**GENERAL**

**----------------------------------------------------------------------**

**Используйте 4 пробела вместо клавиши TAB.**

**Используйте пробел вместе со скобками везде, где это возможно.**

if( x ) {

}

**вместо**

if(x){

}

**Используйте пробелы вместе с математическими и логическими операциями.**

val = ( y + x \* ( f – 2 ) ) == 5.2f;

**вместо**

val = ( y+x\* ( f–2 ) )==5.2f;

**Разрешено не использовать пробельные отступы если скобки вложены друг в друга. Если открывающая скобка не имеет справа от себя пробела то и соответствующая ей закрывающая скобка не должна иметь пробела слева. Это правило применимо ко всем скобкам.**

val = (( y \* (( 2 + z ) / 3 ) \* 4 + x) \* ( f – 2 )) == 5.2f;

**Начинайте операторные скобки на той же строке что и конструкция.**

if( x ) {

}

**вместо**

if( x )

{

}

**Используйте операторные скобки везде, где это возможно.**

if( x ) {

a = 3;

}

**вместо**

if( x )

a = 3;

**Используйте пробелы в выражениях с квадратными скобками.**

val = func()->value[ i + j ];

**Разрешено не использовать пробельные отступы если внутри квадратных скобок простое выражение.**

val = array[i];

**При определении указателей используйте значок указателя “\*” рядом с типом указателя и отделяйте его пробелом справа.**

int\* data;

**вместо**

int \*data;

**Используйте суффикс f вместе со значениями типа float.**

float f1 = 0.5f;

float f2 = 1.0f;

**вместо**

float f1 = 0.5;

float f2 = 1.;

**При объявлении параметров функций соблюдайте очерёдность параметров.**

**Первым по приоритету идёт параметр, который относится к какому-либо модулю.**

**Вторыми по приоритету идут параметры для записи в них значений.**

**Третьими по приоритету идут параметры только для чтения.**

int IniReadDataFromFile( ini\_t\* ini, void\* mem, size\_t memSize, const char\* filename );

**Здесь** ini **это осно параметр относящийся к модулю Ini, по этому он идёт первым по приоритету.** mem **и** memSize **это указатель на буфер памяти, в который будут копироваться данные, и размер буфера памяти, они идут вторыми по приоритету.** filename **параметр предназначен только для чтения и он идёт последним по приоритету.**

**Вторыми идут параметры, предназначенные для записи в них данных.**

**NAMES**

**----------------------------------------------------------------------**

**При именовании функций используйте UpperCamelCase.**

void FuncNumberOne( float a, float b );

void Func( float a, float b );

**При именовании рекурсивных функций используйте UpperCamelCase и суффикс \_r.**

void FuncNumberOne\_r( float a, float b );

void Func\_r( float a, float b );

**При именовании переменных используйте lowerCamelCase.**

float angleBetweenVectors;

float angle;

**При именовании глобальных переменных используйте lowerCamelCase и суффикс \_g.**

float angleBetweenVectors\_g;

float angle\_g;

**При именовании структур используйте lowerCamelCase и суффикс \_t.**

struct myStructOne\_t;

**При именовании typedef используйте** **lowerCamelCase и суффикс \_t.**

typedef int myType\_t;

**При именовании typedef указателя на функцию используйте lowerCamelCase с приставкой fn.**

typedef void(\*fnMyFunc)(void);

При именовании перечислений используйте **lowerCamelCase и суффикс \_t. При именовании констант перечислений используйте UPPER\_CASE. Константы перечислений должны начинаться с имени перечисления, и продолжатся названием перечисления.**

enum myFavouriteFruit\_t {

MY\_FAVOURITE\_FRUIT\_APPLE = 1,

MY\_FAVOURITE\_FRUIT\_BANANA,

MY\_FAVOURITE\_FRUIT\_TOMATO

}

**При именовании глобальных констант используйте UPPER\_CASE.**

const int MINOR\_VERSION = 1;

**При использовании констант препроцессора используйте UPPER\_CASE.**

#define MINOR\_VERSION 1

**При определении функций в файлах исходного кода (\*.c) используйте следующий заголовок.**

/\*

================

FunctionName

Description - описание может отстствовать. Но, количество знаков = должно всегда равняться 16-ти

================

\*/

void FunctionName( float f ) {

}

**При объявлении функций в заголовочных файлах (\*.h) используйте следующую конструкцию.**

void FunctionName( float f );

// Ниже всегда идёт описание самой функции

// В переменную f передаётся занчение от -1.5f до 0.75f и т.д.

**При объявлении структур используйте конструкцию совместно с typedef.**

typedef struct {

} myStruct\_t;

**Используйте const совместно с указателями в случаях, если данные используются только для чтения.**

void Func( int\* forWrite, const int\* forRead );

**Используйте конструкцию const следующим образом.**

const int\* p; // указатель на костантные данные

int\* const p; // константный указатель на данные

const int\* const p;// константный указатель на константные данные

**но не используйте**

int const\* p; // по сути это то же самое что и const int\* p,

// но этот вариант использовать нельзя

**MODULE NAMES**

**----------------------------------------------------------------------**

**При именовании функций какого-либо модуля используйте UpperCamelCase и приставку из двух-трёх букв отражающих суть модуля.**

file “sound.h” :

int SndLoadFromFile( snd\_t\* snd, string\_t\* filename );

// Загрузить из файла. В случае успеха возвращает 0

int SndPlayAtPos( const snd\_t\* snd, const vec2\_t\* pos );

// Проиграть звук snd в позиции pos. Функция всегда возвращает 0

**При именовании констант относящихся к модулю, используйте UPPER\_CASE и приставку из двух-трёх букв отражающих суть модуля плюс знак подчёркивания “\_”.**

const int SND\_MINOR\_VARSION = 2;

**FILE NAMES**

**----------------------------------------------------------------------**

**При именовании файлов использовать snake\_case.**

files:

file\_main.c

file\_main.h

main.c

main.h

**Если файлы исходного кода имеют в себе большую реализацию, то целесообразно будет разделить такие файлы на подгруппы. В каждом файле будет реализован свой функционал.**

files:

file\_writer.c

file\_reader.c

file\_memory.c

file.h

**CONTENT** **IN SOURCE CODE FILES (\*.c)**

**----------------------------------------------------------------------**

**Общее содержание файлов исходного кода.**

#include <my\_file.h> // подключение заголовочных файлов

typedef struct myStruct\_t {

int x;

int y;

} myStruct\_t; // определение структур и типов

// используемых только в этом файле

const int CONSTANTS = 1; // определение констант используемых

// только в этом файле

static int crcTable\_g[32] = {0}; // определение глобальных констант

/\*

================

Sum

================

\*/

static int Sum( int a, int b ) {

// реализация статических функций

// статические функции используются только в том

// файле, в котором они реализованы, то есть, эти  
 // функции не определены ни в одном из заголовочных файлов

// Это как приватные функции класса. Эти функции должны

// всегда начинаться с ключевого слова static

return a + b;

}

/\*

================

Function

================

\*/

void Function() {

// реализация функций которые определены в заголовочных файлах

}

**CONTENT** **IN HEADER FILES (\*.h)**

**----------------------------------------------------------------------**

**Общее содержание заголовочных файлов.**

#ifndef \_\_FILE\_NAME\_H\_\_ // обязательная команда для каждого

// заголовочного файла

#define \_\_FILE\_NAME\_H\_\_ // обязательная команда для каждого

// заголовочного файла

#include <my\_file.h> // подключение заголовочных файлов

typedef struct vec2\_t {

float x;

float y;

} vec2\_t; // типы данных, которые будут использоваться

// в других модулях

typedef void(\*fnMyFunc)(int); // типы указателей на функции,

// которые будут использоваться

// в других модулях

const int UNIQUIE\_VALUE = 123; // глобальные константы

int crcTable\_g[32]; // глобальные переменные

void Function( void ); // перечисление функций

// описание функции Function

int Function2( void );

// описание функции Function2

#endif // \_\_FILE\_NAME\_H\_\_ // обязательная команда для каждого

// заголовочного файла

**CORRECT** **CODE EXAMPLES**

**----------------------------------------------------------------------**

**Оператор if.**

if( value == 0 ) {

Func( 1 );

} else if( value & 0x1 ) {

Func( value );

} else {

Func( 0 );

}

if( a == b ) {

c = b;

}

**Оператор for.**

for( i = 1; i < 10; i++ ) {

a \*= b[i];

}

for(;;) {

sleep(1);

draw( DeltaTime() );

if( onKeyExit ) {

break;

}

}

**Оператор while.**

while( doIt == 1 ) {

DoIt( myDoIt );

}

**Оператор do while.**

do {

DoIt( myDoIt );

} while( doIt == 1 );

**Оператор goto. Иногда очень полезно его использовать. Но не разрешается делать зацикливания вместе с оператором goto.**

if( loadFromFile( data, filename ) == ERROR ) {

goto goError;

}

data += 100;

goError:

return ERROR;

**Оператор switch:**

switch( val ) {

case 1: { // вариант со скобками

// Вариант со скобками используется только тогда

// когда здесь нужно создать временную переменную

float f = 1.0f;

}

break;

case 3: // вариант без скобок

// в варианте без скобок создавать переменные нельзя

break;

case 5:

default:

break;

}

**Функция.**

/\*

================

MyFuncName

================

\*/

int MyFuncName( int\* data, int dataSize ) {

int i;

int a = 1;

for( i = 1; i < 10; i++ ) {

if( data[i] == 0 ) {

goto goError;

}

a \*= data[i];

}

if( a == 2 ) {

goto goError;

}

return a;

goError:

return ERROR;

}

**Несколько файлов исходного кода, с практически полным следованием этому соглашению по оформлению кода, можно посмотреть по ссылке: https://github.com/drPetr/ini-file**