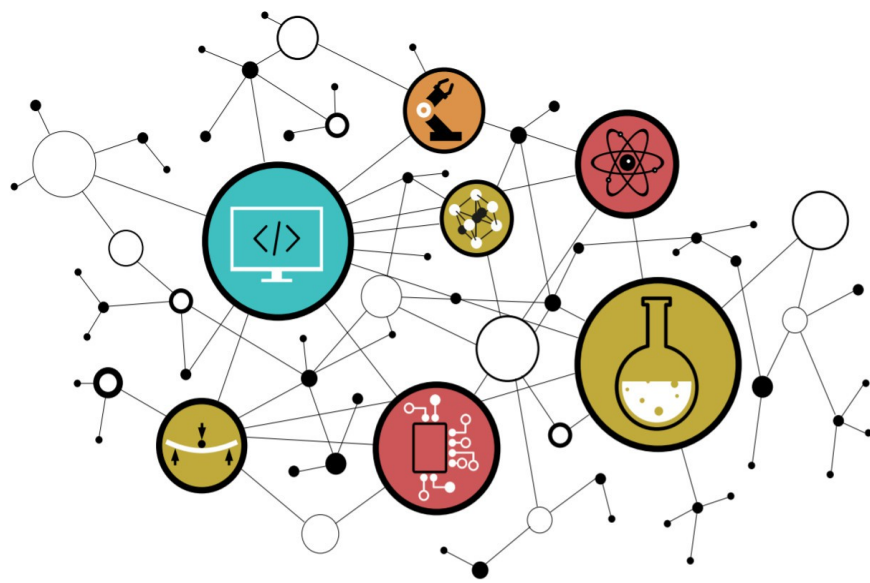


CODING STYLE



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'ENSI CAEN. L'objectif étant de standardiser, clarifier, cadrer et faciliter le partage de codes sources au sein d'une équipe, d'un projet voir d'une entreprise. Les règles de codage imposées sont inspirées du "Linux Kernel Coding Style" et du "GNU Coding Standard". Voici le sommaire du document :

1. Indentation
2. Dénomination
3. Commentaire
4. Divers

INDENTATION

- **Tabulation** : 4 ou 8 caractères (à régler dans les préférences de l'éditeur de texte)

```
switch (data) {  
case 'A':  
case 'B':  
    /* comment */  
    data <=& 20;  
    break;  
default:  
    break;  
}
```

- **Nombres de niveaux d'indentation** : Éviter plus de 3 niveaux d'indentation imbriqués. Facilite la lisibilité du code
- **Nombres de colonnes par page** : 80 caractères/colonnes par page (à régler dans les préférences de l'éditeur de texte)
- **procédures** : cas spécial pour les fonctions, ouvrir l'accolade au début de la ligne suivante

```
int function (int x, int y)  
{  
    int z ;  
  
    ...  
    return 0;  
}
```

- **Structures de contrôle** : règles spécifiant l'indentation ainsi que les espaces à respecter pour l'écriture de structures de contrôle

```
do {
    if (a == b) {
        ...
    } else if (a > b) {
        ...
    } else {
        ...
    }

    /* single statement */
    if (condition)
        action1();
    else
        action2();

} while (condition);
```

DENOMINATION

- **Fonctions** : Toujours débiter par une minuscule. Même si cela est à éviter, pour donner un nom complexe à un objet, utiliser comme séparateur un "_"
- **Variables locales** : Toujours débiter par une minuscule. Les variables servant de compteur de boucle peuvent avoir un nom sans sens précis, par exemple "i, j, k". Sinon, toujours donner à une variable un nom court permettant d'identifier clairement son rôle. Déclarez vos variables locales en début de fonction (portabilité C89/CANSI)
- **Variables globales** : Toujours débiter par une minuscule. Toujours donner à une variable globale un nom permettant d'identifier clairement son rôle (séparateur "_"). Éviter tant que faire se peut les variables globales (ressources partagées à manipuler et protéger avec précaution)

```
float temp_ctrl = 0;

void lm4567_codec_init (void)
{
    temp_ctrl++;
    ...
}
```

- **Définition de type** : Toujours définir un nom permettant d'identifier clairement le type des variables déclarés

```
task_struct foo;    // Bien !
tstruct foo;        // Pas Bien !
```

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

```
#define UART_BAUDRATE 0xFFFF
#define mul(x, y)    x * y
```

COMMENTAIRES

- *Minimiser l'usage de commentaires dans le code. Un code bien écrit, avec des noms d'interfaces bien choisies se suffit à lui-même. Une lecture de code s'adresse avant tout à un ingénieur développeur. Penser sinon à expliquer ce que votre code fait et non comment il le fait !*
- **C99 style** : A éviter "//"
- **C89 style** : A préférer "/* ... */" (solution portable)
- **Balise pour la documentation de code (exemple de Doxygen)** : Insérer dans vos cartouches quelques tags standards, par exemple ceux de Doxygen. Attention, seuls les fichiers d'en-tête doivent contenir des balises pour la génération de documentation. Doxygen permet la génération automatique de documentation vers différents formats (HTML, PDF, CHM ...). Ne pas utiliser de caractères accentués dans vos commentaires

```
/**
 * @file example.h
 * @brief demo header file
 * @author your_name
 */
int temp_conv;

/**
 * @struct str_payload_buffer
 * @brief latest converted datas from temperature sensor
 */
typedef struct {
    int size
    int* conv_tab
} str_payload_buffer;

/**
 * @brief read data codec LM4567 controller
 * @param reset reset LM4567 controller
 * @param conv converted value
 * @return null if data conversion error
 */
handle_spi lm4567_codec_read (char reset, float conv);
```

- Quelques tags supplémentaires (cf. www.doxygen.org):

```
/**
 * @example example insertion
 * @warning warning insertion
 * @li bullet point insertion
 * @mainpage main page comments insertion
 * @image picture insertion
 * @include code insertion
 */
```

DIVERS

- **Valeur de retour** : Essayer de faire en sorte que la valeur de retour d'une fonction soit représentative de la validité de son bon traitement. Par exemple, retourne zéro (ou pointeur nul) en cas d'échec et différent de zéro en cas de succès. Les résultats des procédures seront retournés par pointeur via les paramètres d'entrée