Why Apply Coding Standard ?

- Reduce Code Bugs
- Improve Code Readability
- Ease Code Review process

DO NOT!

- Do not use GOTO
- Do not use TAB

General Principles

- Explicit is better than implicit.
- Be consistent.
- It is easier to prevent a bug than to find it and fix it.
- Write as if you are writing for someone else to use and maintain code.

General Principles

- Use C99
- Avoid proprietary compiler language keyword extensions
- Avoid complicated statements
- Use 4 spaces per indent level

What : Line Width

How?

All lines must be limited to 80 characters.

Why?

Code print-outs must be free from distracting line wraps and missing characters during code review process.

What : Indentation

How?

Indent level is 4 spaces

Why?

Greatly improves readability

What : Braces

How?

Braces must surround each code block, even single line blocks and empty blocks.

Why?

This prevents bugs when near by code is changed or commented out

What : &&, ||

How?

Unless it is a single identifier each operand of logical AND and logical OR shall be surrounded by parentheses.

Why?

Do not depend on C operator precedence rules, those who maintain the code in the future might miss this.

SAFE

```
if (itr > 9)
{
    state = END;
}
```

RISKY

```
if (itr > 9) state = END;
```

SAFE

```
if ((len > 0) && (itr < MAX))
{
      ...do something
}
```

RISKY

```
if (len > 0 && itr < MAX)
{
      ...do something
}
```

DO

```
size_t i;
for (i = 0; i < 9; ++i)
{

}
```

DO NOT

```
size_t i;
for (i = 0; i < 9; ++i){

}
```

DO

```
size_t itr;
for (itr = 0; itr < 9; ++itr)
{

}
```

DO NOT

```
size_t i;
for(itr = 0; itr < 9; ++itr){

}
```

DO

```
uint8_t num;
num = 9 + 7;
```

DO NOT

```
uint8_t num;
num = 9+7;
```

DO

```
#ifdef USE_CRC32
#    define MUL_SIZE   152
#else
#    define MUL_SIZE   254
#endif
```

DO NOT

```
#ifdef USE_CRC32
#define MUL_SIZE   152
#else
#define MUL_SIZE   254
#endif
```

DO

```
#ifdef USE_CRC32
#    define MUL_SIZE   152
#else
#    define MUL_SIZE   254
#endif
```

DO NOT

```
#ifdef USE_CRC32
#define MUL_SIZE   152
#else
#define MUL_SIZE   254
#endif
```

DO

```
typedef struct
{
    uint8_t   buff[MAX_SZE];
    uint8_t   checksum;
} name_t;
```

DO NOT

```
typedef struct
{
    uint8_t  buff[MAX_SZE];
    uint8_t   checksum;
} name_t;
```

DO

```
uint8_t find_shape(uint8_t val)
{
    switch(val)
    {
        case RECT:
            ...do something
        break;

        case TRIA:
            ...do something
        break;

        default:
            ...do something
        break;

    }
}
```

DO NOT

```
uint8_t find_shape(uint8_t val)
{
    switch(val)
    {
        case RECT:
        ...do something
        break;

        case TRIA:
        ...do something
        break;

        default:
        ...do something
        break;

    }
}
```

DO

```
inline int max(int num1, int num2)
```

DO NOT

```
#define MAX(A, B) ((A) > (B) ? (A) : (B))
```

SAFE

```
char * x;
char y;
```

RISKY

```
char * x, y;
```

SAFE

```
if (NULL == count)
{
    return true;
}
```

RISKY

```
if (count == NULL)
{
    return true;
}
```

What : static

How?

‘static’ should be used to declare all variables and function that are unused outside of the modules in which they are declared

Why?

This reduces bugs

What : volatile

How?

‘volatile’ should be used to declare global variables accessible by interrupt service routines

Why?

This reduces bugs

What : volatile

How?

‘volatile’ should be used to declare pointer to a memory-mapped I/O peripheral register set

Why?

This reduces bugs

What : volatile

How?

‘volatile’ should be used to declare a global variable accessible by multiple threads

Why?

This reduces bugs

What : volatile

How?

‘volatile’ should be used to declare delay loop counters

Why?

This reduces bugs

What : const

How?

‘const’ should be used to declare variables that should not change after initialization

Why?

This reduces bugs

What : const

How?

‘const’ should be used as an alternate to #define for numeric constants

Why?

This reduces bugs

What : Comment markers

How?

WARNING: Risk in changing block of code
TODO: Area of code still under construction
NOTE: Descriptive comment about why

Why?

Improves code maintainability

What : if, while, for, switch, and return

How?

Shall be followed by one space when there is additional program text on the same line

Why?

Improves code readability

What : =, +=, -=, *=, /=, %=, &=, |=, ^=, ~=, and !=

How?

Assignment operators shall always be preceded and followed by one space

Why?

Improves code readability

What : +, -, *, /, %, <, <=, >, >=, ==,!=, <<, >>, &, |, ^, &&, and ||

How?

Binary operators shall always be preceded and followed by one space

Why?

Improves code readability

What : +, -, ++, --, ! , and ~,

How?

Unary operators shall be written without a space on the operand side

Why?

For functionality as well as improves code readability

What : Function parameters

How?

Each comma separating function parameters shall always be followed by one space

Why?

Improves code readability

What : for loop

How?

Each semicolon separating the elements of a for statement shall always be followed by one space.

Why?

Improves code readability

What : Statements

How?

Each semicolon shall follow the statement it terminates without a preceding space.

Why?

Improves code readability

What : Statements

How?

No line should contain more than one statement

Why?

Reduces bugs

What : Naming

How?

Module names shall consist entirely of lowercase letters, numbers, and underscores. No spaces.

Why?

Reduces bugs

What : Variable Naming

How?

No variable name should be longer than 31 characters or shorter than 3 characters.

Why?

Reduces bugs

What : Variable Naming

Variable type	***Starting characters***
Global variable	*g_*
Pointer variable	*p_*
Pointer-to-pointer variable	*pp_*
Boolean variable	*b-*

Popularly accepted abbreviations

Pt.1

Term	***Abbreviation***
Minimum	*min*
Manager	*mgr*
Maximum	*max*
Mailbox	*mbox*
Interrupt Service Routine	*isr*
Initialize	*init*
Input/output	*io*
Handle	*h_*
Error	*err*

Popularly accepted abbreviations Pt.2

Term	***Abbreviation***
global	*g_*
current	*curr*
configuration	*cfg*
buffer	*buf*
average	*avg*
millisecond	*msec*
message	*msg*
nanosecond	*nsec*
number	*num*

Popularly accepted abbreviations

Pt.3

Term	***Abbreviation***
transmit	*tx*
receive	*rx*
temperature	*temp*
temporary	*tmp*
synchronize	*sync*
string	*str*
register	*reg*
previous	*prev*
priority	*prio*