Présentation des programmes en C

1 Introduction

Il est indispensable de présenter une certaine rigueur dans la manière d'écrire un programme en respectant des règles de style, particulièrement en C:

- dans un cadre industriel, afin de simplifier la communication et d'accroître la productivité;

- dans un cadre scolaire, afin de prendre de bonnes habitudes et de simplifier le travail de correction.

Le formalisme exact de la présentation des commentaires, dans ses détails, dépend évidemment du langage et de l'industrie ou de l'école.

Nous définissons ici les règles de style pour le langage C en se basant sur les usages des programmeurs C. La référence suivante indique quelques règles de style couramment utilisées en particulier le développement des programmes UNIX (Indian Hill C Sytle and Coding Standards), Linux (Coding Sytle de Torvalds Linus), et des logiciels libres (GNU coding standards), qui nous ont inspirés : http://www.chris-lott.org/resources/cstyle/
.

2 Documentation des programmes et des fonctions

Tout fichier source doit commencer par l'**entête** suivante:

```
/* Fichier : fichier.c
 * Auteur  : Dupont Jean
 * Date    : 8.10.2004
 *
 * But     : Description précise de ce que fait le programme.
 *
 * Remarque(s) : Description de tous les compléments nécessaires pour
 *               l'exécution du programme (bibliothèques, matériel spécifique).
 *
 * Modifications : Date / Auteur / Raison
 *
 * Compilateur : GCC/DevCPP
 */
```

La présentation du programme doit garantir une impression correcte en format A4.

Les accolades du programme principal sont placées en début de ligne, de même que celles des fonctions:

```
void main ( void )
{
   …
}
```

Toute fonction comporte:

- placée avant la fonction: une description de sa fonctionnalité, de l'utilisation de ses paramètres en se référant à leur nom et des erreurs pouvant se produire.
- les variables globales utilisées, si leur utilisation se justifie (fonctions de rappel)

Si les paramètres exigent des commentaires, mettre un seul paramètre par ligne.

Exemple:
Une fonction qui additionne deux nombres.

```
/*
 * retourne la somme de nombre_1 et de nombre_2
 */
int somme (int nombre_1 ; int nombre_2)
{
  …
}
```

Autre forme:

```
/*
 * retourne la somme des deux nombres;
 */
int somme
   (int nombre_1 ;      /* premier nombre*/
    int nombre_2)       /* second nombre*/
{
  …
}
```

Remarques:

- Le nom d'une fonction sans valeur de retour devrait toujours être un verbe à l'infinitif, celui d'une fonction avec valeur de retour un nom ou un adjectif. Ils peuvent être complétés au besoin par un article, un nom ou un adjectif.

Exemples:

```
void presenter_programme(…)
int nombre_maximum(…)
```

- La fonction ne devrait réaliser qu'une opération ;
- Le nombre de paramètres être minimal ;
- Le code ne devrait pas être trop long (idéalement pas plus d'une page) ;
- On ne met pas de caractères accentués dans le code (sauf éventuellement pour les commentaires) ;
- Choisir de manière consistante la langue des commentaires (français ou anglais).

3 Déclaration des constantes

Afin de faciliter la maintenance du programme, on déclare toutes les constantes utilisées au début du programme en les commentant si nécessaire. Les constantes sont écrites en MAJUSCULES. Toute valeur numérique devrait a priori être déclarée comme constante:

Exemple:

```
/* nombre maximum d'éléments du tableau */
#define     MAX   100
```

4 Déclaration des variables / identificateurs

Dans tous programmes, procédures ou fonctions, et quel que soit le langage, chaque variable:

- doit posséder un nom significatif;
- est expliquée par un commentaire si le nom n'est pas explicite;
- en général doit être la seule déclaration d'une ligne.

L'explication se met si possible à côté de la déclaration:

Exemples:

```
int borne_inf;      // limite inférieure du traitement
int borne_sup;      // limite supérieure du traitement
```

Les identificateurs sont à mettre en minuscules, les noms composés étant reliés par un sous-ligné. Les conventions étant variables en C, d'autres formes seront admises (exemple: borneInf) pour autant que vous soyez raisonnables et cohérents dans vos notations.

Note:
Utiliser plutôt compteur_utilisateurs_actifs que cmputa !

5 Mots réservés

Les mots réservés sont **toujours** en minuscules.

Exemples:

Mots réservés

```
main   return
while  if
```

Identificateurs

```
borne_inf     est_lu
```

Au début il n'est pas simple de distinguer les identificateurs des mots réservés. Un bon logiciel permet cependant d'imprimer les mots réservés en gras.

En C, les types prédéfinis sont des mots réservés et s'écrivent en minuscules. Par contre les types définis par l'utilisateurs s'écrivent en majuscules:

Exemple:

```
typedef     ... COULEUR;          // type COULEUR
           COULEUR ma_couleur;  // variable ma_couleur de type COULEUR
```

6 Mise en forme du code

6.1 Indentation

L'indentation correcte du code en facilite énormément la lecture. Beaucoup d'erreurs proviennent d'une mauvaise indentation.

Il est conseillé d'utiliser **2 ou 3 espaces,** en respectant la même convention pour tout le programme.

Utilisez des tabulations pour faire l'indentation, mais en forçant la substitution de ces tabulations par des espace dans votre environnement. Sinon, si vous changez d'éditeur de texte (ou même de poste) et que la taille de la tabulation est différente, toute votre mise en page est à refaire.

Utilisez également une police de chasse fixe (Courier, FixedSys, Terminal, Monaco sur Mac, …) pour que votre mise en page ne change jamais.

6.2 Alignement

Indentation et alignement des instructions sont important pour la lisibilité du code.

Exemple :

```
if (x>2)
  x++;
else
  x--;
```

Les accolades sont soit toutes 2 alignées sur le mot clé, soit seule l'accolade fermante est aligné, l'accolade de gauche ouvrant immédiatement un bloc d'instructions sur la même ligne que le mot clé de la structure, ce qui permet de gagner une ligne.

Exemple plus complexe :

```
/*variante concise*/
if (x>2){
  x++;
  y++;
}
else {
  x--;
  y++;
}
```

```
/*variante avec accolades alignées*/
if (x>2)
{
  x++;
  y++;
}
else
{
  x--;
  y++;
}
```

Il convient aussi d'utiliser systématiquement les accolades lors de **if** imbriqués, pour éviter des confusions en alignant ces accolades:

```
if (juste){
  if (chance)
    gagne() ;
  else
    perd() ;
}
```

Une instruction **switch** sera indentée de la manière suivante :

```
switch(…){
  case …:
    … ;
    break ;
  case …:
    … ;
    break ;
  default :
    …;
    break;
}
```

```
switch(…)
{
  case …:
    … ;
    break ;
  case …:
    … ;
    break ;
  default :
    …;
    break;
}
```

Une instruction **while** sera indentée de la manière suivante :

```
while (continue){
   …
}
```

```
while (continue)
{
   …
}
```

Une instruction **for** sera indentée de la manière suivante :

```
for ( ; ; ){
   …
}
```

```
for ( ; ; )
{
   …
}
```

Lors de déclarations, afin de bien distinguer les variables et les types, on aligne généralement les identificateurs.

Exemples:

```
int   nombre1,nombre2 = 0;
char  resultat;
```

7 Commentaires

Remarques importantes:

- Les commentaires se mettent lors de l'écriture du programme et non après!

- Un commentaire expliquant une partie de programme vient avant cette partie et non après.

- Un commentaire qui n'apporte aucune information ne sert à rien, il est au contraire nuisible à la lisibilité du code. Il doit expliquer ce que fait le code au niveau de l'algorithme et non pas ce que fait l'instruction elle-même.

  Exemples **incorrects**:

```
/*Incrémenter i de 1*/
i++;

/* Si i plus grand que 0 alors  */
if (i > 0)
```

  On le voit en lisant l'instruction! Si une explication est donnée, ce sera pour justifier la présence de cette opération à cet endroit.

  Exemples:

```
/* Passage à l'élément suivant */
i++;

/* Si l'indice existe ... */
if (i > 0)
```

- Les articulations logiques d'un programme sont mises en évidence par une indentation. A un même niveau, si le traitement comporte plusieurs phases, elles sont mises en évidence par une explication en en-tête du bloc et une séparation (ligne vide, ...) nettement visible!

  Exemples:

```
/* Présentation du programme */
printf("   …     ");
printf("   …     ");
printf("   …     ");

/* Boucle d'affichage de l'alphabet minuscule*/
for (char c= 'a'; c <= 'z'; c++){
  printf(…);
}
```

8 Maladresses à ne pas commettre

Voici quelques maladresses fréquemment rencontrées dans le cadre des laboratoires.

Mauvaise utilisation des variables booléennes:

Maladroit	Juste

1) Affectation en fonction d'un résultat de test

```
if (somme > limite)
  depassement = true
else
  depassement = false
```

```
depassement = somme > limite ;
```

2) Test de la valeur d'une variable booléenne

```
if (depassement == true) …
```

```
if (depassement) …
```

```
if (depassement == false) …
```

```
if (!depassement)
```

Comparaison impossible:

`if (valeur > INT_MAX)` … est impossible.
Si `valeur` est réellement plus grande que `INT_MAX`, le programme aura déjà rencontré des problèmes avant.