La qualité du logiciel embarqué : Règles de codage C-ANSI

EISTI

Octobre 2016

Ali Baktash

Déroulement de la Formation

- La problématique du Logiciel embarqué
- Les différents Cycle de développement logiciels
- La qualité au rendez-vous de chaque étape
- Rentrons dans les détails, MISRA

Règles de codages C-ANSI

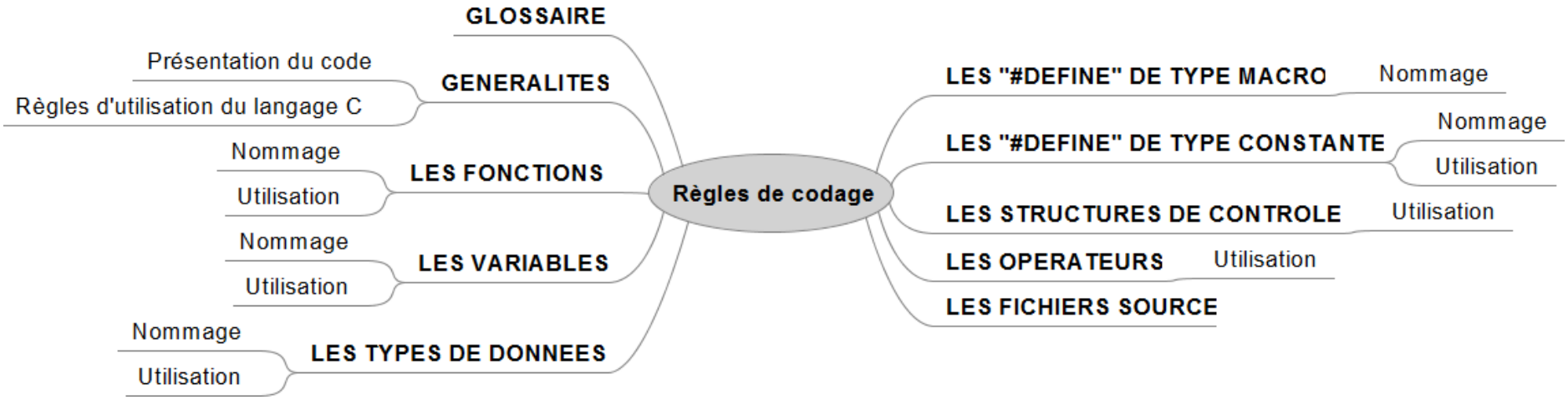
- Modèles de processus SPICE et CMMI
- AUTOSAR

Règles de codage

EISTI

Règles de codage en langage C

Auteur : XX
Date : XX
Version : 1.0
Référence : XX


Règles de codage
GLOSSAIRE
GENERALITES
Présentation du code
Règles d'utilisation du langage C
LES FONCTIONS
Nommage
Utilisation
LES VARIABLES
Nommage
Utilisation
LES TYPES DE DONNEES
Nommage
Utilisation
LES "#DEFINE" DE TYPE MACRO
Nommage
LES "#DEFINE" DE TYPE CONSTANTE
Nommage
Utilisation
LES STRUCTURES DE CONTROLE
Utilisation
LES OPERATEURS
Utilisation
LES FICHIERS SOURCE

Glossaire

- Fonction…
  - …exportée
  - …non exportée
- Variable…
  - …locale
  - …globale
  - …exportée
- Type de données…
  - …non exporté
  - …exporté
  - …commun
- Constantes
- Symboles
- Variables constantes

Généralités

- Indentation du code
- Lisibilité du code
  - Une ligne par instruction
  - Espace avant et après les opérateurs
  - Deux identifiants ne doivent pas différés que par leur casse
  - La base octale est interdite
- Règles d'utilisation du C
  - Mots interdits
    - **goto**
    - **continue**

Les fonctions

- Une seule instruction **return**
- **void** obligatoire (pas de vide laissé)
- Aucune différence n'est permise entre la déclaration et la définition. Les prototypes doivent être identiques.

Les variables

- Variables locales
  - Une fonction ne doit pas renvoyer l'adresse d'une de ses variables locales.
  - Une variable locale ne peut être déclarée qu'en début de fonction.
  - Les variables locales de type **static** ne doivent pas être utilisées.
- Toutes les variables déclarées dans le logiciel doivent être utilisées.
- Les variables *globales* et *exportées* doivent toutes être initialisées dans une fonction d'initialisation du module.
- Seul l'accès par nom de champ est autorisé pour accéder au contenu d'une structure et d'une union.
- Tout accès en écriture à un paramètre d'une fonction est interdit.

Les types de données

- Les types de données natifs du C (char, int, double, long, etc…) ne doivent pas être utilisés. Il est nécessaire, pour des raison de portage et de lisibilité, d'utiliser des types redéfinis tels que « tU8 », « tU16 », « tS16 », etc…
- Pour utiliser un champ de bits, toujours définir un type qui est une union entre le champ de bits proprement dit et l'entier regroupant tous les bits.
  - Ce second champ doit servir dans deux cas seulement :
    - a) initialisation du champ (seule valeur acceptée : 0)
    - b) comparaison par rapport à 0.
  - L'entier qui sert de support au champ de bits doit être non signé.
  - **Attention :** l'ordre de déclaration des champs de bits dépend du compilateur. Il est donc nécessaire de vérifier l'ordre utilisé afin de s'assurer que le bit identifié u1Bit0 sera bien le bit de poids faible et non le bit de poids fort, par exemple.
- Toujours définir un type booléen (tBool). Dans tout le code, faire abstraction du mode de représentation du type booléen, mais n'utiliser que les constantes B_TRUE et B_FALSE.
- N'appliquer ni l'opérateur « ! » ni l'opérateur « ~ » à une donnée de type booléen.

Les « #define » de type macros

- Pas de règle d’utilisation particulière

Les « #define » de type constantes

- La directive #**ifndef** ne doit pas être utilisée.
- La directive #**endif** doit être suivie d'un commentaire rappelant la constante testée.
- Lorsque le bloc d'instructions est important, il est recommandé de faire la même chose après le #**else**.

Les structures de contrôle

- Toutes les expressions **if / else if** doivent avoir une clause **else**, même vide.
- Toujours inclure une clause **default** (documentée) dans une expression **switch / case**. Le code de chaque **case** doit être entre accolades.
- L'instruction **break** ne doit servir que dans une expression **switch / case**. Elle est interdite pour sortir d'une boucle.
- Si la condition de sortie d'une boucle est d'atteindre un maximum ou un minimum, préférer les comparaisons >= et <= au lieu de ==.
- Dans une expression conditionnelle, entourer de parenthèses chaque comparaison.
- Les comparaisons implicites sont interdites.

Les opérateurs

- N’utiliser les opérateurs « ++ » et « -- » que de manière post-incrémentale (décrémentale), et non pré-incrémentale (décrémentale).
- Ne pas oublier que l’opérateur de modulo « % » est associé à une division. Par conséquent, s’assurer que le diviseur n’est pas égal à 0.
- Les opérations bit par bit ('&', '|', '~', '^', '<<' et '>>') sont interdites sur les données signées.
- En employant un opérateur de décalage binaire « << » ou « >> » sur une donnée de longueur N bits, ne pas décaler de plus de N bits.

Les fichiers sources

- extensions « .c » et « .h » uniquement
- Si le module logiciel défini par le fichier principal **module.c** comporte des objets exportés (fonctions, variables, types de données, macros ou constantes), alors il doit exister un fichier d'en-tête **module.h** qui regroupe :
  - les déclarations de toutes les fonctions exportées, sans le mot-clé **extern** ;
  - les déclarations de toutes les variables exportées, avec le mot-clé **extern** ;
  - les définitions de tous les types de données exportés ;
  - les définitions de toutes les macros exportées ;
  - les définitions de toutes les constantes exportées.

  - Afin d'assurer la cohérence entre déclarations et définitions, le fichier **module.c** doit inclure (directive #include) le fichier **module.h**.

Merci de votre attention…

Avez-vous des questions?

Contact:
ali.baktash@smile.fr
ali.baktash@autoliv.com

Présentation inspirée de documents rédigés par:
Christophe Brunschweiler, Smile