Соглашения о стиле оформления исходных кодов на языке Pascal

Алексей Сальников

2022: v1.0.3

Введение

Существовать без соглашений об оформлении кода просто невозможно. Организации, часто требуют следования своим соглашениям об оформлении кода, так что к этому моменту нужно относиться философски. Если вам не нравятся соглашения принятые в данной организации — смените организацию.

Из доступных соглашений автору бросились в глаза [1, 2, 3]. Данный текст написан главным образом на основе соглашения для GNU Pascal. Оно выбрано как наиболее совместимое со стилем принятым при написании системного кода (например исходных кодов ядра операционной системы) на языке программирования Си.

Существуют так же автоматические средства форматирования кода: Например JEDI Code Format [4] и программа astyle [5]. К сожалению программа astyle работает только с Си-подобными языками программирования. Не стоит ожидать от таких средств что они серьёзно улучшат структуру кода программы. Вероятнее всего всё ограничится отступами и отслеживания уровня вложенности операторов в коде. Для существенной переработки кода используются средства «Рефакторинга кода» [6].

В данном опусе мы остановимся только на простых базовых вещах.

1 Структура каталога с исходными кодами

В каталоге с исходными кодами программы должны присутсвовать следующие файлы и каталоги.

- файл **Readme.txt** или **Readme.md**[1] Данный файл должен содержать описание программы, форматы входных и выходных данных программы, как скомпилировать программу, как запустить программу, в том числе описывать стуктуру подкаталогов внутри каталога с исходным кодом. Желательно, чтобы было указано назначение отдельных файлов с исходниками и не только с исходниками. Например для файлов с тестами необходимо описать назначение каждого теста, что он тестирует.
- Файл **Makefile** или **build.sh** или **build.bat** — файлы с действиями по компиляции вашей программы из набора исходных кодов. Имейте ввиду, если файлы такого типа отсутствуют, то человеку может быть совершенно не очевидно, как по набору исходных файлов сделать программу, особенно если из исходников делается не одна программа, а несколько.
- **Файлы с исходными кодами программы**. Файлы должны быть названы латиницей, имя файла целиком должно быть написано в нижнем регистре (маленькими буквами). Если имя файла состоит из 2-х и более слов, то слова должны быть написаны через подчёркивание целиком маленькими буквами без пробельных символов. Исходный код для Pascal программ должен иметь расширение *.pas*
- каталог **tests** должен содержать тестовые примеры, на которых проверяется работоспособность программы. Внутрь каталога с тестами можно положить свой отдельный Readme, который будет описывать тесты и содержать некоторую инструкцию по проведению тестирования. Что запускать, что передавать в параметрах программе, как проверять правильность вывода и т.п..

Правильно: *line_ algebra_ functions.pas*;
неправильно: *linealgebrafunctions.pas*, *LineAlgebraFunctions.pas*.

Если в имени файла использовать символы отличные от латиницы, то могут возникнуть проблемы с разной кодирокой на разных машинах и в разных операционных системах. Также могут быть проблемы связанные с отсутствием шрифтов на машине для того языка на котором у вас написано имя файла, особенно если эти файлы окажутся совсем не на вашем компьютере и возможно в другой стране. Вы файлы назвали русскими буквами, в какой-то популярной кодировке, а они попали допустим в Японию, где вероятность наличия необходимых шрифтов мала.

[1] Файл в формате Markdown [7].

Если программа является многомодульной, то файлы относящиеся к реализации данного модуля должны быть собраны в одном каталоге, который назван так же, как называется модуль.

2 Пробельные символы

Прежде всего, избегайте ненужных пробелов в конце строк. Так же необходимо заменить все символы табуляции на некоторое фиксированное число пробелов (обычно четыре). Для этого в текстовом редакторе необходимо выставить настройку автоматического заполнения символов табуляции пробелами. Команды которые это делают для редактора vim следующие:

```
" отчсёт размера табуляции
set ts=4

" Установка ширины табуляции.
" в заполняемых символах
set sw=4

" Заполнять табы пробелами
set expandtab
```

Данные настройки можно положить в файл с именем *.vimrc* в домашний каталог пользователя (можно обратиться так: *$HOME/.vimrc*). Шаг табуляции в исходных кодах для сдачи заданий практикума следует сделать равным 4-м символам.

Используйте пустые строки для отделения крупных участков кода друг от друга. Предполагается, что эти участки кода связаны некоторой логической общностью. К таким участкам относятся: длинные комментарии, секции (**type**, **const**, **var**, **export**, **uses**, **label**, и т.п.); прототипы процедур и функций, директивы компилятору (несколько идущих подряд директив целесообразно слепить в один целый блок, а не разделять их пустыми строками). Если комментарий ставится перед описанием чего-либо, то перед комментарием ставится пустая строка. Пример:

```
procedure Long;
    const
      ...

    var
      variables of Long ...

      procedure Sub;
      var
        ...

      begin
        ...
      end;


    {
       multi-line
       comments before code
    }
    begin
      ...
    end;
```

Декларации переменных, типов, и.т.п. должны быть оформлены так, что сперва на отдельной строчке идёт ключевое слово (например var), далее сами декларации с новой строчки с отсупом эквивалентным одной табуляции.

```
var
    items :integer;
    x_coord, y_coord, radius :real;
```

Если определяется много переменных или других объектов и они отделяются друг от друга разделителем (запятая, точка запятая), то за разделителем ставится пробел (можно больше чем один для визуального выравнивания текста, но особо не увлекаться).

Вложенные блоки кода должны отступать от предыдущего уровня вложенности на отступ эквивалентный одной табуляции (напоминаю символы табуляции заполняются пробелами). Ключевые слова **begin** и **end** должны быть на внешнем уровне вложенности. См. пример правильного кода с точки зрения отступов, но не имён переменных:

```
procedure TAskCompNameDialog.FormKeyDown(Sender: TObject; var Key: Word;
                                         Shift: TShiftState);
begin
    if (Key=VK_RETURN) and (ssCtrl in Shift) then
    begin
        Key := 0;
        ButtonPanel1.OKButton.Click;
    end;
end;
```

В том случае, когда список параметров процедуры или функции не помещается на одну строку, необходимомо сделать так, чтобы список параметров продолжается на новой строке. Так, что продолжение оказывается под первым параметром. См. предыдущий пример.

Операнды от оператора должны быть отделены пробелами.
Так правильно:

```
    key := 0;
    b := (Key = VK_RETURN) and (ssCtrl in Shift);
```

Так не правильно:

```
    key:= 0;
    b := (Key=VK_RETURN) and (ssCtrl in Shift);
```

Для циклов и условий не стоит экономить количество строк кода. Оператор или блок операторов составляющий тело цикла или условия, должен начинаться на новой строчке. Так правильно:

```
max := matr[0, 0];
for i := 0 to ROWS do
begin
    for j := 0 to COLUMNS do
    begin
        if max < matr[i, j] then
            max := matr[i, j];
    end;
end;
```

Так не правильно:

```
max := matr[0, 0];
for i := 0 to ROWS do begin
    for j := 0 to COLUMNS do begin
        if max < matr[i, j] then max := matr[i, j];
    end;
end;
```

Содержимое внутри скобочек всегда пишется вплотную к открывающей и закрывающей скобочки. См. предыдущие примеры где встречаются квадратные и круглые скобки.

Метка всегда должна начинаться на предыдущем уровне вложенности по отношению к тому коду, который она метит. Сам код должен начинаться со следующей строчки в соответствии со своим уровнем вложенности. См. пример:

```
    x_coord := -1;
    y_coord := -1;
    for i := 0 to ROWS do
    begin
        for j := 0 to COLUMNS do
        begin
            if  matr[i, j] = find_value then
            begin
                x_coord := i;
                y_coord := j;
```

```
                goto process_result;
            end;
        end;
    end;

process_result:
    if x_coord <> -1 and y_coord <> -1 then
        writeln('Found at (', x_coord, ',', y_coord, ')');
```

3 Ширина текста

Ширина кода должна быть не более 80 символов. В исключительных случаях, когда это СИЛЬНО нарушит читаемость допускается превышение данного ограничения. Однако следует писать длинные выражения в «столбик» примерно так:

```
    if (Control1.Top + Control1.Height div 2 < Control2.Top) and
       (Control1.Top + Control1.Height < Control2.Top + Control2.Height)
    then
        Result := TopDiff
    else
        Result := HorzDiff;
```

На предыдущей строчке оставляем оператор, а операнд переносим на следующую строчку.

В vim есть последовательность команд, которая показывает, что вы вылезли за допустимую ширину:

```
set colorcolumn=80
highlight ColorColumn ctermbg=gray
```

4 Именования переменных, констант и типов

В программистком мире принято использовать 2 различные нотации. Одна нотация называется “Camel case”. Её суть заключается в том, что имя переменной пишется таким образом, что первое слово в имени идёт с маленькой буквы, а остальные слова с большой. Примерно так: *veryImportantVariable*. Вторая нотация называется “Snake case”. Здесь все имена пишутся через подчёркивания и маленькими буквами. Примерно так: *very_important_variable*. В рамках практикума мы будем придерживаться нотации “Snake case”.

1. Все ключевые слова в тексте программы должны быть записаны маленькими буквами.
2. Все функции и процедуры должны быть написаны начиная с БОЛЬШОЙ буквы. Это важно, так как в Pascal не пишутся круглые скобки при вызове функции без параметров. Это приводит к тому, что вызов функции визуально не отличим от просто подстановки переменной. Чтобы избежать путанницы переменные всегда с маленькой буквы, а функции и процедуры с большой (за исключением **read**, **readln** и **write**, **writeln**).
3. Все переменные пишутся с маленькой буквы.
4. Константы записываются ИСКЛЮЧИТЕЛЬНО БОЛЬШИМИ БУКВАМИ.
   То есть так правильно: *MAX_RADIUS*, а вот так не правильно: *MAX_radius*.
5. Переменные должны быть названы разумно, в соответствии с тем, что в них хранится, допускаются локальные однобуквенные переменные, только как счётчики в коротких циклах и внутри коротких фрагментах кода (очень коротких функциях).
6. Из за того, что начинающееся с большой буквы зарезервировано для имён функций, для имён типов предполагается, что будет дописан «хвост» *_t*. Таким образом имя типа должно быть написано маленькими буквами, через подчёркивания, где на конце добавлено *_t*. См. пример:

```
type
    matrix_t = array [1..100, 1..100] of integer;

```

7. Метки задаются в нотации “Snake case” большими буквами.

5 Комментарии к коду

Некоторые замечания по поводу комментариев. Хорошим тоном считается вставка коментария перед объявлением переменной или поля записи (**record**). Коментарий должен описывать что это за переменная или поле record, и зачем они нужны. Далее приведён пример такого описания:

```
const
    MAX_BASE = 36;
    MIN_BASE = 2;

type
    long_num_t =
        record

            {
                Signum of number 1 or -1
            }
            sign    :integer;

            {
                This number actual
                length in digits.
            }
            len     :integer;

            {
                Digits of number in one of available numeral
                system.
            }
            digits :array[1..N] of 1 - MAX_BASE .. MAX_BASE - 1;
        end;
```

Перед описанием функций и процедур также стоит оставлять комментарий описывающий суть функции и её поведение. Важно описать все параметры функции и возвращаемое значение. Если функция обладает побочным эффектом и меняет какие либо глобальные переменные, то это тоже должно быть описано в комментарии. Особенно важно оставлять такие коментарии в интерфейсной части модуля паскаль, так как реализация функции или процедуры может быть недоступна человеку.

В начале файла принято оставлять коментарий описывающий назначение данного файла с исходным кодом, указать частью чего, какого проекта, является данный файл. Принято указывать авторство кода и контактную информацию для обращения к автору/авторам в случае каких либо неполадок или проблем с пониманием.

6 Прочее

Следует избегать использования оператора goto. Оператор goto осмысленно использовать для выхода из вложенного цикла на метку за циклом. Ни в коем случае нельзя использовать goto, который отправляет на метку раньше по коду относительно точки прыжка.

Следует избегать использования оператора with с полями record, в той ситуации, когда это действие производится над продолжительным участком кода.
Хорошо:

```
with student do
begin
    id        := 910678;
    full_name := 'John Smit';
    group     := 'mags_216';
end;
```

Плохо:

```
for i:=1 to 20 do
begin
    Writeln('Please describe student ', i);
    with students[i] do
    begin
        Writeln('Document id');
        Readln(id);
        { Code to check and process id }
```

```
9           Writeln('full name');
10          Readln(full_name);
11          Writeln('Group for classes');
12          Readln(group);
13          {Code to check and process group}
14      end;
15  end;
```

Используйте для выхода из середины функции оператор **exit**. Если целесообразно выйти из середины программы, то можно использовать процедуру **halt**, указав ей в качестве параметра 0 – что означает программа завершается корректно, или число большее 0, но меньшее 255 – что означает, что программа завершается с ошибкой. Параметр процедуры при этом может задавать код ошибки. Этот код ошибки можно получить из комманндной оболочки, в которой программа запускалать на выполнение.

Настоятельно рекомендуется минимизировать число глобальных переменных в программе. Например, все параметры программы считываемые из потока ввода или из аргументов командной троки при запуске целесообразно собрать в одну **record** с многими полями.

Сам код «основной» программы, между главными **begin** и **end.** стоит минимизировать, и перенести как можно больше внутрь процедур и функций.

Вместо заключения

Основная цель соглашения о стиле исходного кода заключается в двух вещах:

1. Повысить читаемость кода. Читаемость повышается как для автора программы, так и для стороннего человека, который вынужден будет с большим трудом понимать мысль автора кода. Помните, что в первую очередь вы пишете программу, не для себя, а для «того парня», которому придётся разбираться в вашем коде. Облегчите ему жизнь, тем более, что «тем парнем» можете оказаться вы сами по истечении некоторого времени.

2. Привести код к единому стандарту. Приводится к стандарту принятому в данной огранизации (сообществе), это очень важно для упрощения процесса коллективной разработки и модификации кода разными людьми.

Соглашение о стиле программного кода, на самом деле не является совсем уж строгим законом. Если результат следования правилам сильно нарушит понимаемость кода, то стоит, именно на этот короткий участок кода, отказаться от следования букве соглашения, оставшись верным его духу. Во всех странных, сложнопонимаемых местах следует оставлять комментарий подробно описывающий происходящее в коде. Если в каком-то месте мы не следуем соглашению о стиле кода, то в этом месте, если не очевидно, стоит написать почему не следуем.

Список литературы

[1] Страндарт соглашения для GNU Pascal: `http://www.gnu-pascal.de/h-gpcs-en.html`.

[2] Стандарт во Free Pascal: `http://wiki.freepascal.org/Coding_style/ru`.

[3] Принятые соглашения в Delphi и Lazarus: `http://edn.embarcadero.com/article/10280`.

[4] JEDI Code Format: `http://jedicodeformat.sourceforge.net`.

[5] Artistic Style: `http://astyle.sourceforge.net`.

[6] Фаулер М., Бек К., Брант Д., Робертс Д., Апдайк У. Рефакторинг: улучшение существующего кода (Refactoring: Improving the Design of Existing Code (2000)) // Спб: Символ-Плюс, 2009. – 432 с. – ISBN: 5-93286-045-6.

[7] Руководство по оформлению Markdown файлов: `https://gist.github.com/Jekins/2bf2d0638163f1294637`.