**Стиль оформления исходных тестов программ. Отступы, "лесенка", пробелы. Преимущества и недостатки использования утилит для автоматического форматирования исходного текста программы.**

Одним из важнейших факторов, влияющих на способность программы к развитию, является ее понимаемость. Одним из существенных факторов понимаемости программы, в свою очередь, является информативность исходного текста. Если исходный текст не является хорошо читаемым, то есть написан без соблюдения определенного стиля и системы и представляет собой "мешанину" операторов и знаков препинания, то вносить изменения в него очень сложно даже автору. Рассмотрим ряд требований и рекомендаций, позволяющих выработать хороший стиль оформления программ, повышающий ее информативность.

* ***Лесенка в 4 пробела***

Обязательна «лесенка» с отступом в 4 пробела. При этом запрещается в редакторе изменять размер *отображаемой* табуляции, например, выставлять отображение табуляции в 4 пробела. Код, созданный Вами при таких настройках, будет некорректно отображаться в других редакторах с другими настройками.

* ***Пробелы после запятых***

Ставятся после запятых и точек с запятой (если, конечно, они не расположены в конце строки). Перед запятой и точкой с запятой пробелы не ставятся:

|  |
| --- |
| a = (1, 2, 3);  *for (int i = 0; i < count; i++ ) { };* |

## *Пробелы вокруг знаков операций*

|  |
| --- |
| a = b \* c + d \* e;  a = b \* c + d \* e; |

Любые операторы / знаки операций ( например "=", "==", "=>", "<", ">", "&&", "||" и т.п.) отделяются пробелами с обеих сторон:

## *Пробелы вокруг сложных индексных выражений*

В случае, если Вы обращаетесь к элементу массива по индексу и индексное выражение достаточно сложное, отделяйте его пробелами для улучшения удобочитаемости. Если выражение простое — пробелы не обязательны.

|  |
| --- |
| a[1];  a[ 1 + 2 + 2 + 4 ]; |

## *Пробел перед открывающей фигурной скобкой*

|  |
| --- |
| if (condition) { statement; } |

Перед открывающей фигурной скобкой в блочных конструкциях ставится пробел:

## *Пробелы после знака комментария*

После символа начала комментария («//») перед текстом самого комментария ставится пробел.

|  |
| --- |
| // INIT |

## *Максимальная длина строк*

Строки желательно не оставлять слишком длинными; условное ограничение — 80-100-120 символов в строке. При необходимости строка разбивается на несколько.

|  |
| --- |
| very\_long\_statement  if condition;  if (  very\_long\_condition\_1  && very\_long\_condition\_2  ) {  statement;  }  if (  ..  &&  ...  ||  ...  ) {  ...  } |

## *Открывающая фигурная скобка на той же строке, что и ключевое слово*

Старайтесь придерживаться компактного стиля оформления циклов и блоков ветвления: открывающая фигурная скобка находится на той же строке, что и ключевое слово "for", "if", "else", "while" и т.п.  
Закрывающая фигурная скобка блока, состоящего из нескольких строк, должна находиться на одной вертикали с ключевым словом начинающим конструкцию.

|  |
| --- |
| if (condition) {  statement1;  }  else {  statement2;  }  for (int i = 0; i < $count; i++ ) {  statement;  } |

## *Допускается компактное оформление блоков из одного оператора*

Однострочные блоки, состоящие из единственного оператора, могут быть помещены в одну строку вместе с открывающими и закрывающими скобками:

|  |
| --- |
| for (int i = 0; i < count; i++ ) { summ += i; }; |

## *Разбивайте код на абзацы, при необходимости снабжённые комментариями*

Код внутри функций должен быть разделён на смысловые блоки, выполняющие определённую узкую задачу. Смысловые блоки отделяются друг от друга пустыми строками. Для дальнейшего улучшения читаемости кода, добавляйте вначале каждого абзаца однострочный комментарий, объясняющий, что делает эта последовательность операторов.

|  |
| --- |
| addarray\_internal {  string (string var\_name, string needs\_quotemeta) ;  // Запомнить оригинал...  raw = var\_name;  // Добавить экранирование спецсимволов, если необходимо...  string quotemeta = needs\_quotemeta ? q{map {quotemeta $\_} } ;  # Перевести элементы переменной в строку, соединяя их с помощью "|"...  my $perl5pat = qq{(??{join q{|}, $quotemeta \@{$var\_name}})};  # Добавить отладочный код, если необходимо...  my $type = $quotemeta ? 'literal' : 'pattern';  debug\_now("Adding $var\_name (as $type)");  add\_debug\_mesg("Trying $var\_name (as $type)");  return $perl5pat;  } |

## *Не отделяйте имена переменных и функций от следующей за ними открывающей скобки*

Важно ставить открывающую скобку слитно с именем функции или переменной. В противном случае можно визуально спутать функцию с ключевым словом, а начало выражения для элемента массива или хэша со скаляром.

|  |
| --- |
| # ХОРОШО  next CANDIDATE if open\_region($i);  $candidates[$i] = $incumbent{ $candidates[$i]{region} }; |

|  |
| --- |
| # ПЛОХО!  next CANDIDATE if open\_region ($i);  $candidates[$i] = $incumbent { $candidates[$i]{region} }; |

## *Осмысленные названия идентификаторов*

Выбирайте осмысленные названия для идентификаторов (переменных, констант, функций). Исключение составляют итераторы циклов, где допускаются короткие идентификаторы: ib, n и т.п

* ***Одна функция выполняет одну задачу***

Если функция выполняет несколько разных, слабо связанных друг с другом задач, подумайте о том, чтобы разбить эту функцию на несколько. И наименования функций по возможности должны представлять из себя глагол, например get\_domain\_name, или chash\_my\_program.

## *Отступы и комментарии для функций*

Функции отделены друг от друга минимум одной пустой строкой. Для каждой функции необходимо краткое однострочное описание того, что она делает:

|  |
| --- |
| # Получить имя домена по его id  sub get\_domain\_name {  ...  } |

## *Наименование моделей в стиле MyModuleName*

Модули следует называть в стиле MyModuleName, т.е. слова склеиваются между собой, каждое слово записывается с заглавной буквы. Последняя строка кода модуля обязательно заканчивается переносом строки.

## *Комментарий вначале модуля*

Вначале модуля желателен комментарий, описывающий назначение модуля.

## *Сокращённая форма записи, основанная на ||*

Приветствуется использование сокращённой формы записи, основанной на применении логического оператора "или".

|  |
| --- |
| $a = $b || $c  лучше, чем  $a = $b ? $b : $c  и тем более лучше, чем  if ($b) {  $a = $b;  }  else {  $a = $c;  }  Также  $a ||= $b  лучше, чем  $a = $b unless $a; |

## *Используйте and и or вместо условных конструкций if*

В ряде случаях (когда при каком либо условии нужно выполнить всего один оператор) удобнее выполнять условный код с помощью and или or, чем громоздить блоки if:

|  |
| --- |
| imp back or die "can't imp back!";  $username = get\_user\_name() and print "Username: $username\n"; |

* ***Минимизация использования циклов for и обращений к элементам по индексу***

Старайтесь избегать использование классического цикла for с целочисленным индексом, везде, где это возможно. При обработке элементов списка отдавайте предпочтение циклам foreach, а также функциям, обрабатывающим сразу весь список сразу, таким как grep, map, sort, join, splitи т.д.

### *Преимущества и недостатки использования утилит для автоматического форматирования исходного текста программы.*

Преимущества:

* Выполняется форматирование и подсветка кода ;
* В некоторых программах присутствует, как гибкий инструмент, который может быть сильно изменен пользователем, как в плане интерфейса, так и функциональности, чтобы лучше подходить его личным задачам и предпочтениям.

Недостатки:

* Редакторы приходится регулярно запускать;
* Иногда присутствуют проблемы с буфером обмена. Русские символы, скопированные в буфер из редактора, превратятся в «крякозябры»;
* невысокая скорость работы и требовательность к ресурсам системы.