L’écriture des programmes en C/C++

Le programmeur doit s’efforcer de rendre son code le plus lisible possible afin de facilité la mise au point et la maintenance du programme

1) L'indentation

L'indentation consiste en l'ajout de tabulations ou d'espaces dans un fichier, pour une meilleure lecture et compréhension du code.

Une 'indentation correcte permet de définir clairement où commence un bloc d’instruction et ou il se termine.

L’indentation recommandée est de huit espaces ce qui correspond à une tabulation.

En C les espaces et les tabulations sont ignorés par le compilateur, ils ne serviront qu’à présenter clairement le code.

Exemple :

```
switch (suffix) {
        case 'G':
        case 'g':
                mem <<= 30;
                break;
        case 'M':
        case 'm':
                mem <<= 20;
                break;
        case 'K':
        case 'k':
                mem <<= 10;
                /* fall through */
        default:
                break;
        }
```

Ne mettez pas les affectations multiples sur une seule ligne .
exemple à éviter :

```
char c=3 ;char v=67, float u=3.14 ;
```

2) Longueur des lignes

Eviter les lignes et chaines de caractères trop longues, ne dépasser pas 80 caractères pour une ligne.

Exemple :

```
char chaine[]= « Bonjour, voici une chaine bien longue qu’il est préférable d’écrire
sur deux lignes, le C ignorant les espaces cela ne changera rien au code produit » ;
```

3) Les accolades

Il n'y a pas de règle précise pour repérer les groupes d'instructions, cependant les créateurs du C (Kernighan and Ritchie) recommandent de placer l'accolade ouvrante à la fin de la ligne et l'accolade fermante alignée sous l'instruction d'ouverture

Exemple :

```
if (x is true) {
                on fait …
        }
```

Ceci est valable pour tous les blocs d'instructions non-fonction (if, switch, for, while, do).
Exemples :

```
switch (action) {
        case ADD:
                return "add";
        case EMOVE:
                return "remove";
        case CHANGE:
                return "change";
        default:
                return NULL;
}


do {
                body of do-loop
} while (condition);



if (x == y) {
                ..
}
else if (x > y) {
                ...
}
else {
                ....
}
```

Cependant, dans le cas des corps de fonctions, on placera de préférence l'accolade ouvrante en début de deuxième ligne sous la déclaration de la fonction.
Ne pas utiliser les accolades où une seule déclaration suffit.

Exemples :

```
if (condition)
        action();

if (condition)
```

```
        fairececi();
else
        fairecela();
```

4) Les espaces

On place un espace après la plupart des mots-clés. Les exeptions sont sizeof, typeof, alignof et __attribute__, qui s'écrive comme des fonctions.
Limiter au maximum les espaces, on ne les placera uniquement llorsqu'ils permettent une meilleur lecture comme l'étoile des pointeurs mais pas d'espace autour du '.' et "->" opérateurs membres de la structure.

Exemples :

```
t=sizeof(int) ;
char * p ;
t.m=23 ;
u->p=2 ;
```

5) Le nommage

Le nommage est un élément fondamental permettant une lecture facile d'un programme en C. Un bon nommage demande de l'imagination, le nommage doit être clair, simple et représentatif de du rôle de l'élément nommé.

Eviter les noms trop simples et les noms trop compliqués.
Il faut cependant veiller à ce que le nommage soit lisible, on préférera « compteur_d_utilisateurs() » à « cntusr () ».
Pour séparer les mots d'un nommage on utilise la casse ou « _ » mais pas les deux

calculDuPrix ou calcul_du_prix

Pour le nommage des données, préférer des noms simples comme « tmp » plutôt que « CetteVariableEstTemporaire » même si ici l'utilisation de la casse facilite la lecture.

Choisir un nom représentatif :
Exemple pour un compteur de boucles :

```
int i ;
```

ne signifie rien, préférer

```
int cpt ;
```

6) Typedefs

Veiller à ne pas masquer le type réel par un typedef.
Exemple : typedef char bonbon;
puis

```
bonbon tagada ;
```

Cela crée un nouveau type sans rien apporter à la compréhension du code mais nuit à la lecture.

typedef sera utilisé principalement pour la déclaration de structures :

```
typedef struct livre
```

```
{
  char  titre[50];
  char  auteur[50];
  char  sujet[100];
  int   livre_id;
} ;
```

La déclaration :
livre mobydick ;
est ici appropriée.

Les typedefs U8 / u16 / u32 sont couramment utilisés, ils n'apportent qu'une écriture allégées, mais ne rendent pas le code plus clair. Souvent inconnus des éditeurs de texte, ils ne bénéficient pas de la couleur syntaxique.

7) Commentaires

La norme C99 autorise les // pour commenter une ligne, mais les programmeurs C préfèrent en générale les commentaires multilingues avec une étoile en début de chaque ligne (la zone de commentaires est ainsi facilement repérable).

```
/ *
* Ceci est un commentaire multi-lignes
* avec une * en debut de chaque ligne
*
* Description: Une colonne d'astérisques sur le côté gauche
* /
```

Pensez à commenter chaque fonction en indiquant les variables d'entrées, de sortie ainsi que ce que fait la fonction.

Commenter également chaque déclaration de donnée.

8) énumérations

Les noms des macros définissant des constantes et des étiquettes dans les énumérations sont capitalisés.

```
#define 0x12345 CONSTANT
```

On utilise toujours une énumération lors de la définition de plusieurs constantes connexes.

```
enum jour {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE};
/* les valeurs d'une énumération sont entières et commencent à 0*/
enum jour j1, j2;
j1 = LUNDI;
j2 = MARDI;
```

9) Astuces

S'il faut isoler provisoirement une portion de code, le mieux est de ne pas utiliser les commentaires (il pourrait y avoir des commentaires imbriqués), mais plutôt les directives du préprocesseur : #ifdef .. #endif ou #if .. #endif

Sélectionnez

```
#if 0
  /* Compteur */
  int cpt ;
#endif
```