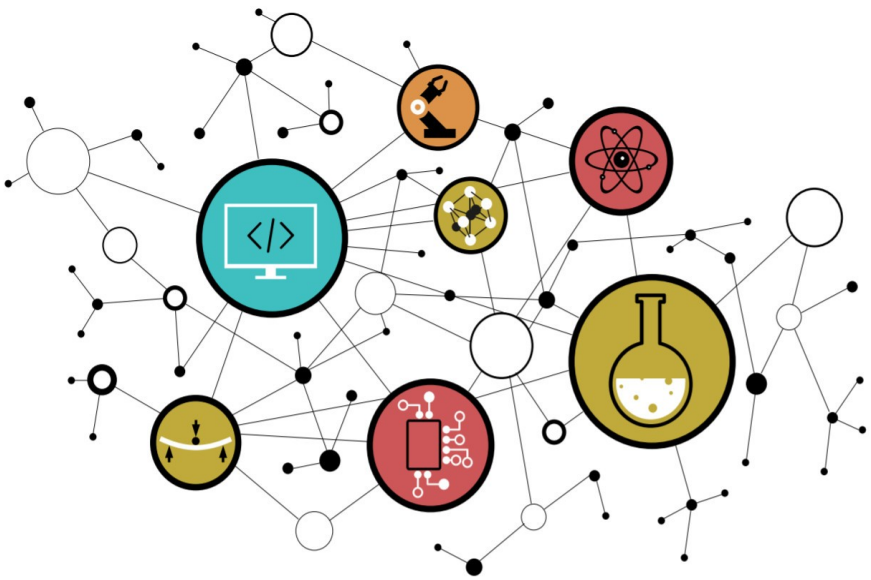
CODING STYLE



Ce document a pour objectif de fixer un cadre et des règles de codage en langage C pour les travaux en Systèmes Embarqués à l'ENSICAEN. L'objectif étant de standardiser, clarifier, cadrer et faciliter le partage de codes sources au sein d'une équipe de développement. Les règles de codage proposées, même si très fortement allégées, sont inspirées du ''Linux Kernel Coding Style'' et du ''GNU Coding Standard''

INDENTATION

```
uInt8_t function_demo ( uInt8_t x_demo, uInt8_t y_demo )
{
    /* all variables declarations on beginning of function */
    uInt8_t z_demo ;

----do {
--------if ( x_demo == y_demo) {
------------/* TODO */
        } else if ( x_demo > y_demo) {
            /* TODO */
        } else {
            /* TODO */
        }

        /* WARNING - single statement */
        if (z_demo != 12) {
            /* Only one C instruction*/

        } else {
            /* Only one C instruction*/
        }

    } while (condition);

    /* TODO */

    return NULL;
}
```

- Structures de contrôle : ouvrir l'accolade sur la même ligne que la structure de contrôle (if, else, for, etc). Fermer l'accolade sur la même colonne que le caractère de début de la structure ou de la fonction.

- Fonctions : cas spécial pour le fonctions, ouvrir l'accolade au début de la ligne suivante

- Tabulation : 4 espaces (à régler dans les préférences de l'éditeur de texte)

```
switch (result_unlock) {
default:
    /* TODO */
    break;
case 1:
    /* TODO */
    break;
case 2:
    /* TODO */
    break;
}
```

- Nombres de niveaux d'indentation : Éviter plus de 4 niveaux d'indentation imbriqués (lisibilité du code)

DÉNOMINATION

```
uInt32_t goku_is_back ;

uInt32_t lm4567_codec_init ( void )
{
    uInt32_t stack_limit ;


     /* TODO */
}
```

- Fonctions : Toujours donner un nom explicite en utilisant un séparateur un ''_'' (ou majuscule)
- Variables locales : Toujours donner un nom explicite en utilisant un séparateur un ''_'' (ou majuscule). Les variables servant de compteur de boucle peuvent avoir un nom simple, par exemple ''i, j, k''. Déclarez vos variables locales en début de fonction (portabilité C89/CANSI)
- Variables globales : Toujours donner un nom explicite en utilisant un séparateur un ''_'' (ou majuscule). Utiliser les variables globales qu'en cas d'absolue nécessité . Il s'agit de ressources partagées à manipuler et protéger avec précaution (section critiques, mutex, etc)
- Définition de type : Toujours définir un nom permettant d'identifier clairement le type de la variable déclarée et le suffixer par ''_t''

```
mugiwara_power_descriptor_t luffy ; /* BIEN */

mpow_desc_t luffy ;  /* PAS BIEN */
```

- Macros : Toujours en Majuscule. Utiliser comme séparateur un ''_'' pour les noms explicites. Pour les macros fonctions, appliquer les même règles de dénomination que pour les fonctions classiques

```
#define UART_MODULE_BAUDRATE 0x7777
#define multiply_fast(x,y)  x*y
```

- Valeur de retour : si la fonction est adaptée à ce qui suit, essayer de faire en sorte que la valeur de retour d'une fonction soit représentative de la validité de son bon fonctionnement. Par exemple, retourne zéro (ou pointeur nul) en cas d'erreur et différent de zéro avec une valeur représentative en cas de succès (1, 2, etc). Pour ces fonctions, les résultats seront retournés par pointeur via les paramètres d'entrée

COMMENTAIRES

- Minimiser l'usage de commentaires dans le code. Un code dit propre, avec des noms d'interfaces (données et API) bien choisies se suffit à lui-même. Cependant, si l'ajout d'un commentaire peut éviter à un autre développeur un temps de recherche pour comprendre une section de code (besoin d'une lecture de la datasheet, mode fonctionnement hérité d'une ancienne version du produit, section de code critique, manipulation d'une ressource partagée critique, etc), alors mettre un commentaire. Toujours écrire son code pour qu'il puisse être facilement relu par un autre ingénieur.

- C89 style : de préférence /* comment */ sinon //comment (C99 style)