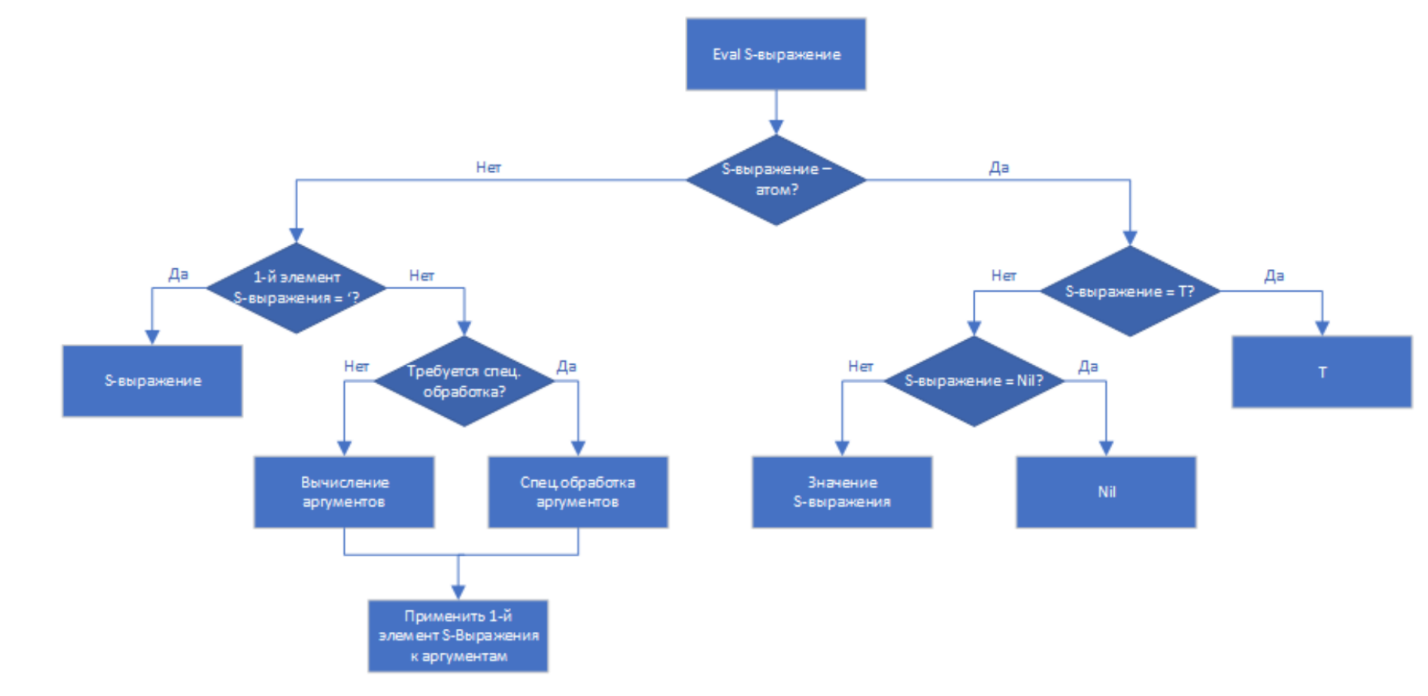
Функция **FUNCALL** принимает произвольное количество аргументов. Эта функция применяет свой **первый** (функциональный) аргумент к оставшемуся списку аргументов

**(funcall '\* 1 2 3 4 5 6 7 8 9 10)**

Функция **APPLY** очень схожа функция **FUNCALL.** Отличие заключается в том, что у функции **APPLY** ровно два аргумента: первый - функциональный, а второй является списком произвольной длины. Вызов **APPLY** заключается в том, что вычисляется функция, заданная первым аргументом, со списком параметров, заданным вторым аргументом **APPLY**.

**(apply ‘\* '(1 2 3 4 5 6 7 8 9 10))**

|  |
| --- |
| Функционалы используются для вычисления одной функции с разными входными значениями. Например, для применения какой-то операции ко всему списку. Некоторые функционалы используют повторные вычисления, что позволяет организовывать накие вычисления на этих функционалах (например, mapcar) |
|  |

* Как система трактует элементы eval/Порядок реализации программы ЭВАЛ – это интерпретор lisp. 

**List(‘a ‘b) – (a b)**

**List(‘a b) ошибка**

**List (a ‘b) ошибка**

**Cons(‘a ‘b) – a.b**

**cons(‘a b) ошибка**

**cons(a ‘b) ошибка**

|  |
| --- |
| **Синтаксическая форма и хранение программы в памяти** |
|  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Синтаксическая программа на LISP выполняется в виде S-выражения (обычно - списка). Программы в памяти хранятся точно так же, как и данные. Чтобы в LISP выдавать информацию то за данные, то за программу, надо чтобы все помещалось в кучу, при этом ведется работа с указателями.  **Работа функций AND OR IF COND**   |  | | --- | | Функция cond принимает произвольное количество аргументов. Каждый аргумент функции должен быть списком ровно из двух элементов. Первый из этих элементов условно называется условием, а второй - результатом. Принцип работы заключается в выполнении каждого условия из каждого списка условий до тех пор пока не встретися условие равное T. Если такого выражения нет, то COND вернет NIL. | |  |  |  | | --- | | **Функция if:** | |  |  |  | | --- | | Оператор if обозначает то же, что и конструкция cond. Сначала выполняется условие. Если результат не равен nil, тогда выбирается форма then. Иначе выбирается форма else. Выбранная ранее форма выполняется, и if возвращает то, что вернула это форма. if более читабельный. Форма else может быть опущена. В таком случае, если значение условия является nil, тогда ничего не будет выполнено и возвращаемое значение формы if будет nil. | |  |  |  | | --- | | **Функция and:** | |  |  |  | | --- | | Логическая функция AND берет один или несколько аргументов. Она выполняет эти аргументы слева направо. Если она встречает аргумент, значение которого NIL, она возвращает NIL, не продолжая вычисления остальных. Если NIL аргументов не встретилось, то возвращается значение последнего аргумента. | |  |  |  | | --- | | **Функция or:** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Логическая функция OR берет один или несколько аргументов. Она выполняет эти аргументы слева направо и возвращает значение первого аргумента, который не NIL(т е возвращает ПЕРВЫЙ ТРУ).Если все аргументы OR имеют значение NIL, то OR возвращает NIL.   |  | | --- | | **Чем принципиально отличаются функции CONS, LIST, APPEND?\\** | |  |  |  | | --- | | **CONS** является базовой функцией Lisp, LIST и APPEND реализованы через CONS. CONS является чистой математической функцией, принимающей 2 параметра. Она создает списочную ячейку, CAR-указатель которой указывает на 1-й аргумент, а CDR-указатель - на 2-й аргумент. | |  |  |  | | --- | |  | |  |  |  | | --- | | **LIST** является формой, так как принимает произвольное число аргументов, возвращая список, состоящий из переданных ему аргументов. | |  |  |  | | --- | |  | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **APPEND** является формой, так как принимает произвольное число аргументов, которые должны быть списками, кроме последнего. Она возвращает точечную пару, CAR-указатель которой указывает на конкатенацию всех переданных ей аргументов, кроме последнего, а CDR-указатель - на последний аргумент. При этом структура исходных аргументов не разрушается, для этого происходит копирование всех аргументов, кроме последнего.  ПРИМЕР   |  | | --- | | (append ‘(a b) ’(c d))\\ | |  |   Результат: (A B C D)  ИТОГО   * **cons** всегда берет два аргумента и помещает первый в начало второго. * **list** берет один или больше аргументов и образует список, помещая аргументы в скобки. * **append** образует новый список, убирая скобки вокруг аргументов и помещая их в один список.   **Структурно разрушающие и не разрушающие структуру списка функции**   |  | | --- | | Структурно разрушающие функции - это те функции, которых после обработки теряются данные. Эти функции при работе работают с самимми данными, то есть не создают копии чтобы с ними работать  SETF, last, nreverse, append - разрушающий | |  |  |  | | --- | | **Структурно не разрущающие** - это функции в которых после обработки остаются данные. Эти функции при работе при работе создают копию и работают с ней.  Неразрушающая - reverse | |  |   Например функцци с приставкой n(r??)- разрушающие функции, а без - не разрушающие. | |  |  |  | | --- | |  | |  | | |  | |
|  |

**Функция REVERSE изменяет порядок элементов в аргументе.**

 Аргументом **reverse** должен быть список.

 **reverse** не меняет порядок в списках более нижнего уровня.

**\* ( reverse ' ( ( a b c ) e ) )  
( e ( a b c ) )**

**Функция LAST удаляет из списка все элементы кроме последнего.**

**\* ( last ' ( a b c ) )  
( c )**

NCONC В качестве аргументов [**nconc**](http://filonenko-mikhail.github.io/cltl2-doc/ru/symbols.html#x200-396666r666) принимает списки. Функция соединяет списки и возвращает результат. При этом аргументы изменяются, а не копируются. (В сравнении с [**append**](http://filonenko-mikhail.github.io/cltl2-doc/ru/symbols.html#x200-396099r99), которая копирует аргументы, а не разрушает их.) Например:

(setq x ’(a b c))   
(setq y ’(d e f))  
(nconc x y) ⇒ (a b c d e f)  
x ⇒ (a b c d e f)

**PRINC возвращает результат без кавычек.** PRINC будет печатать их таким путем, чтобы они могли считываться функциями типа READ-LINE.

TERPRI Эта функция печатает новую строку на экране. Она также возвращает новую строку. TERPRI не используется для ввода/вывода файлов. Для записи новой строки в файл используйте PRINT или PRINC.

MAPCON применяет функцию указанную ко всем элементам списка, использую cdr пока они не закончатся

***(numberp <элемент>)***

Эта функция возвращает Т, если <элемент> - целое или действительное число, иначе nil.

Например, дано:

mapcar работает с последовательными элементами списков. функция применяется к первому элементу каждого списка, затем ко второму элементу каждого списка и так далее. Итерация завершается, когда заканчивается самый короткий список, а лишние элементы в других списках игнорируются. Значение, возвращаемое mapcar, представляет собой список результатов последовательных вызовов функции.

maplist похож на mapcar, за исключением того, что функция применяется к последовательным подспискам списков. Функция сначала применяется к самим спискам, затем к cdr каждого списка, а затем к cdr cdr каждого списка и так далее.

**MAPCAR - выполняет действия, указанные в функции FUNC над car элементами списков**

**MAPLIST - Функция сначала применяется к самим спискам, затем к cdr каждого списка, а затем к cdr cdr каждого списка и так далее.**

**(MAPCON FUNC LIST1 ... LISTN)**выполняет действия, указанные в функции **FUNC**, над **CAR**-элементами списков **LIST1,...,LISTN**, затем - над **CDR**-элементами каждого списка, и т.д. до тех пор, пока каждый из списков не будет исчерпан. Функция **MAPCAN *"связывает"*** свои результаты, используя функцию **NCONC**. Например:

**$ (MAPCON 'REVERSE '(A B C D))**

**(D C B A D C B D C D)**

**(MAPCAN FUNC LIST1 ... LISTN)**

выполняет действия, указанные в функции **FUNC**, над **CAR**-элементами списков **LIST1,...,LISTN**, затем - над **CADR**-элементами каждого списка, и т.д. до тех пор, пока каждый из списков не будет исчерпан. Функция **MAPCAN *"связывает"*** свои результаты, используя функцию **NCONC**.

    Функционал **MAPCAN** удобен для выборки нежелательных элементов из списка. В приведенном примере показано, как можно удалить из списка отрицательные числа:

**$ (MAPCAN '(LAMBDA (N) ( (MINUSP N)) (LIST N))**

**'(3 -7 4 0 -5 1))**

**(3 4 0 1)**

*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, используя функцию* ***NCONC****.*

*Функции* ***MAPCAN*** *и* ***MAPCON*** *можно определить с помощью функций* ***MAPCAR*** *и* ***MAPLIST*** *следующим образом:*

**(MAPCAN FN x1 x2 ...xN) <--->**

**(APPLY (NCONC (MAPCAR FN x1 x2...xN)))**

**(MAPCON FN x1 x2 ...xN) <--->**

**(APPLY (NCONC (MAPLIST FN x1 x2...xN)))**

**РЕКУРСИЯ**

|  |
| --- |
| **Рекурсия** — это ссылка на определяемый объект во время его определения. |
|  |

|  |
| --- |
| Способы организации рекурсивных функций: |
|  |

|  |
| --- |
|  |

|  |
| --- |
| Хвостовая рекурсия |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Результат формируется не на выходе из рекурсии, а на входе в рекурсию, все действия выполняются до ухода на следующий шаг рекурсии. |
|  |

|  |
| --- |
|  |

|  |
| --- |
| Рекурсия по нескольким параметрам |
|  |

|  |
| --- |
|  |

|  |
| --- |
| Дополняемая рекурсия |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| При обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова, а вне его. |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Множественная рекурсия |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| На одной ветке происходит сразу несколько рекурсивных вызовов. Количество условий выхода также может зависеть от задачи. |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Способы повышения эффективности реализации рекурсии |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Рекомендуется организовывать и отлаживать реализацию отдельных подзадач исходной задачи, обращая внимание на эффективность реализации и качество работы, а потом, при необходимости, встраивать эти функции в более крупные, возможно в виде лямбда-выражений. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Для повышения эффективности рекурсии необходимо **правильно организовавывать условия выхода из нее.** Основное правило: при построении условного выражения **первое условие - это всегда выход из рекурсии**, но если условий выхода несколько, то надо думать о порядке их следования. **Некачественный выход из рекурсии может привести к переполнению памяти** из-за "лишних" рекурсивных вызовов. Кроме того возможна потеря аргумента - кажется что функция возвращает результат и он используется, но на деле результат теряется и ответ неверен. |
|  |

|  |
| --- |
|  |
|  |

**В целях повышения эффективности рекурсивных функций рекомендуется формировать результат не на выходе из рекурсии, а на входе в рекурсию, все действия выполняя до ухода на следующий шаг рекурсии.**