Norme de documentation des programmes

Auteur : Marc Frappier

Collaborateurs

Benoit Fraikin
Gabriel Girard
Jean Goulet
Gérard Houdeville
Luc Lavoie

Version : 1.02
30 août 2004

Département d'informatique
Faculté des sciences
Université de Sherbrooke

1. Introduction

Ce document décrit de manière très synthétique des règles concernant la documentation du code source des programmes C++ et Java. Ce document propose aussi des lignes directrices quand à la mise en forme du code source (indentation, espacement, etc).

2. Documentation

1. **Principes généraux**
   a. La documentation s'adresse aux *lecteurs* et aux utilisateurs d'un programme, non pas à son auteur (sauf 6 mois après l'avoir écrit; l'auteur ne se souvient alors plus très bien de ce qu'il a fait).
   b. Elle aide le lecteur à comprendre le code source et à l'utiliser; elle *ne duplique pas* les informations déjà contenues de manière explicite dans le code source.
   c. Elle doit être claire, concise, précise, complète, uniforme et simple à maintenir.
   d. Elle doit être faite au fur et à mesure que le programme est développé, sinon elle ne sera jamais faite (correctement).
   e. Elle facilite la maintenance des programmes dans une organisation en *uniformisant* le style de programmation, ce qui permet à une personne de reprendre plus facilement le travail d'une autre.

2. **Fichier source** : Chaque fichier source comprend une entête constituée des éléments suivants :
   a. le nom du programme (ie, le nom du fichier source)
   b. une courte description (quelques lignes)
   c. les auteurs et leurs matricules;
   d. la date de fin de première réalisation;

e. la date de fin de chaque version avec les auteurs et matricules;
f. le cours pour lequel le fichier est développé;
g. une description plus longue qui contient :
    i. les entrées du programme et leur préconditions;
    ii. les sorties du programme et leur postconditions.
h. un historique des modifications pour chaque version.

3. **Méthode et fonction** : Chaque méthode et chaque fonction :
    a. comprend une entête décrivant le traitement effectué; idéalement, on décrit les préconditions sur les entrées et les postconditions sur les sorties;
    b. comprend, *au besoin*, des commentaires dans le corps qui expliquent les sections plus complexes du traitement effectué;

4. Identificateur

a. Usage d'identificateurs significatifs
    i. Les noms de classes, de variables et de constantes doivent être précis.
    ex : `nbDePrets`, `nbPrets`.
    contre-exemple : `a`, `v`, `nb`, `k`, `cpt`, `flag`, `toto`, `id`, `booleen`, `temp`, `traitement`, `validation`, etc
    ii. Un nom de fonction ou de méthode décrit bien le traitement effectué. On utilise généralement :
        1. un *adjectif* ou une *expression propostionnelle* pour une méthode ou une fonction retournant un booléen .
        ex : `pile.vide()` ou bien `pile.estVide()` ou bien `resteMotALire()`;
        contre-exemple : `pile.vérifierSiVide()`
        2. un *verbe* pour une fonction ou une méthode ayant un effet de bord ou modifiant les propriétés d'un objet.
        ex : `pile.empiler(assiette), calculerCheminPlusCourt(…)`
        contre-exemples : `pile.traitementAjout(assiette)`, `pile.empilee(assiette)`.
    iii. Utilisez des abréviations courantes et largement connues à l'intérieur d'un nom; par exemple :
        1. `nb…` pour nombre de …;
        2. `cpt…` pour compteur de …;
        3. `maj…` pour mise-à-jour de ….
    iv. Les indices dans une itération peuvent être des lettres simples (i, j, …); dans le cas où les indices dénotent un concept particulier (par une matrice à 3 dimensions : pays, province, comté), il est préférable d'utiliser un nom significatif (par ex : `pays`, `province`, `comte`).
    v. Un nom doit avoir un et un seul sens dans un programme. Par exemple, n'utilisez pas `nombre` pour dénoter à la fois un nombre de livres et un nombre de prêts.
    vi. Un concept doit avoir un et un seul nom. Par exemple, n'utilisez pas `…Pret` à un endroit, et `…Emprunt` à l'autre.
b. Construction des identificateurs

i. Un nom de classe commence par une majuscule.
ex : `GestionLivre`
ii. Un nom de méthode, de fonction ou de variable commence par une minuscule. ex : `emprunterLivre, nbDePrets`
iii. Un nom de constante est écrit en majuscules.
ex : `NB_MAXIMAL_PRETS`
iv. On sépare les mots d'un identificateur par des majuscules ou des « _ ».
ex : `emprunterLivre, nbDePrets, emprunter_livre, nb_de_prets`

c. Portée des identificateurs
i. Règle générale, les variables et les fonctions sont toujours déclarées de manière locale.
ii. Les constantes universelles, les constantes communes à plusieurs fichiers sources ou plusieurs fonctions sont déclarées de manière globale.
ex : π, e, unité de conversion métrique-impérial, une constante de votre application comme la taille maximale d'une ligne d'entrée.
iii. Les constantes utilisées dans une seule fonction sont déclarées dans cette fonction.
ex : taille commune de plusieurs vecteurs locaux d'une fonction.

5. Traitement

a. On essaie de limiter au maximum les sentinelles (*flags*). Il est préférable d'utiliser, quand cela est possible, des structures conditionnelles. Si on doit utiliser une sentinelle, il faut lui donner un nom significatif
ex :

Non recommandé	Recommandé

`if (condition)`
`  flag8=true;`
…
`if(flag8) {`
`  énoncés;`
`  }`
	`if(condition) {`
	`  énoncés;`
	`  }`
	ou bien, si on ne peut regrouper en une seule condition,
	`if (condition)`
	`  ressourceLibre = true;`
	…
	`if(ressourceLibre) {`
	`  énoncés;`
	`  }`

b. On utilise directement une variable booléenne dans une condition.
ex : `if (ressourceLibre)` au lieu de `if (ressourceLibre == true)`

c. On évite les effets de bord dans une condition.
ex : `if (a = b) …`, `if (lireMot(fichier,mot))` où `lireMot(fichier,mot)` lit un mot d'un fichier et retourne vrai si un mot a été lu. Il est préférable de faire

```
sauterEspaces(fichier);
if (resteMotALire(fichier))
    {
    mot = lireMot();
    …
    }
```

3. Mise en forme

3.1. Longueur

1. La longueur d'une fonction ou d'une méthode ne dépasse généralement pas 30 lignes, en excluant de ce nombre les lignes blanches et les lignes ne contenant que « { » ou « } ».
2. La longueur d'une ligne ne dépasse généralement pas 80 caractères.

3.2. Indentation

3. Ne pas utiliser les caractères de tabulation pour indenter les programmes, car il s'affichent de manières différentes dépendamment de l'éditeur utilisé et de la plateforme (Unix, Linux, PC, MAC, courriel, etc). Utiliser seulement des espaces.
4. Les énoncés doivent être indentés de manière lisible. Il existe plusieurs formes acceptables. En voici quelques exemples.

Norme SUN	Variante SUN	Norme GNU

`if (condition) {`
`  énoncés;`
`}`
	`if (condition) {`
	`  énoncés;`
	`  }`
		`if (condition)`
		`  {`
		`  énoncés;`
		`  }`

`if (condition) {`
`  énoncés;`
`} else {`
`  énoncés;`
`}`
	`if (condition) {`
	`  énoncés;`
	`  }`
	`else{`
	`  énoncés;`
	`  }`
		`if (condition)`
		`  {`
		`  énoncés;`
		`  }`
		`else`
		`  {`
		`  énoncés;`
		`  }`

`if (condition) {`
`  énoncés;`
`} else if (condition) {`
`  énoncés;`
`} else if (condition) {`
`  énoncés;`
`…`
`} else {`
`  énoncés;`
`}`
	`if (condition) {`
	`  énoncés;`
	`  }`
	`else if (condition){`
	`  énoncés;`
	`  }`
	`else if (condition) {`
	`  énoncés;`
	`…  }`
	`else{`
	`  énoncés;`
	`  }`
		`if (condition)`
		`  {`
		`  énoncés;`
		`  }`
		`else if (condition)`
		`  {`
		`  énoncés;`
		`  }`
		`else if (condition)`
		`  {`
		`  énoncés;`
		`  }`
		`…`
		`else`
		`  {`
		`  énoncés;`
		`  }`

`for (initialisation; condition; incrément) {`
`  énoncés;`
`}`
	`for (initialisation; condition; incrément) {`
	`  énoncés;`
	`  }`
		`for (initialisation; condition; incrément)`
		`  {`
		`  énoncés;`
		`  }`

`while (condition) {`
`  énoncés;`
`}`
	`while (condition) {`
	`  énoncés;`
	`  }`
		`while (condition)`
		`  {`
		`  énoncés;`
		`  }`

Les énoncés imbriqués sont aussi indentés en conséquence.

Norme SUN	Norme GNU
`if (condition)`	`if (condition)`

```
    if (condition) {
        énoncés;
    } else {
        énoncés;
    }
else
    if (condition) {
        énoncés;
    }
```

```
    {
    if (condition)
        {
        énoncés;
        }
    else
        {
        énoncés;
        }
    }
else
    {
    if (condition)
        {
        énoncés;
        }
    }
```

3.3. Règles particulières pour JAVA

1. Déclarer les énoncés `import` en ordre alphabétique.
2. Les commentaires d'entêtes de classe et de méthode doivent utiliser le format javadoc qui permet de générer une documentation avec la commande javadoc (ie, /** */, avec les étiquettes appropriées comme @return, @param, @see, etc); pour plus d'information, voir http://java.sun.com/javadoc/index.html

3.4. Règles particulières pour C++

1. Déclarer les énoncés `#include` en ordre alphabétique.
2. Les commentaires d'entêtes de classe et de méthode doivent utiliser le format javadoc qui permet de générer une documentation avec l'outil Doxygen http://www.doxygen.org/
3. Il convient de préciser le mode des paramètres de fonction et de méthode (entrée, sortie, entrée-sortie).
4. Aucune fonction n'est définie avant le main (particulièrement en IFT159, par souci pédagogique).
5. Les fonctions ne sont déclarées que dans la ou les fonctions ou elles sont utilisées (particulièrement en IFT159, pour imposer le respect de la modularité).
6. On déclare les paramètres d'une fonction ou d'une méthode de la manière suivante : les paramètres par valeur en premier, les paramètres par référence en second.
7. Les passages par référence ne sont utilisés que lorsqu'une valeur de sortie est transmise par effet de bord (particulièrement en IFT159, par souci pédagogique).

4. Glossaire

1. **Entrée** : les éléments suivants sont qualifiés de données d'entrée : un *paramètre* d'un programme, d'une fonction ou d'une méthode; une *variable globale* utilisée par un programme, une fonction ou une méthode.

2. **Effet de bord** : modification d'une variable globale par un programme, une fonction ou une méthode.

3. **Précondition** : une condition qui doit être satisfaite par les entrées d'un programme, avant son exécution. Si un programme est appelé avec des entrées qui ne satisfont pas la précondition, le programme n'a aucune obligation de terminer ou d'effectuer un traitement tel que spécifié dans la postcondition.

4. **Programme** : un ensemble d'unités de compilation fonctionnellement reliés. En C++ et en Java, un programme dénote généralement une méthode `main` ainsi que toutes les fonctions ou classes utilisées par `main`, directement ou indirectement.

5. **Postcondition** : une condition qui doit être satisfaite à la fin de l'exécution d'un programme. Elle détermine une relation entre les entrées et les sorties du programme. Si les entrées satisfont la précondition, le programme doit alors terminer en satisfaisant la postcondition.

6. **Sortie** : : les éléments suivants sont qualifiés de données de sortie : un *paramètre par référence* d'un programme, d'une fonction ou d'une méthode; un paramètre de retour d'un programme, d'une fonction ou d'une méthode; une *variable globale* modifiée par un programme, une fonction ou une méthode.

5. Références

1. Sun Microsystems. *Code Conventions for the Java™ Programming Language*, http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
2. Apache. The Jakarta Site – Coding Standards, http://jakarta.apache.org/turbine/common/code-standards.html
3. GNU. Gnu Coding Standards, http://www.gnu.org/prep/standards_toc.html