Правила оформления решений на языке Lisp

Содержание

1 Общие правила оформления

§ 1.1 Внимательно читайте задание. Посылаемое на проверку решение должно удовлетворять всем условиям, представленным в задании. В том числе это относится к имени основной функции, последовательности и типу ее аргументов и типу ее результата (если таковые указаны в задании).

§ 1.2 Каждая строка программы не должна превышать по длине 80 символов (включая отступы в начале строки).

§ 1.3 Код программы не должен содержать символов табуляции. Для отступов необходимо использовать пробелы. Текстовый редактор необходимо настроить, чтобы при нажатии клавиши табуляции вместо символа табуляции вставлялось 2 пробела.

§ 1.4 Программа должна компилироваться. Любая программа, посылаемая на проверку, должна проходить успешную компиляцию *без ошибок и предупреждений* (errors или warnings). Без исключений. После внесения изменений в код программы, даже самых незначительных, убедитесь в том, что программа проходит успешную компиляцию, прежде чем посылать ее на проверку.

§ 1.5 В решениях допускаются только однострочные комментарии. Допускаются только комментарии начинающиеся со знака `;` . При этом следует различать категории таких комментариев:

- комментарий начинающийся с одного символа `;` — комментарий к текущей строке кода. Одиночный символ `;` не должен начинать строку;
- комментарий начинающийся с двух символов `;;` — комментарий описывающий последующий блок программы. При этом символы `;;` начинают строку с тем же отступом, что и описываемый блок;
- комментарий начинающийся с трех символов `;;;` — глобальный комментарий относящийся к последующей части программного кода. При этом символы `;;;` начинают строку с первой позиции.

```
;;; the next 20 functions do various sorts of frobbing
(defun frob1 (num)
  ;; return double frob of num
  (let ((tmp (random num)))      ; breaks if 0, fix!
    (double-frob tmp num :with-good-luck t)))
```

2 Переносы, отступы, пробелы и выравнивание

§ 2.1 Используйте пустые строки разумно. Пустые строки используются, в основном, на верхнем уровне, для отделения одного определения (группы определений) от другого.

Пустые строки могут повысить читабельность программы, но есть и негативный эффект: текстовый редактор отображает ограниченное количество строк. Если необходимо прочитать и понять за раз функцию, занимающую большое количество строк, лишние пустые строки могут помешать в этом.

Не должно быть пустых строк в выражении (включая выражение — тело функции).

§ 2.2 Закрывающие скобки не должны размещаться на отдельных строках. Закрывающая скобка не должна начинать строку, открывающая скобка не должна заканчивать строку.

```
(define (factorial x)             ; ОЧЕНЬ ПЛОХО
  (if (< x 2)
      1
      (* x (factorial (- x 1
                         )
           )
      )
  )
)
```

```
(
  define (factorial x) (          ; ОЧЕНЬ ПЛОХО
    if (< x 2)
      1
      (* x (factorial (- x 1)))
    ))
```

```
(define (factorial x)             ; ХОРОШО
  (if (< x 2)
      1
      (* x (factorial (- x 1)))))
```

§ 2.3 Если все параметры в вызове функции являются атомарными или простыми выражениями, то такой вызов следует разместить в одной строке.

```
(+ 3              ; ПЛОХО
   y
   x
   (+ 2 x))
```

```
(+ 3 y x (+ 2 x))  ; ХОРОШО
```

§ 2.4 Если один или несколько параметров в вызове функций являются сложными выражениями, то каждый из параметров (за исключением, возможно, первого) следует разместить на новой строке.

```
(+ (* (+ (/ 4 3) (/ 5 2)) 3 (+ (/ 4 3) (/ 5 2))) ; ОЧЕНЬ ПЛОХО
   (* (- (/ 6 11) 7) (+ (/ 11 3) 5)))
```

```
(+ (* (+ (/ 4 3)                                ; ХОРОШО
         (/ 5 2))
      3
      (+ (/ 4 3)
         (/ 5 2)))
   (* (- (/ 6 11) 7)
      (+ (/ 11 3) 5)))
```

В большинстве случаев стоит придерживаться правил: после двух подряд идущих закрывающих скобок в строке не должно быть никаких символов, кроме закрывающих скобок; если в строке уже встречалась закрывающая скобка, то после неё нежелательны открывающие.

§ 2.5 Описание первого аргумента должно оставаться на первой строке в вызовах `lambda`, `let`, `labels`. В конструкциях `defun`, `defmacro` описание первого и второго параметров должно начинаться в первой строке.

```
(defun                          ; ПЛОХО
  fact
  (n)
  (cond
    ((< n 2) 1)
    (T (* n (fact (- n 1))))))
```

```
(defun fact (n)                 ; ХОРОШО
  (cond ((< n 2) 1)
        (T (* n (fact (- n 1))))))
```

§ 2.6 Описание первого аргумента функции остается на первой строке или начинается с новой строки с обычным отступом относительно предыдущей строки. Остальные аргументы выравниваются относительно первого.

§ 2.7 Если описание одного из аргументов функции начинается с новой строки, то с новой строки обязан начинаться каждый аргумент (кроме первого). Возможно исключение: если несколько первых параметров функции являются атомарными, то все их можно разместить на первой строке.

§ 2.8 Делайте отступы постоянной длины. Обычный отступ — 2 пробела. Обычный отступ делается в теле конструкций `lambda`, `defun`, `defmacro`, `cond`, `let`, `labels`, `progn` и др., а также в вызовах функций, когда каждый параметр функции, включая первый, пишется с новой строки.

```
(compute-closure
  this-function
  (list other-function my-method)
  56)
```

```
(defun frob1 (num)
  (let ((tmp (random num)))
    (double-frob tmp num :with-good-luck t)))
```

§ 2.9 Выравнивание в вызове функций. Выравнивайте параметры функции по левому краю. Если первый параметр функции указывается на той же строке, что и имя функции, то параметры, указываемые на следующих строках выравниваются относительно него.

```
(compute-closure this-function                  ; ПЛОХО
  (list other-function my-method)
  56)
```

```
(+ (sqrt -1)                                    ; ПЛОХО
  (* x y)
  (+ p q))
```

```
(compute-closure this-function                  ; ХОРОШО
                 (list other-function my-method)
                 56)
```

```
(+ (sqrt -1)                                    ; ХОРОШО
   (* x y)
   (+ p q))
```

§ 2.10 Выравнивайте элементы списка по левому краю. Если необходимо привести в программе длинный громоздкий список в несколько строк, то вторая и последующие строки выравниваются по левому краю первого элемента списка. Такое же выравнивание должно осуществляться для первого аргумента конструкций `let`, `labels` и подобных.

```
(setq colour-names
  '(blue
    cerulean
    green
    magenta
    purple
    red
    scarlet
    turquoise))
```

```
(let ((d 1)
      (e (if (zerop b) t nil)))
  (check-compatibility d c)
  (foo-aux a b c d e))
```

§ 2.11 Правильно используйте пробелы.

1. Ставьте пробел перед открывающей скобкой если перед ней в строке присутствует что-то, отличное от открывающей скобки.
2. Ставьте пробел после закрывающей скобкой, если после нее в строке присутствует что-то, отличное от закрывающей скобки.
3. Не ставьте пробелов после открывающей скобки и перед закрывающей скобкой.
4. Не ставьте в выражении два или несколько подряд идущих пробелов (за исключением пробелов перед комментариями).

```
(foo (bar baz) quux daasum)   ; ХОРОШО

(foo( bar baz )quux  daasum)  ; 5 раз ПЛОХО
```

3 Имена и объявления

§ 3.1 Заглавные буквы в идентификаторах не допускаются.

§ 3.2 В именах, состоящих из нескольких слов, слова разделяются дефисом.

§ 3.3 Давайте имена в соответствии с их смыслом. Имя переменной или функции должно говорить о том что переменная обозначает или что функция вычисляет. В небольшой области видимости допустимы короткие имена.

§ 3.4 Допускаются следующие общепринятые обозначения (обычно, для аргументов функции): `f` для функций, `s` для строковой переменной, `n` для целочисленной переменной, `x` и `y` для переменной любого типа, `a` или `b` для переменной любого типа, `xs` или `ys` для списков, `xss` для списка списков, и т. д.

§ 3.5 Имена предикатов должны заканчиваться на символ `p`.

§ 3.6 Именуйте функцию по значению, которое она возвращает.

§ 3.7 Не перегружайте базовые функции. Не пишите свои варианты базовых функций (в особенности `filter`, `map` и т. п.).