Conventions d'écriture et de codage

v1.0.5.0 – 13/09/2009
peignotc(at)arqendra(dot)net / peignotc(at)gmail(dot)com

Recommandations de nommage

_ De manière générale

Pour tout nom de projet, fichier, classe, variable, fonction, attribut, méthode, etc. :

Nommer intelligemment les éléments de manière représentative ;
Éviter les espaces, les caractères accentués, les caractères spéciaux, les caractères non-anglophones, etc. ;
Utiliser à la place les caractères non-accentués, les caractères underscore (_), point (.), tiret (–).

Ex. : *Projet_liaison-serie.cbp*, *LiaisonSerie.cpp*.

_ En programmation objet (spécification CLS (Common Language Specification))

Pour tout nom de classe, méthode, attribut ou variable constitué de plusieurs mots :

Les mots sont accolés les uns aux autres, sans être séparés par un quelconque caractère ;
À partir du deuxième mot, la première lettre de chaque mot est en majuscule.

Pour tout nom de classe ou méthode :

La première lettre est en majuscule ; à partir du deuxième mot, la première lettre de chaque mot est en majuscule. Nda : il s'agit de la casse PascalCase (casse Pascal – le langage de programmation –).

Pour tout nom d'attribut ou variable :

La première lettre est en minuscule ; à partir du deuxième mot, la première lettre de chaque mot est en majuscule. Nda : il s'agit de la casse camelCase (casse « chameau »).

Pour toute utilisation d'un membre (attribut ou méthode) dans une classe :

Le nom du membre est précisé du préfixe `this` (`this.` ou `this->`).

Ex. : `LiaisonSerie`, `InitialiserLiaison()`, `nbBits`, `nbConnexions`.

_ En programmation C++

Pour tout nom de classe, méthode ou fonction :

La première lettre est en majuscule.

Pour tout nom d'attribut :

La première lettre est en minuscule ;
La première lettre est précédée du caractère _ (underscore).

Pour tout nom de variable :

La première lettre est en minuscule.

Ex. : `LiaisonSerie`, `InitialiserLiaison()`, `_nbBits`, `_nbConnexions`.

_ En programmation C++ sous Microsoft Visual Studio avec les MFC (Microsoft Foundation Classes) [1]

Pour tout nom de classe :

La première lettre est en majuscule ;
La première lettre est précédée du caractère `C` majuscule.

Pour tout nom de méthode ou fonction :

La première lettre est en majuscule.

[1] Notation appelée *notation hongroise* (Hungarian notation (eng)) inventée par un ingénieur de Microsoft, décriée pour son côté explicite trop figé (les attributs portent dans leur nom une information de leur type ; ex : dwCount pour une variable de type mot double (double word)) allant à l'encontre des concepts véhiculés par la POO. Ainsi, il existe certains objets appartenant à des API dont le nom inclut le type mais lequel ne correspond plus au type réel car l'API et donc les classes, les objets, les membres ont évolués.

Pour tout nom d'attribut :

La première lettre est en minuscule ;
La première lettre est précédée du préfixe `m_` (pour « member »).

Pour tout nom de membre statique :

La première lettre est précédée du préfixe `s_` (pour « static »).
Alternative : La première lettre est précédée du préfixe `c_` (pour « constant »).

Ex. : `CLiaisonSerie`, `InitialiserLiaison()`, `m_nbBits`, `s_nbConnexions`.

Conseil :

Choisir l'une ou l'autre des recommandations de style, et s'y conformer scrupuleusement du début à la fin.
Préférer un style généraliste, non marqué par un langage ou un environnement.

Recommandations de mise en forme et de structure

_ De manière générale

Pour toute écriture, syntaxe, mise en forme, structure :

Utiliser les règles de syntaxe et de ponctuation de la langue française (nda : signe de ponctuation simple (`.`, `,` etc.) : pas d'espace avant, 1 espace après / signe de ponctuation double (`;` `:` `=` `!` etc.) : 1 espace avant, 1 espace après / signe de ponctuation inclusif (`()` `[]` `{}` etc.) : 1 espace à l'extérieur, pas d'espace à l'intérieur).

_ Espacement

Pour toute affectation :

Insérer un espace avant et après l'opérateur d'affectation.

Pour tout tableau de variables ou d'attributs (utilisation des crochets `[]`) *:*

Ne pas insérer d'espace après le crochet ouvrant, ni d'espace avant le crochet fermant.

Pour toute évaluation d'expression (utilisation des parenthèses `()`) *:*

Insérer un espace avant et après l'opérateur d'évaluation d'expression.

Pour toute structure de contrôle :

Insérer un espace entre le mot-clef de la structure de contrôle et la parenthèse ouvrante précédant l'expression.

Pour toute structure de contrôle de type « pour… faire… » :

Insérer un espace avant et après les points-virgules séparant l'initialisation, l'expression et la modification.

Pour tout appel ou déclaration de fonction ou méthode :

Ne pas insérer d'espace entre le nom de la fonction ou de la méthode et la parenthèse ouvrante de la liste de paramètres.

Pour toute liste de paramètres d'entrée d'une fonction ou d'une méthode :

Ne pas insérer d'espace après la parenthèse ouvrante, ni d'espace avant la parenthèse fermante ;
Ne pas insérer d'espace avant la virgule de séparation de paramètres, insérer un espace après la virgule de séparation.

Ex. :
```
void EcrireLiaison(char* mots[], int nbMots)
{
   int i;
   for (i = 0 ; i <= nbMots ; i++)
      writeWordRS232(pRS232, mots[i]);
}
```

_ Indentation, ouverture et fermeture d'un bloc

Ouverture d'un bloc :

Sauter une ligne, positionner l'accolade ouvrante `{` au même niveau d'indentation (même colonne) que le début de la ligne précédente et sauter une nouvelle ligne.
Alternative : Positionner l'accolade ouvrante `{` à la suite de la ligne ouvrant le bloc, et sauter une ligne.

Fermeture d'un bloc :

Sauter une ligne, positionner l'accolade fermante } au même niveau d'indentation (même colonne) que le début de la ligne ayant ouvert le bloc, et sauter une nouvelle ligne.

Imbrication des blocs :

Insérer plusieurs espaces (2-4) supplémentaires en début de ligne pour toute ligne d'un bloc imbriqué dans un autre.
Alternative : Insérer une tabulation ; nda : dans les éditeurs des EDI [1], un appui sur la touche de tabulation n'insère pas une vraie tabulation (caractère ASCII 0x09) mais un groupe de plusieurs espaces, ceci afin de favoriser la portabilité avec d'autres éditeurs.

Ex. :
```
int main(void)
{     // début fonction
   int tab[10], i;

   do {    // début do/while
      for (i=0 ; i<10 ; i++)
      {     // début for (1)
         printf("saisir une valeur");
         cin << tab[i];
      }     // fin for (1)
      for (i=0 ; i<10 ; i++) {     // début for (2)
         cout << tab[i] << endl;
      }     // fin for (2)
   } while (true);      // fin do/while
}     // fin fonction main
```

_ Organisation des fichiers source

Pour du code objet :

Un fichier source ne doit contenir qu'une seule classe publique ;
Le fichier source doit avoir exactement le même nom (sans extension) que la classe publique qu'il contient.

Déclaration des membres :

Regrouper les variables, fonctions ou membres en catégories (constructeurs+destructeur, accesseurs, gestion des évènements, …) ;
Dans chaque catégorie, classer les entités par ordre alphabétique.

Ex. : Fichier *LiaisonSerie.cpp*.

```
class LiaisonSerie {
   public:
      LiaisonSerie();
      ~LiaisonSerie();
      bool Configurer(int, int, bool);
      void EcrireLiaison(char* mots[], int);
      void InitialiserLiaison();

   private:
      int debitLiaison;
      int nbBits;
      static int nbConnexions;
      bool paritePaire;
};
```

_ Commentaires (directives classiques)

Pour tout commentaire dans une fonction ou méthode :

Insérer un commentaire mono-ligne avec // (2x slash) avant la partie de code commenté, ou bien à la fin de la ligne de code commentée.

Pour toute définition de classe :

Préciser ce qu'elle représente en 1 ou 2 lignes ;
Préciser quelles sont la/les éventuelle(s) classe(s) de base ;
Préciser succinctement le rôle de chaque membre.

[1] Environnement de Développement Intégré.

Pour toute fonction ou méthode :

Préciser son rôle en 1 ou 2 lignes ;
Préciser le type et le rôle de chaque paramètre d'entrée ou <rien> s'il n'y en a pas ;
Préciser le type et le rôle du code de retour ou <rien> s'il n'y en a pas.

Ex. :
```
//* classe LiaisonSerie
// permet de représenter et manipuler une liaison série
//
// dérive de : <rien>
//
class LiaisonSerie {
   public:
      LiaisonSerie();    // constructeur par défaut
      ~LiaisonSerie();   // destructeur
      bool Configurer(int, int, bool);  // configure la liaison
      void EcrireLiaison(char*[], int); // écrit sur la liaison
      void InitialiserLiaison();   // initialise la liaison

   private:
      int debitLiaison;    // débit de la liaison en bauds
      int nbBits;     // nombre de bits de la trame
      static int nbConnexions;  // nombre total de connexions réalisées
      bool paritePaire;    // parité paire ('true') ou impaire ('false')
};

//* bool Configurer(int debit, int bits, bool parite)
// configure la liaison série avec les valeurs passées en paramètres
//
// entrées :
// - debit (int): débit de la liaison
// - bits (int): nombre de bits
// - parite (bool): parité paire/impaire
// sortie :
// (bool): 'true' si la liaison a été configurée sans erreur, 'false' sinon
//
bool LiaisonSerie::Configurer(int debit, int bits, bool parite)
{
   ...
}
```

_ Commentaires (directives pour documentation XML)

Pour toute méthode :

Utiliser les commentaires XML qui permettent de construire automatiquement la documentation.

Ex. :
```
/// <summary>Configure la liaison série avec les valeurs passées
/// en paramètres </summary>
/// <param name="debit">débit de la liaison</param>
/// <param name="bits">nombre de bits</param>
/// <param name="parite">parité paire ('true') ou impaire ('false')</param>
/// <devdoc>commentaires du développeur...</devdoc>
///
bool LiaisonSerie::Configurer(int debit, int bits, bool parite)
{
   ...
}
```

Bibliographie

Adams Brad, *Design guidelines, managed code and the .net framework*, http://blogs.msdn.com/brada/articles/361363.aspx, 2008 ;

Divers, *Blogs MSDN*, http://blogs.msdn.com/, 2008 ;

Divers, *Hungarian Notation – The good, the bad and the ugly*, http://ootips.org/hungarian-notation.html, 1998 ;

Sharp John, *Visual C# 2005 – étape par étape*, Dunod – Outils du développeur Microsoft, 2006.