Micriμm, Inc.

Стандарт кодирования на Си

Application Note
AN-2000

Jean J. Labrosse
Jean.Labrosse@Micrium.com
www.Micrium.com

C Coding Standard
Application Note AN-2000

Перевод: Андрей Шлеенков
http://andromega.narod.ru
mailto:andromega@narod.ru

Содержание

1.00 Введение

Соглашения о стиле программирования должны быть установлены в самом начале проекта. Эти соглашения необходимы для поддержания непротиворечивости всего проекта в целом. Принятие соглашений увеличивает производительность и упрощает сопровождение проекта.

Имеется много стилей программирования на Си (как и на любом другом языке). Стиль, который используете вы, может быть так же хорош, как и любой другой, пока вы стремитесь к достижению следующих целей:

- мобильность
- непротиворечивость
- аккуратность
- сопровождаемость
- понимание
- простота

Любой стиль, который вы используете, и я подчеркнул бы это, должен применяться последовательно во всех ваших проектах. Далее, я настоял бы, чтобы единый стиль был бы принят всеми членами группы в большом проекте. Принятие общего стиля уменьшает проблемы и издержки сопровождения и избавляет от многократного переписывания кода. Следующие соображения описывают стиль, который автор применял в течение многих лет.

2.00 Базовые принципы

Фундаментальная цель этих стандартов состоит в том, чтобы поддержать сопровождение кода. Это означает, что код должен быть читабельным, понятным, тестируемым и переносимым.

Следуйте духу стандарта.

Когда принимаются решения относительно кодирования и отсутствует официальный общепринятый стандарт, вы должны всегда придерживаться выбранного вами собственного стандарта.

Весь код должен быть написан в стандарте ANSI C.

Это означает, что прототипы функций всегда должны присутствовать и быть оформлены в стиле ANSI и, следовательно, определения типов аргументов должны быть заключены внутри круглых скобок.

Стремитесь к простоте кода.

Пишите ясно.

Избегайте неявных или малоизвестных возможностей языка. Говорите то, что имеете в виду.

Будьте последовательны.

Используйте одни и те же правила везде, где возможно.

Избегайте сложных выражений.

Выражения, включающие много подвыражений, трудно изучать и особенно тестировать.

Не используйте оператор GOTO.

Модификация старого кода.

Всякий раз при изменении существующего кода пытайтесь его модифицировать, соблюдая соглашения, описанные в данном документе. Это создает уверенность, что код может быть расширен снова через какое-то время.

3.00 Исходные файлы

Ширина строки.

Вы не должны ограничивать ширину исходного текста Си 80 символами только потому, что старые мониторы позволяли отображать текст шириной только до 80 символов. Ширина строки может быть основана на том, сколько символов могут быть напечатаны в строке страницы, используя приемлемый размер шрифта. Вы должны иметь возможность разместить до 132 символов (в режиме печати «Портрет») и иметь достаточно места с левого края страницы для отверстий для вставки страницы в папку при помощи брошюратора. Разрешение 132 символа на строку избавляет от необходимости чередовать исходный текст с комментариями. Если необходимо большее количество символов чтобы сделать код более ясным, то вы не должны ограничиваться и 132 символами. На практике вы можете иметь код, который содержит инициализированные структуры (помещенные в память только для чтения) которые, например, имеют ширину более 300 символов. Конечно, вы не можете увидеть (или напечатать) все элементы этих таблиц сразу, но, по крайней мере, различные поля могут быть аккуратно выровнены.

Использование символа табуляции – TAB

Символы табуляции – TAB (ASCII символ **0x09**) НЕ ДОЛЖНЫ использоваться вообще. Выравнивание ДОЛЖНО быть выполнено, используя только символ SPACE – пробел (ASCII символ **0x20**). Символы табуляции по-разному расширяются на различных компьютерах и принтерах. Отсутствие их гарантирует, что будет выдерживаться задуманный интервал.

Уровень отступа – 4 пробела.

Выравнивание кода должно быть кратно 4 пробелам (ASCII символы **0x20**). Обратите внимание, что значения в операторе выбора `case` на самом деле выровнены 5 пробелами (см. раздел Конструкции). Всегда пытайтесь начинать выражения на позициях, отличающихся 4 пробелами (столбцы 1, 5, 9, 13, 17 и т.д.).

Включайте заголовок файла.

В начале каждого файла включайте блок комментария (этикетку), содержащий название компании, адрес, авторские права, список программистов, описание файла, и т.д. См. ниже.

```
/*
*********************************************************************************
*                                    Company Name
*                                  Company Address
*                                  City, State ZIP
*                                      Country
*
*                     (c) Copyright YYYY, Company Name, City, State
*
* All rights reserved. Company Name's source code is an unpublished work and the
* use of a copyright notice does not imply otherwise. This source code contains
* confidential, trade secret material of Micrium, Inc. Any attempt or
* participation in deciphering, decoding, reverse engineering or in any way
* altering the source code is strictly prohibited, unless the prior written
* consent of Company Name is obtained.
*
* Filename :
* Programmer(s): Joe Programmer (JP)
*                John Doe (JD)
* Created : YYYY/MM/DD
* Description :
*********************************************************************************
*/
```

Файл реализации – это файл, который содержит выполнимые операторы, тогда как заголовочный файл их не содержит. Оба типа файлов состоят из схожих разделов, перечисленных ниже. Файл может содержать все эти разделы или их часть. Пустые разделы могут быть опущены, но, если раздел включен, он должен следовать в порядке, показанном ниже.

План файла реализации:

- Заголовок файла
- Хронология изменений
- Раздел `#include`
- Раздел `#define constants`
- Макросы
- Локальные типы данных
- Локальные переменные
- Локальные таблицы
- Локальные прототипы функций
  В порядке реализации
- Глобальные функции
  В порядке функциональности
- Локальные функции
  В порядке функциональности

План заголовочного файла:

- Заголовок файла
- Хронология изменений
- Раздел `#define constants`
- Глобальные макросы
- Глобальные типы данных
- Глобальные переменные
- Внешние объявления
- Глобальные прототипы функций
  В том же порядке, как и в файле реализации
  (отделите разделы для функций, объявленных в разных файлах)
- Раздел `#error`
  Раздел индикации отсутствующих или недопустимых `#define`.

Отделяйте основные разделы.

Каждому разделу должен предшествовать блок комментария, как показано ниже.

```
/*
*********************************************************************************
*                                       DATA TYPES
*********************************************************************************
*/

typedef unsigned char BOOLEAN;

/*
*********************************************************************************
*                                       PROTOTYPES
*********************************************************************************
*/

BOOLEAN OSIsTaskRdy(void);
```

Заголовочные файлы ДОЛЖНЫ быть защищены от двойного включения проверкой значения определения макроса.

Заметьте, что `!defined(X)` предпочтительней, чем `#ifndef X`.

```
#if !defined(module_H)
#define module_H

  Тело заголовочного файла.

#endif /* End of module_H */
```

Используйте #error для сообщения об отсутствии определения константы или макроса и проверяйте значения на допустимость.

Стандартная директива `#error` препроцессора Си должна использоваться, чтобы сообщить программисту об отсутствии определения константы или макроса или указать, что определенное значение находится вне допустимого диапазона. Эти утверждения обычно находятся в заголовочных `*.H` файлах модулей. Директива `#error` отобразит сообщение внутри двойных кавычек при выполнении указанного условия.

```
#ifndef OS_MAX_TASKS
#error "OS_CFG.H, Missing OS_MAX_TASKS: Max. number of tasks in your application"
#else
    #if OS_MAX_TASKS < 2
    #error "OS_CFG.H, OS_MAX_TASKS must be >= 2"
    #endif
    #if OS_MAX_TASKS > 63
    #error "OS_CFG.H, OS_MAX_TASKS must be <= 63"
    #endif
#endif
```

4.00 Комментирование

Используйте комментарии только в стиле Си (/* и */), и не используйте комментарии в стиле C++ (//).

Применяйте только существенные, значимые комментарии.

Располагайте код и комментарии визуально раздельно.

Минимизируйте комментарии, расположенные между операторами. НИКОГДА не начинайте комментарии непосредственно над кодом как показано ниже. Это затрудняет изучение кода, потому что комментарии отвлекают от визуального следования последовательности кода.

```
void ClkUpdateTime (void)
{                                    /* Не комментируйте так, как показано ниже! */
    /* Update the seconds */
    if (ClkSec >= CLK_MAX_SEC) {
        ClkSec = 0;
        /* Update the minutes */
        if (ClkMin >= CLK_MAX_MIN) {
            ClkMin = 0;
            /* Update the hours */
            if (ClkHour >= CLK_MAX_HOURS) {
                ClkHour = 0;
            } else {
                ClkHour++;
            }
        } else {
            ClkMin++;
        }
    } else {
        ClkSec++;
    }
}
```

Не используйте многострочные комментарии с одиночным терминатором.

НИКОГДА не делайте следующее, потому что имя функции выглядит как фактический код:

```
/* This comment can lead to confusion especially when describing a function like
   ClkUpdateTime (). The function looks like actual code! */
```

Используйте блоки комментария, чтобы отделить разделы кода.

Блок комментария показан ниже. Обратите внимание на то, что заголовок блока комментария центрирован и написан используя символы ВЕРХНЕГО регистра.

```
/*
*********************************************************************************
*                                      VARIABLES
*********************************************************************************
*/
```

Не используйте 'эмоции' в комментариях.

Например, НЕ используйте комментарии, типа “Давайте сделаем эту большую красивую структуру!”. Используйте структурированные предложения в максимально возможной степени. Вы можете использовать ВЕРХНИЙ регистр, чтобы подчеркнуть значение некоторых слов. Также возможно использование акронимов, сокращений и мнемоник, если все понимают их значение.

Насколько возможно используйте только концевые комментарии.

По возможности всегда начинайте концевые комментарии в одном и том же столбце. Если код заходит в позиции комментариев, помещайте комментарий выше строки кода, начиная его в столбце для комментариев. Если возможно, выравнивайте символы завершения комментария. Использование концевых комментариев позволяет им быть визуально отделенным от кода.

```
void ClkUpdateTime (void)
{
    if (ClkSec >= CLK_MAX_SEC) {                    /* Update the seconds */
        ClkSec = 0;
        if (ClkMin >= CLK_MAX_MIN) {                /* Update the minutes */
            ClkMin = 0;
            if (ClkHour >= CLK_MAX_HOURS) {         /* Update the hours   */
                ClkHour = 0;
            } else {
                ClkHour++;
            }
        } else {
            ClkMin++;
        }
    } else {
        ClkSec++;
    }
}
```

Используйте #if 0 и #endif для комментирования блоков кода.

Комментарии никогда не должны быть вложенными. Вместо этого, используйте **`#if 0`** и **`#endif`**, чтобы *закомментировать* большие части кода.

```
#if 0                           /* Указание причины комментирования кода.       */
#define  DISP_TBL_SIZE  5       /* Size of display buffer table                 */
#define  DISP_MAX_X     80      /* Max. number of characters in X axis          */
#define  DISP_MAX_Y     25      /* Max. number of characters in Y axis          */
#define  DIS            0x5F
#endif
```

Используйте специальные комментарии для указания наличия известных ошибок, а также прошлых и будущих реализаций.

Например, вы можете использовать следующие комментарии. Вы можете затем легко проводить поиск по вашему коду, чтобы найти эти пометки.

```
/* ???? Ошибка или известная проблема */

/* $$$$ Будущая функция, требующая реализации */

/* @@@@ Старый код, оставленный потому, что ... */
```

5.00 Соглашения об именах

Общие соглашения:

- **#define constants:**
- **#define macros:**
- **typedefs:**
- **enum tags:**
  - **Все символы в верхнем регистре.**
  - **Слова разделяются подчеркиванием (т.е. '_').**
  - **Примеры: DISP_BUF_SIZE, MIN(), MAX(), и т.п.**
- **Локальные переменные (т.е. в области видимости функции):**
  - **Все символы в нижнем регистре.**
  - **Слова разделяются подчеркиванием (т.е. '_').**
  - **Стандартные имена (т.е. i, j, k для счетчиков, p для указателей и т.п.).**
- **Переменные в области видимости файла:**
  - **Префикс из имени модуля, сопровождаемый подчеркиванием (т.е. '_').**
  - **Именование отдельных слов, начиная с заглавной буквы.**
  - **Объявляются static.**
  - **Примеры: Disp_Buf[], Comm_Ch, и т.п.**
- **Глобальные переменные:**
  - **Префикс из имени модуля.**
  - **Именование отдельных слов, начиная с заглавной буквы.**
  - **Примеры: DispMapTbl[], CommErrCtr, и т.п.**
- **Локальные функции:**
  - **Префикс из имени модуля, сопровождаемый подчеркиванием (т.е. '_').**
  - **Именование отдельных слов, начиная с заглавной буквы.**
  - **Объявляются static.**
  - **Пример: static void Comm_PutChar()**
- **Глобальные функции:**
  - **Префикс из имени модуля.**
  - **Именование отдельных слов, начиная с заглавной буквы.**
  - **Пример: void CommInit()**

Отдельные слова начинайте с заглавной буквы (пример, DispBuf[]).

Старый код, содержащий символы только строчных букв ДОЛЖЕН разделяться подчеркиванием (то есть '_', ASCII символом **0x2D**).

Используйте формат 'module-object-operation' для аббревиатур.

При создании идентификаторов глобальных констант, переменных и функций, сначала определите имя модуля (или подсистемы) с последующим именем объекта и затем операцией как показано ниже.

```
OSSemPost()
OSSemPend()
```

и т. д.

ВСЕГДА используйте стандартные аббревиатуры.

Всегда используйте аббревиатуру, даже если можно написать полное слово. Например, ВСЕГДА используйте **`Init`** вместо **`Initialize`**!

Используйте аббревиатуры.

Создайте словарь стандартных аббревиатур для общего использования и освоения. Ниже приводится неполный пример такого списка. Также должен быть сгенерирован список, сортируемый обратно по полю Аббревиатура.

Словарь аббревиатур

Термин	Аббревиатура
Argument	Arg
Buffer	Buf
Clear	Clr
Clock	Clk
Compare	Cmp
Configuration	Cfg
Context	Ctx
Delay	Dly
Device	Dev
Display	Disp
Error	Err
Function	Fnct
Hexadecimal	Hex
High Priority Task	HPT
I/O System	IOS
Initialize	Init
Mailbox	Mbox
Manager	Mgr
Maximum	Max
Message	Msg
Minimum	Min
Operating System	OS
Overflow	Ovf
Pointer	Ptr
Previous	Prev
Priority	Prio
Read	Rd
Ready	Rdy
Schedule	Sched
Semaphore	Sem
Stack	Stk
Synchronize	Sync
Timer	Tmr
Trigger	Trig
Write	Wr

6.00 Типы данных

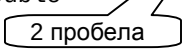
Все типы данных ДОЛЖНЫ быть объявлены, используя символы верхнего регистра.

Слова отделяются символом подчеркивания ('_', ASCII символ **0x2D**).

Используйте следующие переносимые типы данных.

НЕОБХОДИМО избегать применения стандартных типов данных Си, потому что их размер не является переносимым. Вместо этого, на основании целевого процессора и используемого компилятора, должны быть объявлены следующие типы данных (**INT??** и **FP??**).

```
typedef  unsigned  char   BOOLEAN;    /* Logical data type (TRUE or FALSE) */
typedef  unsigned  char   CHAR;       /* Unsigned 8 bit character */
typedef  unsigned  char   INT08U;     /* Unsigned 8 bit value */
typedef  signed    char   INT08S;     /* Signed 8 bit value */
typedef  unsigned  short  INT16U;     /* Unsigned 16 bit value */
typedef  signed    short  INT16S;     /* Signed 16 bit value */
typedef  signed    int    INT32U;     /* Unsigned 32 bit value */
typedef  signed    int    INT32S;     /* Signed 32 bit value */
typedef  signed    long   INT64U;     /* Unsigned 64 bit value (if available)*/
typedef  signed    long   INT64S;     /* Signed 64 bit value (if available)*/
typedef  float            FP32;       /* 32 bit, single prec. floating-point */
typedef  double           FP64;       /* 64 bit, double prec. floating-point */
```



Структуры и объединения ДОЛЖНЫ быть типизированы.

Все структуры и объединения ДОЛЖНЫ быть типизированы, как показано ниже. Также, тип данных ДОЛЖЕН быть объявлен, используя ВСЕ символы в верхнем регистре. Контекст кода сделает очевидным, что все символы в верхнем регистре в начале переменной или функции должны означать, что это тип данных в противоположность константе или макросу.

```
typedef struct {
    char     RxBuf[COMM_RX_SIZE];    /* Storage of characters received */
    char    *RxInPtr;                /* Pointer to next free loc. in buffer */
    char    *RxOutPtr;               /* Pointer to next char. to extract */
    INT16U   RxCtr;                  /* Number of characters in Rx buffer */
    char     TxBuf[COMM_TX_SIZE];    /* Storage for characters to send */
    char    *TxInPtr;                /* Pointer to next free loc. in Tx Buf */
    char    *TxOutPtr;               /* Pointer to next char to send */
    INT16U   TxCtr;                  /* Number of characters left to send */
} COMM_BUF;
```

Выравнивание структуры.

Тип данных каждого элемента выравнивается 4 пробелами, а также имена элементов структуры выравниваются относительно друг друга. Обратите внимание также, что комментарии выровнены, начинаясь в одном и том же столбце (возможно начинаясь на предыдущей строке, если элемент структуры не заканчивается до позиции комментария).

Область видимости типа данных.

Если тип данных должен использоваться только в файле реализации, то он ДОЛЖЕН быть объявлен в файле реализации. Если тип данных глобален, то он ДОЛЖЕН быть помещен в заголовочный файл модуля.

7.00 Форматирование

Используйте одно действие на одну строку программы.

Пример №1:

```
DispSegTblIx = 0;
DispDigMsk   = 0x80;
```

Вместо:

```
DispSegTblIx = 0; DispDigMsk = 0x80;
```

Пример №2:

```
ptcb++;
*ptcb = (OS_TCB *)0;
```

Вместо:

```
*++ptcb = (OS_TCB *)0;
```

Отделяйте секции кода пустыми строками или комментариями.

Между именем функции и круглыми скобками после него не должно быть пробелов при вызове функции.

```
DispInit();
```

Как минимум один пробел необходим после каждой запятой для разделения аргументов функции.

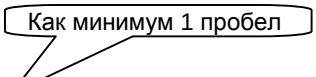
```
DispStr(x, y, s);
```

Унарные операторы пишутся без пробела между ними и их операндом.

```
!value
~bits
++i
j–
(INT32U)x
*ptr
&x
sizeof(x)
```

Бинарные операторы и тернарный оператор пишутся как минимум с одним пробелом между ними и их операндом.

```
c1 = c2;
x + y
i += 2;
n > 0 ? n : -n;
a < b
c >= 2
```



Как минимум один пробел необходим после каждой точки с запятой

```
for (i = 0; i < 10; i++)
```

Ключевые слова if, else, while, for, switch и return сопровождаются одним пробелом.

```
if (a > b)
while (x > 0)
for (i = 0; i < 10; i++)
} else {
switch (x)
return (y);
```

В присваиваниях знаки равенства и числа должны быть вертикально выровнены, чтобы их размещение было аккуратным.

Обратите внимание также, что младшая часть целых чисел и число с плавающей точкой выровнены в один столбец.

```
DispSegTblIx =    0;
DispDigMsk   = 0x80;
DispScale    =    1.25;
```

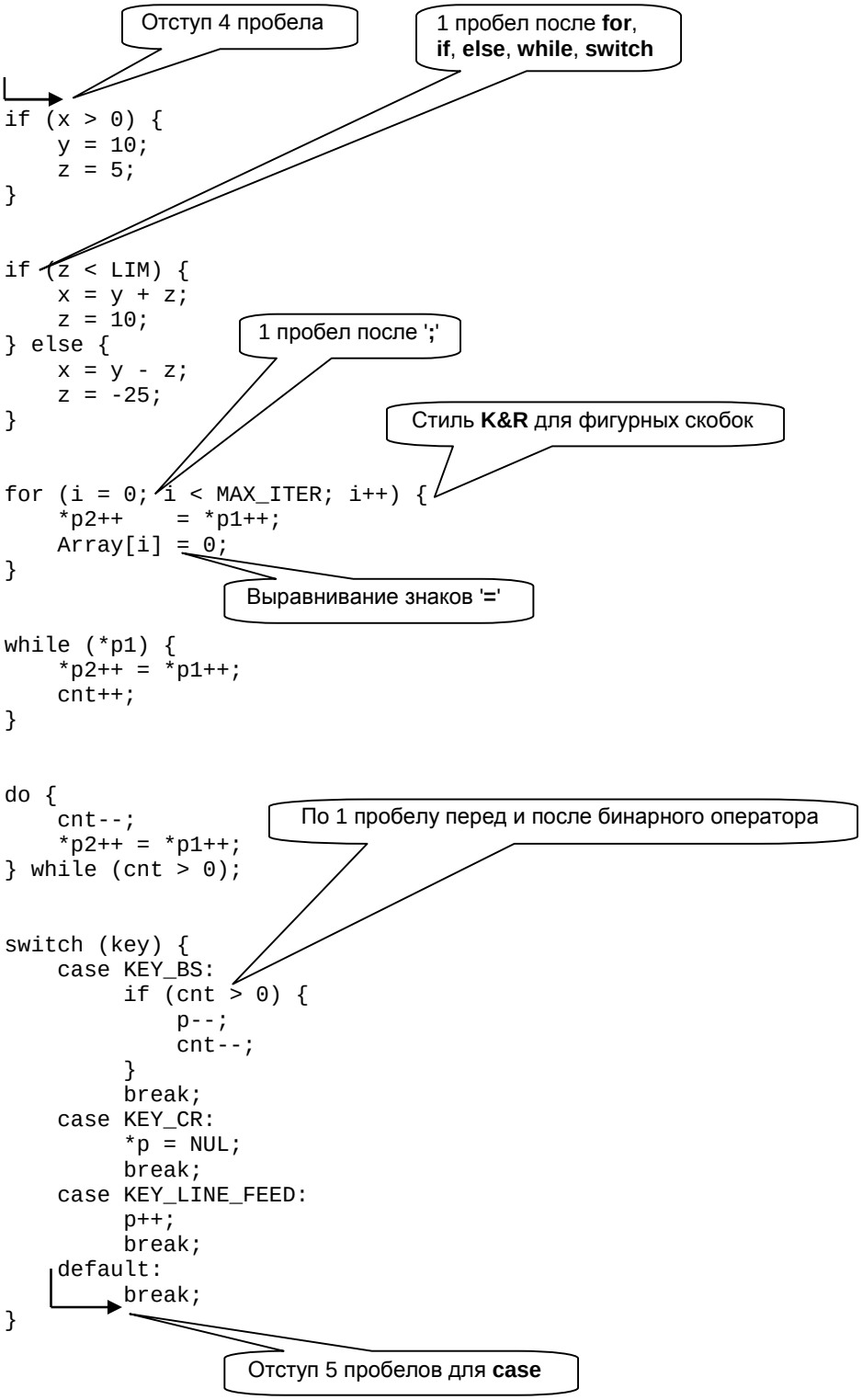
Выражения внутри круглых скобок пишутся без пробелов после открывающейся круглой скобки и без пробелов перед закрывающейся круглой скобкой.

```
x = (a + b) * c;
```

8.00 Конструкции

Должен быть использован следующий стиль конструкций языка.

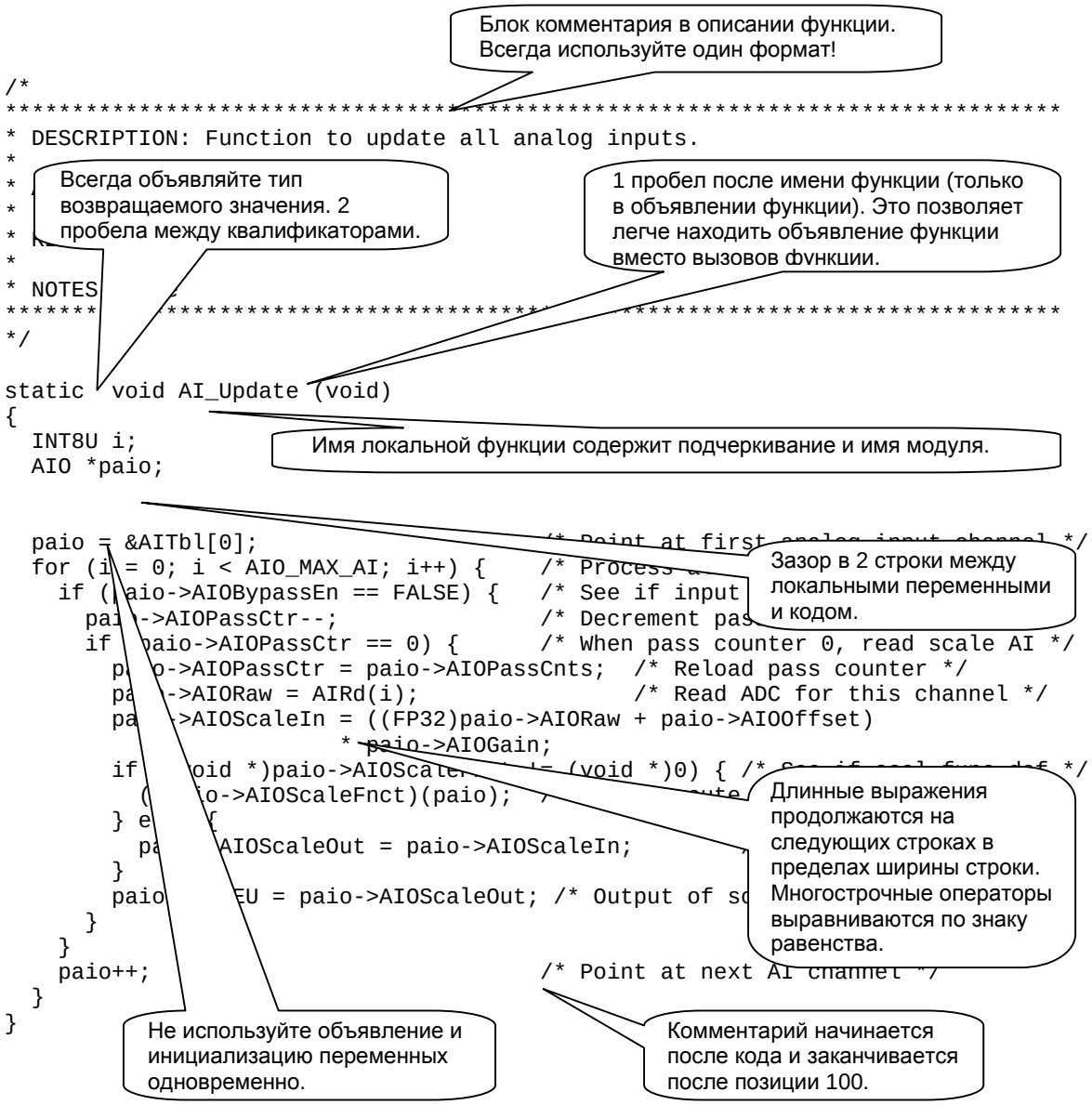
Выравнивание – 4 пробела.
Символы табуляции (TAB) не должны использоваться.
Всегда используйте фигурные скобки, даже для нуль-операторов.
Используйте стиль K&R (Kernighan and Ritchie) для фигурных скобок.




```
if (x > 0) {
    y = 10;
    z = 5;
}


if (z < LIM) {
    x = y + z;
    z = 10;
} else {
    x = y - z;
    z = -25;
}


for (i = 0; i < MAX_ITER; i++) {
    *p2++    = *p1++;
    Array[i] = 0;
}

while (*p1) {
    *p2++ = *p1++;
    cnt++;
}


do {
    cnt--;
    *p2++ = *p1++;
} while (cnt > 0);


switch (key) {
    case KEY_BS:
         if (cnt > 0) {
             p--;
             cnt--;
         }
         break;
    case KEY_CR:
         *p = NUL;
         break;
    case KEY_LINE_FEED:
         p++;
         break;
    default:
         break;
}
```

9.00 Функции

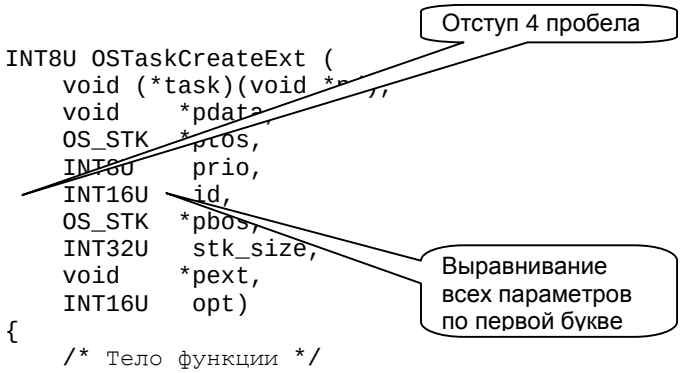
Формат функции должен быть таким как показано ниже.




```
/*
*********************************************************************************************************
* DESCRIPTION: Function to update all analog inputs.
*
*
*
*
* NOTES
*********************************************************************************************************
*/

static  void AI_Update (void)
{
  INT8U i;
  AIO *paio;


  paio = &AITbl[0];                                  /* Point at first analog input channel */
  for (i = 0; i < AIO_MAX_AI; i++) {                 /* Process a
    if (paio->AIOBypassEn == FALSE) {                /* See if input
      pai->AIOPassCtr--;                             /* Decrement pas
      if (paio->AIOPassCtr == 0) {                   /* When pass counter 0, read scale AI */
        p  o->AIOPassCtr = paio->AIOPassCnts;        /* Reload pass counter */
        pa ->AIORaw = AIRd(i);                       /* Read ADC for this channel */
        pa ->AIOScaleIn = ((FP32)paio->AIORaw + paio->AIOOffset)
                        * paio->AIOGain;
        if  oid *)paio->AIOScale      = (void *)0) { /*
          ( io->AIOScaleFnct)(paio);
        } e    {
          pa    AIOScaleOut = paio->AIOScaleIn;
        }
        paio     U = paio->AIOScaleOut; /* Output of sc
      }
    }
    paio++;                                          /* Point at next AI channel */
  }
}
```

Функции с большим числом аргументов.

Если функция имеет много параметров, не имеет смысла перечислять их на одной строке. Вместо этого, объявите функцию как показано ниже.




```
INT8U OSTaskCreateExt (
    void (*task)(void *p
    void    *pdata
    OS_STK  *ptos,
    INT8U    prio,
    INT16U   id,
    OS_STK  *pbos,
    INT32U   stk_size,
    void    *pext,
    INT16U   opt)
{
    /* Тело функции */
```

Одна функция на страницу.

Насколько возможно, на одной странице объявляется только одна функция.

На одной странице можно объявить несколько функций малого размера. Однако все они должны содержать блок комментария с описанием функции. Начало функции должно отделяться двумя-тремя строками после конца предыдущей функции. Функции должны также начинаться на границе страницы.

Функция должна умещаться на одной странице. В очень немногих случаях существует смысл иметь функцию, занимающую несколько страниц. В тех редких случаях, когда это абсолютно необходимо, перевод страницы внутри функции должен производиться в логически обоснованных позициях.

Функции, которые используются только внутри файла, должны быть объявлены как **`static`**, чтобы скрыть их от других функций в других файлах.

10.00 Инициализированные таблицы

Выравнивайте схожие поля в таблице.

Очень часто полезно создавать таблицы, основанные на структурах данных, как показано в примере ниже. Вы должны обратить внимание, что инициализированные поля аккуратно организованы в столбцах. Это делает код легко читаемым. В такой ситуации вы не должны также ограничивать себя шириной столбца, так как не имеет смысла переносить запись таблицы в начало следующей строки по словам.

```
typedef struct {
    INT16U  ParamNbr;
    void   *ParamAddr;
    void   *ParamMin;
    void   *ParamMax;
    void   *ParamDflt;
    void   *ParamInc;
} PARAM;


const PARAM RPM_ParamTbl[] = {
/*- ParamNbr - *ParamAddr --------- *ParamMin --- *ParamMax --- *ParamDflt -----
*ParamInc */
    { 2981, (void *)&RPM,          (void *)&FP0, (void *)&FP0,  (void *)&FP0,
(void *)&FP0},
    { 2982, (void *)&RPMAvg,       (void *)&FP0, (void *)&FP0,  (void *)&FP0,
(void *)&FP0},
    { 2984, (void *)&RPMFltrConst, (void *)&FP0, (void *)&FP1,  (void *)&FP0Pt1,
(void *)&FP0Pt01},
    { 2985, (void *)&RPMTeeth,     (void *)&W4,  (void *)&W500, (void *)&W60,
(void *)&W1},
};


const  UWORD  RPM_ParamTblSize = sizeof(RPM_ParamTbl) / sizeof(PARAM);
```

11.00 Глобальные переменные

Помещайте объявления глобальных переменных только в заголовочный файл.

Как вы знаете, глобальная переменная должна быть размещена в пространстве памяти, доступном другим модулям, используя ключевое слово Си `extern`. Объявления, таким образом, должны быть помещены и в .C и в .H файлы. Однако такое дублирование объявлений может приводить к ошибкам. Методика, описанная ниже, позволяет размещать объявление только в одном месте – .H файле.

Во все .H файлы, которые определяют глобальные переменные, необходимо добавить следующий код:

```
#ifdef   xxx_GLOBALS
#define  xxx_EXT
#else
#define  xxx_EXT extern
#endif
```

Каждая переменная, которая должна быть объявлена глобальной, должна иметь префикс `xxx_EXT` в .H файле как показано ниже. Обозначение `'xxx'` представляет префикс, идентифицирующий имя модуля.

```
xxx_EXT  INT16U  ParamVars;
xxx_EXT  FP32    ParamMaxVal;
```

Си-файл модуля будет содержать следующее объявление:

```
#define xxx_GLOBALS
#include "INCLUDES.H"
```

Модуль, которому принадлежат глобальные переменные, ‘выделит’ память для переменных, в то время как все другие модули, которые просто включают .H файл, будут видеть перед этими переменными квалификатор `extern`.

Ссылки

µC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-57820-103-9

Embedded Systems Building Blocks
Jean J. Labrosse
R&D Technical Books, 2000
ISBN 0-87930-604-1

C Style: Standards and Guidelines
David Straker
Prentice Hall, 1992
ISBN 0-13-116898-3

Контакты

Micriµm, Inc.
949 Crestview Circle
Weston, FL 33327
954-217-2036
954-217-2037 (FAX)
e-mail: Jean.Labrosse@Micrium.com
WEB: http://www.Micrium.com

R&D Books, Inc.
1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
(785) 841-1631
(785) 841-2624 (FAX)
e-mail: rdorders@rdbooks.com
WEB: http://www.rdbooks.com