Convention de codage en programmation C/C++

1.1 Objectif

Le but de ce document est de décrire les principes de codage à utiliser lorsque vous codez en C ou en C++. **Ces principes sont à appliquer pour chaque projet.** Les sections grisées sont recommandées mais non imposées.

La langue utilisée pour le codage de l'application et des commentaires est définie lors de la séance de démarrage d'un projet. Sinon, par défaut, la langue est l'anglais comme spécifié dans la suite du document

1.2 Domaine d'application

Ce document est valable pour les langages C et C++.

2. Qu'est-ce qu'une convention de codage ?

Une convention de codage est une manière de programmer qui permet de simplifier l'écriture des programmes, d'améliorer leur lisibilité et de rendre plus aisée la reprise d'un projet par une personne externe. L'expérience tend à montrer que les quelques inconvénients rencontrés par l'application de ces conventions ne sont rien par rapport au gain d'efficacité et au temps économisé lors de la reprise des projets. Cette convention est inspirée d'autres conventions existantes.

3. Organisation des répertoires de projets

Bien que non imposée, la structure des répertoires du projet conseillée est la suivante :

.\bin	Fichiers exécutables
.\src	Fichiers sources (*.cpp, *.c)
.\include	Fichiers d'entêtes (*.hpp, *.h)
.\doc	Documentation relative au projet
.\lib	Librairies utilisées par le projet
.\res	Ressources du projet (images, icônes, …)

4. Langue de programmation

Le code et les commentaires de tous les projets doivent être **en anglais** ! Les noms des variables, des fonctions, etc., sont tous définis **en anglais** !

5. Conventions de nommage des différentes entités du code

5.1 Classes et structures

Le nom d'une classe ou d'une structure doit toujours commencer par une majuscule. S'il est composé de plusieurs mots, chaque mot commence également par une majuscule. Le reste des lettres qui composent chaque mot est écrit en minuscules (PascalCase).

Exemple:

```
class MyClass : public AnotherClassComposedOfSixWords
{…}
struct MyStruct
{…};
```

Si une nomenclature est définie pour un projet, elle doit être utilisée tout au long du projet.

Il est possible de préfixer les classes par un identificateur propre au projet. Ce préfixe doit être défini une fois pour toute dans la description du projet.

Exemple:

```
class VSMyClass
{…};
```

VS pour : projet **V**ision **S**téréo

5.2 Fonctions et Méthodes

Pour les noms de fonctions ou méthodes, chaque mot (sauf le premier) commence par une majuscule suivie de minuscules (camelCase). Les fonctions d'accès aux variables internes « protected » ou « private » d'une classe se font en utilisant le préfixe "get" ou "set". Les fonctions d'initialisation commencent par "init" et les fonctions retournant des booléens commencent par "is". Exemple : *initMotorAxis(…), isLightOn()*

Exemple:

```
int getUserChoice()
{…}

void MyClass::setMyParam(int _param)
{…}

bool MyClass::initMyDatabse()
{…}
```

5.3 Constantes

Les symboles déclarés par la directive `#define` doivent être écrits entièrement en majuscules et s'ils sont composés de plusieurs mots, on sépare ces mots par *underscore:* _

Exemple:

```
#define MYCONST 1
#define NUMBER_OF_AXES 3
```

5.4 Variables membres d'une classe

Les variables membres d'une classe ne sont pas distinguées particulièrement, mais on veillera à choisir des noms explicites qui "parlent" à tout le monde.
Il est permis d'utiliser le préfixe m_ (à décider au début du projet).

Exemple:

```
Class UserAccount
{
  private:
        string login;
        string password;
};
```

5.5 Variables globales

Les variables déclarées globalement doivent être préfixées de « g_ ».

5.6 Ressources

Les préfixes suivants PEUVENT être ajoutés aux noms des ressources :

Préfixes	Contrôle visuel
btn	Bouton
fld	Champs (field)
lb	Label
bmp	Image (bitmap)
sc	Scrollbar
rb	Bouton radio
chk	Case à cocher (checkbox)
frm	Formulaire
cb	Combobox

6. Indentation

L'indentation est un principe fondamental de la programmation et n'est pas uniquement imposée par cette convention. Son caractère obligatoire ne fait ici aucun doute.

Les accolades sont en principe déclarées sous l'instruction précédant le bloc (style Allman). On peut utiliser également un autre style d'indentation (K&R, Whitesmith) l'important est d'être cohérent sur l'ensemble.

```
if(axisValue == 2)
{
}
```

La première instruction du bloc est décalée et l'indentation générale est d'un « tab ». Attention il s'agit bien d'un « tab » et pas d'espaces !!!

```
if(myVar == 2)
{
     déclaration ;

     instruction;
     instruction;

     if(optionActive == true)
     {
          instruction ;
     }
}
```

7. Organisation et commentaires de documentation

Sauf bonne raison exposée en commentaire, il n'y aura qu'une classe déclarée par fichier d'entête .h et implémentée par son fichier ".cpp". Ces 2 fichiers porteront le nom de classe entièrement en minuscules.

La classe, déclarée dans son fichier d'entête ".h", sera précédée du commentaire suivant :

```
/**
* Description of the class
*
* @author      Name of the author
* @version     Number of the version
*/
```

Les fonctions, implémentées dans le fichier ".cpp", seront précédées du commentaire suivant :

```
/**
* Name of the function
* Description of the function
*
* @param type param1 : Description of the parameter "param1"
* @param type param2 : Description of the parameter "param2"
* @return description of the returned value
*/
int MyClass::FunctionName(int param1, int param2)
{
 ...
}
```

Des variantes sont admises si le projet impose l'utilisation d'un générateur de documentation (*Doxygen*, …)

8. Dernières remarques (bonnes pratiques)

- Si on utilise un *framework*, il est admis d'adopter les conventions du *framework* plutôt que celles ci-dessus (p. ex : les getters portent le nom des attributs dans Qt sans être préfixés par *get*)
- Prenez l'habitude de commenter votre code lorsque c'est nécessaire (subtilité ou complexité)
- Commentez « en temps réel » et pas tout le projet d'un trait lorsque celui-ci se termine.
- Commentez également les fichiers d'entêtes (".h").
- Un bon commentaire doit être concis, utile (ne répète pas le code), intelligible pour tous (anglais) et nécessaire (met en lumière une subtilité, difficulté du code). Voir par exemple comments.html
- Evitez les répétitions d'un même groupe d'instructions selon le principe DRY (*Don't Repeat Yourself*). Pour cela, factorisez ce groupe en une fonction ou méthode.
- Si une fonction devient trop longue et compliquée, décomposez-la en plusieurs fonctions (KISS).
- N'imbriquez pas trop d'instructions les unes dans les autres ; préférez la lisibilité du code, plutôt que le gain de place.
- Evitez de mettre des valeurs littérales (des chiffres) directement dans les fonctions. Définissez plutôt une constante ou macro à un seul endroit.
- Un bon nom de variable doit être : descriptif, concis, clair. Pour en savoir plus :

  -http://www.makinggoodsoftware.com/2009/05/04/71-tips-for-naming-variables/

- Quelques liens pour ceux désireux d'aller plus loin :

  - https://sourcemaking.com/refactoring/smells (https://fr.wikipedia.org/wiki/Code_smell)
  - https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
  - (https://siderite.blogspot.com/2017/02/solid-principles-plus-dry-yagni-kiss-final.html)