Соглашение о стандарте кодирования при выполнении лабораторных и практических работ по дисциплине «Системное программирование»

Редакция 1.5 от 05.02.2015

0x00. Введение. Общие положения

0x0001. Основная цель настоящего Стандарта – улучшение читабельности и понимания кода, и как следствие, его общего качества.

0x0002. В силу того, что нельзя описать все возможные случаи, разработчики Стандарта старались придать ему лаконичный и в то же время гибкий характер.

0x0003. Разработчики Стандарта тоже люди и могут ошибаться. Они будут признательны за все сообщения об ошибках, недочетах, а также за ценную критику и предложения.

0x0004. Большая часть кода должна создаваться на языке C стандарта C99. Остальные — язык ассемблера процессорных систем семейства *x*86, язык командной оболочки *bash*, входной язык утилиты GNU *make*.

0x01. Именование файлов

0x0101. Исходный код программ хранится в файлах, поэтому очень важно следить за тем, чтобы у них были осмысленные имена и, по возможности, содержательные. Например, *file_processing.c* вместо *file1.c*. Имена файлов с исходными текстами должны иметь суффиксы (в некоторых ОС они называются расширениями), отражающими используемые языки программирования.

0x0102. Код, создаваемый на языке C, должен размещаться в файлах, имена которых заканчиваются суффиксами *.*c* (программная реализация) и *.*h* (описание программного интерфейса).

0x0103. Код, создаваемый на языке ассемблера *x*86, должен размещаться в файлах, имена которых заканчиваются суффиксом *.*s*.

0x0104. Код, создаваемый на языке командной оболочки *bash*, должен размещаться в файлах, имена которых заканчиваются суффиксом *.*sh*.

0x0105. Код *make*-файлов подчиняется общим правилам именования входных файлов утилиты GNU *make*.

0x02. Параметры текста, пробелы, табуляции

0x0201. Рекомендуемая ширина строки текста исходного кода до 78 символов. Данная ширина обеспечивает наилучшее визуальное восприятие кода программы и в большинстве случаев умещается на экране. Максимальная ширина строки не должна превышать 99 символов и определяется шириной печатного листа при установках шрифта в используемой среде программирования по умолчанию.

0x0202. Пробелами **с обеих сторон** должны выделяться знаки **бинарных** операций (два исключения — операции '.' и '->'). Пробелы с одной стороны ставятся только перед фигурными скобками, а также перед круглыми скобками, если до них указано ключевое слово языка C. Пробелы с одной стороны ставятся после запятой, точки с запятой или двоеточия, если эти символы не являются последней конструкцией в строке кода.

0x0203. Организация отступов в коде символами табуляции запрещена. В место них в обязательном порядке используются пробелы. Количество пробельных символов для отступов должно быть одинаковым по всему исходному коду и кратным первой, второй или третьей степени двойки. Иначе говоря, один шаг отступа может содержать только 2, 4 или 8 пробела. Многие средства редактирования текста позволяют заменять символы табуляции заданным количеством пробелов.

0x03. Комментирование

0x0301. Если это не оговорено особо, то в программах на языке С обязательно использование *doxygen*-комментариев.

0x0302. Комментарий выравнивается по левому краю того блока, к которому он относится. При организации многострочного комментария, лучше пользоваться обозначением, принятым в С99, т.е. “//” вместо “/* ... */”. Это позволит в дальнейшем быстро «закомментировать» часть кода программы с помощью “/* ... */”.

0x0303. Не следует чередовать комментарии “//” и “/* ... */”, если на то нет веских причин.

0x0304. При расположении комментария на одной строке с оператором, необходимо сделать отступ минимум на два пробела. Необходимо следить за тем, как данный комментарий соотносится с соседними. Для большей наглядности, рекомендуется выравнивать такие комментарии в столбцы.

0x0305. Комментарий всегда должен быть виден на экране и не выходить за его пределы.

0x0306. Текст комментария рекомендуется отделять от обозначения “//” как минимум одним пробелом. После каждого знака препинания внутри комментария должен стоять пробел.

0x0307. Если внутри многострочного комментария, оформляемого с помощью “//”, встречается пропуск строки, в такую строку также добавляется “//”, для подчеркивания единого блока. Пример:

```
int32_t sort_array(const int* pArray, const int size)
{
  // Здесь начинается первая часть дополнительной информации о работе функции
  // ... что-то
  // конец первой части.
  //
  // Вторая часть дополнительной информации
  // ... что-то
  // конец второй части.

  int count = 0;  // Назначение переменной
  int temp  = 0;  // Назначение объекта


  // Пояснения к работе данного блока программы
  {

      // ... делать что-то
   }

  // ...

  return 0;
}
```

0x04. Именование и использование типов и программных объектов в программах на языке С

0x0401. В общем случае важно следить за тем, чтобы у типов и программных объектов были осмысленные и содержательные, по возможности, названия.

Все имена рекомендуется писать на английском языке, что во многих случаях позволяет повысить степень понимания программ при чтении. Это также дает информацию о цели использования конкретного имени. Для составных имен допустимыми являются две схемы – *snake_case* и *Camel Case*. В *snake_case* слова отделяются друг от друга подчеркиванием и записываются строчными буквами. В *Camel Case* повсюду, за исключением глобальных макроопределений, слова не

отделяются подчеркиванием, начальные литеры слов — заглавные, остальные — строчные. Следует придерживаться единообразного схемы именования, но для глобальных макроопределений в любом случае используется *snake_case*.

Нежелательно использовать аббревиатуры, так как они могут затруднить понимание и расшифровку для лиц, которые будут читать код, в том числе имена функций и переменных.

Запрещается использовать имена из 1 и 2 символов, за исключением переменных циклов и временных переменных, которые являются локальными для функции и используются для целей, очевидных читателю кода. Например, в следующем фрагменте кода очевидно использование целочисленной переменной *t* как временного хранилища информации при взаимном обмене ею между двумя объектами.

```
int FirstValue  = 10;
int SecondValue = 20;
int t = FirstValue;
FirstValue = SecondValue;
SecondValue = t;
```

Желательно избегать имен, в которых смешиваются цифры 0 и 1 и литеры *O* (*заглавная «оу»*), *I* (*заглавная «ай»*), *l* (*строчная «эль»*), поскольку они могут не различаться при чтении (произнесении) кода.

Запрещается использовать имена, которые отличаются только регистром написания, например, *kuku* и *KUKU.* Исключение могут составить объявления типа *String string*, где имя типа и имя переменной различаются регистром, хотя такая конструкция может обескуражить читателя кода.

0x0402. Типы
Для имен типов рекомендуется использовать имена существительные. Нежелательно, чтобы количество слов, составляющих имя типа, превышало три.

Для *snake_case* рекомендуется использовать литеру 't' в качестве суффикса. Например:

```
typedef unsigned int number_t;

typedef struct any_struct any_t;
```

Для *Camel Case* рекомендуется использовать литеру 'T' в качестве префикса. Например:

```
typedef unsigned int TNumber;

typedef struct AnyStruct TAny;
```

0x0403. Переменные
Именование переменных, включая аргументы функций, в целом, подчиняется тем же правилам, что и типы. В языке С переменные по отношению к блокам операторов и функциям могут быть локальными и нелокальными. При этом нелокальные переменные самого верхнего уровня являются глобальными.

Для глобальной переменной независимо от принятой схемы имя должно иметь префикс ”g_”, с очевидным назначением. Например, идентификатор *g_FirstName* имеет глобальную область видимости и носит смысл имени некоей персоны. При этом использование глобальных переменных должно быть сведено к минимуму

Имена локальных переменных, обычно размещаемых компиляторами в стеке, должны полностью писаться строчными буквами. Все локальные переменные должны объявляться в начали области их видимости, как можно ближе к месту первого их использования. Со своей стороны область видимости таких переменных должна быть как можно меньше.

Взаимосвязанные переменные одного типа можно объявлять в одном общем операторе. Например:

```
int value, index, counter;
```

Указатели должны начинаться с префикса *p*. Знак указателя (*) должен располагаться возле имени типа, а не имени переменной.

Для указателей следует обязательно придерживаться правила: одно объявление – одна строка кода.

```
int* pValue;
```

0x0404. Константы и элементы перечислений
В целом, имена констант должны отвечать тем же требованиям, что и приведенные выше программные объекты. Глобальная константа в виде макроопределения и элемент перечисления должна полностью писаться заглавными буквами в *snake_case*. Например,

```
#define MAX_ITERATION 333
```

Константа может быть объявлена как неизменяемая глобальная переменная, с соблюдением соответствующих им правил.

```
const int g_constant = 25;
```

Запрещается использование **«магических констант»,** то есть таких литералов, которые встречаются в коде без каких-либо объяснений. Вместо них должны использоваться символьные литералы, за исключением редких случаев, оговоренных ниже. Именование литералов должно делаться по правилам, приведенным выше. Например, цикл

```
for (int i = 0; i < 33; ++i)
{
  total = total + 1;
}
```

должен быть заменен на

```
for (int i = 0; i < MAX_ITERATION - 1; ++i)
{
  ++total;
}
```

Разрешается использовать жестко заданные 0 и 1 для инкремента, декремента, а также в начале циклов при нумерации первого элемента массива.

Данное правило распространяется: а) на числовые литералы; б) на коды символов (вместо них желательно использовать символьные константы); в) на строковые литералы, если они используются в коде программы больше одного раза; г) на логические константы.

0x0405. Функции
Обычно функция выполняет некое действие. Следовательно, их имена должны давать понять, что **делает** конкретная функция. На имена функций распространяются те же ограничения, что на программные объекты или типы, то есть несколько слов английского языка. Рекомендуется ограничиться форматами глагол-существительное (*DeleteFile*), прилагательное-глагол или причастие-глагол (*FoundFile*).

Объявление функции должно содержать тип возвращаемого значения, имя функции, и параметры (или слово *void* в том случае, если параметры отсутствуют).

При объявлении параметров функции желательно указывать тип параметра и его имя. Допускается использовать сокращенную запись, при которой указывается только тип аргумента.

Перед объявлением каждой функции следует писать *doxygen*-комментарий, содержащий информацию о назначении функции. Исключение составляют функции, чье назначение очевидно из названия самой функции, а код содержит небольшое количество строк. Комментарий располагается непосредственно перед объявлением функции.

При объявлении и определении функции имя типа возвращаемого значения, имя функции, открывающая скобка и первый аргумент (при наличии) должны находиться на одной строке, причем между именем функции и открывающей скобкой, а также между открывающей скобкой и первым аргументом можно не ставить никаких разделителей. Если место на строке позволяет, то остальные аргументы и закрывающая скобка могут находиться на этой же строке. Если места на строке не остается, то остаток списка последующих аргументов пишется на новой строке.

Если тип возвращаемого функцией значения имеет слишком длинное имя, допускается его выносить на отдельную строку перед именем функции.

Примеры:

```
void SetPoint(int x, int y);

void GetPointColor(int x,
  int y,
  TWindowSubclass* pMyWindowSubclass,
  TOtherWindowSubclass* pMyOtherWindowSubclass
)
{
  // ...
}
```

Каждая функция комментируется так, чтобы документировались ее назначение, аргументы, возвращаемые значения и дополнительную информацию о работе функции. Сюда заносится информация об ограничениях, накладываемых на работу, побочные эффекты функции, краткое описание неочевидных из текста кода вещей, которые помогают лучше и быстрее понять логику работы функции.

При оформлении вызова функции после имени функции без разделяющего пробела может ставиться открывающая круглая скобка. Далее идет список аргументов функции (если они есть), который заканчивается закрывающей круглой скобкой и точкой с запятой. Нигде, кроме как между аргументами не должны ставиться пробелы. Аргументы функции разделяются запятой и пробелом.

Пример:

```
HRESULT   hr = S_OK;
const int g_Max_len = 10;
TChar     buffer[g_Max_len];

// ...

hr = PutTextToEditWindow(buffer, g_Max_len);
```

В списке аргументов входные параметры должны предшествовать выходным. Количество аргументов функций рекомендуется ограничить 5-7.

В случае если аргументов у функции очень много или они имеют длинные имена и данная конструкция не умещается на одной строке, вызов оформляется способом, похожим на описанное выше правило для объявления и определения функции. Единственное отличие – отступы для аргументов и закрывающей круглой скобки делаются по отношению к открывающей круглой скобке. Пример:

```
HRESULT hr = S_OK;

hr = AnyFunction(XPosition,              // комментарий
                 YPosition,              // комментарий
                 pMyWindowSubclass,      // комментарий
                 pMyOtherWindowSubclass  // комментарий
                );
```

Не рекомендуется осуществлять реализацию функций, занимающих более 160 строк текста с учетом комментариев. Это повод задуматься о ее декомпозиции на несколько более мелких функций.

0x05. Кодирование и форматирование управляющих конструкций в программах на языке С

0x0501. Последовательность операторов

Два основных принципа, которым следует придерживаться:
1. Один простой оператор рекомендуется располагать на одной строке кода. Пустой оператор также является простым. Пример:

```
while (i < MAX_ITERATION - 1)
  ;
```

2. Взаимосвязанные действия должны располагаться близко и не накладываться друг на друга.

0x0502. Блок операторов

Он должен быть замкнут на операторные скобки, которые также должны восприниматься как простые операторы, форматированные по принципу один оператор — одна строка кода. Например,

```
grandTotal = 0;
for (int i = 0; i < MAX_ITERATION - 1; ++i)
{
  total = 0;
  for (int j = i; j < GLOBAL_CONSTANT; ++j)
  {
    ++total;
  }
  grandTotal += total;
}
```

Исключением являются операторы инициализирующего объявления переменных и операторы цикла с постусловием (см. п. 0x0505 ниже).

Максимальный уровень вложенности операторов должен принимать значение, не превышающее **четырех**, в редких, хорошо документированных случаях — **пяти**.

0x0503 Условные операторы *if* и *if-else*

В различных ветвях выполнения операторов *if* и *if-else* сначала должен записываться код для нормального хода выполнения алгоритма, а потом в блоке *else* — исключительный случай.

Рекомендуется во всех случаях указывать блок *else*, даже если он оказывается пустым.

При вычислении логических выражений, содержащих большое количество логических операторов, рекомендуется каждый из них размещать на отдельной строке. Например:

```
if (IsLetter(inputCharacter)
    || IsDigit(inputCharacter)
    && IsControl(inputCharacter)
   )
{
  // нормальная ветвь выполнения алгоритма
}
else
{
    // исключительный случай
}
```

Код для последовательности условных операторов должен форматироваться по похожему принципу. Например:

```
if (IsLetter(inputCharacter))
{
  // буква
```

```
}
else if (IsDigit(inputCharacter))
{
  // цифра
}
else if IsControl(inputCharacter))
{
  // управляющий символ
}
else
{
  // остальные символы
}
```

0x0504. Оператор множественного выбора
Оператор множественного выбора также должен форматироваться аналогично последовательности операторов проверки условия. На *case*-метки не распространяется правило отступов из параграфа 0x0203 настоящего Стандарта.

Пример:

```
switch (inputCharacter)
{
case LETTER:
  printLetter(inputCharacter);
  break;
case DIGIT:
  CountDigit(inputCharacter);
  break;
default:
  DisplayInternalError();
}
```

Для большей универсальности и читаемости рекомендуется всегда указывать секцию *default*, даже если она пуста. **Неиспользование** оператора *break* во всех или части *case*-веток вычислений должно быть очевидным или подробно задокументировано. Использование вложенных операторов множественного выбора не рекомендуется. Как правило, это повод задуматься о декомпозиции кода.

0x0505. Операторы цикла
На циклы с постусловием и предусловием распространяются те же правила форматирования,.что и на условные операторы.

В циклах с постусловием ключевое слово *while* , за которым следует условное выражение, должно находиться в одной строке с закрывающей фигурной скобкой. Это исключительный случай по отношению к правилу оформления блока операторов (см. п. 0x0502 выше).

```
do
{
  ++i;
} while (i < MAX_ITERATION - 1);
```

Компоненты заголовочной части универсального оператора цикла *for* рекомендуется размещать на отдельных строках. Например,

```
for (int i = 0;              // Инициализация переменной цикла
     i < MAX_ITERATION - 1;  // Цикл выполнится такое число раз
     ++i                     // Инкремент переменной цикла
    )
{
  // Тело цикла
}
```

Использование бесконечного цикла должно документироваться до его начала.

Рекомендуется избегать использования имен переменных циклов, не отвечающих конвенции именования обычных переменных, описанной выше.

0x0506. Тернарный оператор
Его компоненты должны располагаться в одной строке, например:

```
c = (a == b) ? d + f(a) : f(b) - d;
```

При невозможности такого форматирования все компоненты тернарного оператора располагаются на отдельных строках. Пример:

```
c = (a == b)
  ? d + f(a)
  : f(b) - d;
```

0x0507 Операторы безусловного перехода
Использование операторов типа *goto* разрешается, только при рассмотрении их в качестве последнего варианта, если другими способами не удается реализовать требуемую логику. В любом случае в коде это должно тщательно документироваться.

0x06. Замечание по языкам *bash* и ассемблер

0x0601. При использовании данных языков приведенные выше правила носят рекомендательный характер.

0x0602. Для языка ассемблера обязательно использование синтаксиса AT&T [15.].

0x07. Заключение

В связи с тем, что в реальной практике всегда может встретиться ситуация, явно не описанная в данном документе, следует принимать решение, максимально согласующееся с данным стандартом. Решения о любых отклонениях от данного стандарта должны приниматься по согласованию с преподавателем.

0x08. Использованные и рекомендованные ресурсы

1. Linux kernel coding style - https://www.kernel.org/doc/Documentation/CodingStyle
2. C Coding Standard - http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html
3. Recommended C Style and Coding Standards - https://www.doc.ic.ac.uk/lab/cplus/cstyle.html
4. GNU Coding Standards - http://www.gnu.org/prep/standards/standards.html
5. Стандарт кодирования GNU - http://linuxshare.ru/docs/devel/standard/standard.html
6. http://www.chris-lott.org/resources/cstyle/
7. Doxygen - http://www.stack.nl/~dimitri/doxygen/index.html
8. Лучшие приемы программирования на C - http://www.ibm.com/developerworks/ru/library/au-hook_duttaC/index.html
9. Системный подход к выбору идентификаторов - http://www.osp.ru/pcworld/2007/11/4656416/
10. Соглашения по оформлению кода команды RSDN - http://www.rsdn.ru/article/mag/200401/codestyle.XML
11. Holub, A. Enough Rope to Shoot Yourself in the Foot: 195 Rules for C++ and C Programming. - NY: McGraw-Hill Companies, 1995. - 208 pp.
12. Макконнелл, С. Совершенный код. Мастер-класс. - М.: Издательско-торговый дом «Русская редакция», СПб.: Питер, 2005. - 896 с.

13. Как писать программы без ошибок - http://wiki.pic24.ru/doku.php/osa/articles/encoding_without_errors
14. C and C++ Style Guides - http://www.maultech.com/chrislott/resources/cstyle/
15. AT&T-синтаксис - https://ru.wikipedia.org/wiki/AT%26T-синтаксис