

Charte de Programmation – Projet C/C++

*Langage C/C++*

Document de travail.

DREYER Quentin

JAMBET Pierre

NGUYEN Michael

Table des matières

[I. Références du document 3](#_Toc262397289)

[II. Objectifs du document 4](#_Toc262397290)

[III. Normes de programmation 5](#_Toc262397291)

[1. Forme générale du code 5](#_Toc262397292)

[1. Langages 5](#_Toc262397293)

[2. Organisation globale des fichiers 5](#_Toc262397294)

[3. Organisation interne des fichiers d’en-tête (.h ou .hpp) 6](#_Toc262397295)

[4. Organisation interne des fichiers sources (.c ou .cpp) 7](#_Toc262397296)

[2. Forme détaillée du code 8](#_Toc262397297)

[1. Règles de nommage 8](#_Toc262397298)

[2. Composition des fonctions 8](#_Toc262397299)

[3. Commentaires 9](#_Toc262397300)

[4. Espaces 9](#_Toc262397304)

[3. Règles d’utilisation 10](#_Toc262397305)

# Références du document

|  |  |
| --- | --- |
| **DOCUMENT** | |
| Nom du fichier | charte.docx |
| Version | 1 |
| Client | Document interne |
| Projet | Réseaux sociaux : Recherche de composantes fortement connexes |
| Auteurs | DREYER Quentin,  JAMBET Pierre,  NGUYEN Michael |
| Date de création | 25/04/2010 |
| Nombre de page |  |

|  |  |  |
| --- | --- | --- |
| GESTION DES VERSIONS | | |
| Version | **Date de Création** | **Auteurs** |
| 1 | 25/04/2010 | NGUYEN Michael |
| 2 | 23/05/2010 | NGUYEN Michael |
|  |  |  |

# Objectifs du document

Ce document décrit les normes adoptées par notre groupe. N’importe qui voulant lire nos codes pourra se référer à ce document pour en comprendre la structure.

Chaque membre de notre groupe se conforme donc aux règles votées unanimement présentes dans ce document.

Les informations qui suivent permettent donc :

* L’homogénéité du code,
* La lisibilité et facilité de relecture du code
* La compréhension mutuelle de nos travaux

# Normes de programmation

## Forme générale du code

### Langages

Pour des raisons de problèmes avec les accentuations, le code (nom des structures, des fonctions et des variables) sera écrit en anglais.

Cependant, pour les personnes désirant simplement avoir une vue globale des algorithmes, nous avons décidé de rédiger tout ce qui est commentaire en français. Les caractères spéciaux comme les cédilles ou les accents ne seront pas pris en compte dans la rédaction.

### Organisation globale des fichiers

Les fichiers seront séparés dans des dossiers bien définis :

* Les fichiers de code source (/src) qui contiendra :
  + Un dossier de header (/headers) contenant tous les fichiers d’en-tête. Ces fichiers devront avoir obligatoirement l’extension .h ou .hpp.
  + Un dossier de code source (/source) contenant tous les fichiers sources correspondant à leurs en-têtes respectives. Ces fichiers auront l’extension .c (pour le langage C) ou .cpp (pour le langage C++).
  + Un fichier nommé *makefile* (aucune extension)
* Les fichiers externes utilisés pour tests (/test)

### Organisation interne des fichiers d’en-tête (.h ou .hpp)

Les fichiers d’en-tête auront tous une organisation similaire, quelque soit leur contenu.

#### En-tête

/\* ===============================================================

PROJET : *Nom du projet*

NOM DU FICHIER : *Nom du fichier .h*

OBJET : *Description brève du fichier*

--------------------------------------------------------------------------------

DATE DE CREATION : *Date de création du fichier*

AUTEUR : *Liste des auteurs du fichier*

--------------------------------------------------------------------------------

DETAILS : *Informations complémentaires (optionnel)*

=============================================================== \*/

*- Ligne vide -*

#### Inclusions

/\*=================================

Includes

===================================\*/

Liste des inclusions de la librairie standard

*- Ligne vide -*

Liste des inclusions des autres modules du programme

*- Ligne vide -*

*- Ligne vide -*

#### Constantes

/\*=================================

Constantes

===================================\*/

Liste des constantes

*- Ligne vide -*

#### Attributs

Que le fichier contienne une structure ou une classe, il y a dans les deux cas des attributs à déclarer. Elles devront l’être avec le format suivant :

/\* ATTRIBUTS \*/

Liste des attributs

*- Ligne vide -*

/\* EXPLICATIONS DES ATTRIBUTS :

Petite ligne décrivant l’utilité de la structure ou de la classe

- Att1 : Description de l’attribut 1

…

- Attn : Description de l’attribut n \*/

*- Ligne vide -*

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

*- Ligne vide -*

#### Fonctions et procédures

/\* PROCEDURE ou FONCTION : *nom* – *description rapide* \*/

*Signature de la fonction ou de la procédure*

/\* COMPLEXITE : *sa complexité*

ENTREE : *les données en entrée*

ALGORITHME : *l’algorithme utilisé* \*/

*- Ligne vide -*

*- Ligne vide -*

### Organisation interne des fichiers sources (.c ou .cpp)

#### En-tête

/\* =================================================================

PROJET : *Nom du projet*

NOM DU FICHIER : *Nom du fichier .c ou .cpp*

OBJET : *Description brève du fichier*

------------------------------------------------------------------

DATE DE CREATION : *Date de création du fichier*

AUTEUR : *Liste des auteurs du fichier*

------------------------------------------------------------------

DETAILS : *Informations complémentaires (optionnel)*

=============================================================== \*/

*- Ligne vide -*

#### Fonctions et procédures

/\* PROCEDURE ou FONCTION : *nom* – *description rapide* \*/

*Corps de la fonction ou de la procédure*

/\* ------------------------------------------------------------ \*/

*- Ligne vide -*

*- Ligne vide -*

## Forme détaillée du code

### Règles de nommage

|  |  |
| --- | --- |
| Elément | Syntaxe à adopter |
| Fichier source | Le nom du fichier sera identique à celui de la classe.  Le fichier contenant le main devra cependant s’appeler main.c ou main.cpp |
| Classe | Le nom de la classe commencera par une majuscule.  S’il est composé de plusieurs mots, tous les mots débuteront par une majuscule et seront collés sans espaces.  Exemple : MyClass |
| Fonction (action) | Le nom de la fonction commencera par une minuscule.  S’il est composé de plusieurs mots, le premier mot débutera par une minuscule, et tous les autres par une majuscule. Tous les mots seront collés sans espaces  Exemple : giveAName |
| Fonction (vérification) | Le nom de la fonction commencera obligatoirement par is ou are.  Les autres mots débuteront par une majuscule et seront collés sans espaces.  Exemple : isAnotherName |
| Constantes | Une constance devra avoir un nom explicite et être écrite tout en majuscule. S’il est composé de plusieurs mots, chaque mot sera séparé par un « \_ ».  Exemple : MY\_CONSTANT |
| Variables (locales) | Si le nom est composé de plusieurs mots, le premier commencera par une minuscule et les autres par une majuscule. Tous les mots seront collés sans espaces.  Exemple : myVariable |
| Variables (Classe) | La syntaxe est identique que pour les variables locales, excepté qu’elle devra débuter par « m\_ ».  Exemple : m\_myVariable |
| Compteurs | Les variables « compteur » seront en général i, j et k.  Si on a besoin d’autres compteurs, celui-ci commencera par « cpt » |
| Pointeurs | Pour les pointeurs, le nom sera identique à celui d’une variable locale, excepté qu’il devra débuter par « p\_ »  Exemple : p\_myPointer |

### Composition des fonctions

Chaque fonction devra suivre l’organisation suivante :

Type\_retour nom(variables) {

Déclaration des variables

*- Ligne vide –*

Initialisation

*- Ligne vide -*

*- Ligne vide -*

Corps principal

*- Ligne vide -*

Retour du résulat (si le type de retour est différent de void)

}

Note : Il est interdit de retourner une valeur dans le corps principal de la fonction.

### Commentaires

/\* Commentaire sur une seule ligne \*/

/\* Commentaire sur

Plusieurs lignes \*/



### Espaces

Il y a plusieurs cas où l’espace est obligatoire :

* + - Après un mot réservé
    - Après une virgule ou un point-virgule
    - Après un cast
    - Dans la condition d’un **while** ou d’un **if**
    - Entre des opérateurs de calcul

for (a; b; c) {

}

while ( condition ) {

}

if ( condition ) {

}

a = (int) b;

Int a, b, c;

a = b + c;

## Règles d’utilisation

Dans un souci de bon fonctionnement, il est recommandé :

* + - De toujours spécifier une fonction avant de l’utiliser. Soit plus haut dans le fichier, soit dans le fichier d’en-tête s’il existe.
    - D’éviter d’utiliser **break** autre que dans un **switch**
    - De toujours initialiser les valeurs

Il est recommandé dans un souci de lisibilité :

* + - D’utiliser les indentations de 4 : Un bloc inclus dans un autre devra être indenté convenablement

for (a; b; c) {

if ( condition ) {

while ( condition ) {

/\* code \*/

}

}

}

* + - De respecter les lignes vides indiquées dans cette charte
    - D’éviter au maximum les affectations multiples

a = b = c;

a++ = b++;

* + - D’éviter les opérateurs ternaires