Langage C : Guide de style élémentaire

Les fonctions : Chaque fonction est précédée d'un bloc de commentaires précisant le nom de la fonction, son rôle, les paramètres en entrée et en sortie ainsi que sa valeur de retour. Les fonctions C qui ne retournent rien sont déclarées comme ayant le type **void**. Par exemple :

```
/* ---------------------------------------------------------------- */
/* strcmp          Compare deux chaînes de caractères               */
/*                                                                  */
/* En entrée: s1, s2 Deux chaînes de caractères                     */
/*                                                                  */
/* En sortie: La valeur entière retournée est 0 si les chaînes sont */
/*            égales; négative si s1 < s2 et positive si s1 > s2.   */
/* ---------------------------------------------------------------- */
```

Déclarations : Les déclarations sont alignées suivant leurs types. Les pointeurs sont décalés d'un caractère. Par exemple :

```
int                     i,j;
struct adresse          Adresse_Benny;
char                  * ptr;
float                   x = 10.45,
                        y = 20;
```

Commentaires : Les commentaires sont alignés. Les sources sont aérés autant que possible.

Indentation : Les sources sont indentés de manière cohérente et rigoureuse. Exemple d'indentation avec 3 caractères.

```
if (condition)   while(condition)    for(condition)    do
{                {                   {                 {
   Bloc alors       Bloc Tant que       Bloc pour         Bloc repeter
}                }                   }                 } while(condition);
else
{
   Bloc sinon
}
```

Fichiers : Un fichier commence par un bloc de commentaires donnant le nom du fichier et son rôle, que ce soit un entête (fichier .h : déclaration des types, constantes et prototypes) ou un module (fichier .c).

ATTENTION: Les fichiers d'entête (.h) ne doivent contenir que des déclarations de types, des déclarations de constantes, de macro-instructions, des inclusions d'autres fichiers d'entête, des déclarations d'accès à des variables externes ou des directives de compilation. En aucun cas ils ne doivent contenir de code C exécutable ou des déclarations de variables.

Conventions de nommage pour les constantes, variables et sous-programmes :

Une des façons d'obtenir du code clair est de s'en tenir à une convention de nommage des identificateurs qui soit cohérente et parlante. Eviter les noms trop proches les uns des autres. Utiliser des noms courts et explicites

Les types utilisateurs : Les noms des types définis par l'utilisateur se terminent par **'_t'**. Par exemple **'complex_t'** ou **'boolean_t'**.

Les constantes : Les constantes définies par l'utilisateur sont en majuscule. Par exemple : **#define PI 3.14**. On doit toujours essayer de nommer les constantes.

Les variables (int, char, etc.), les sous-programmes seront en minuscule. Dans le cas où ces variables ont des noms composés, chaque mot commence par une majuscule. Par exemple :
int NumeroLivre = 10;
AfficherNomLivre ();

Il faut rester homogène dans les noms des sous-programmes, ne pas donner le nom RechercheValMax() à un sous-programme puis ValMinRechercher() à un autre et enfin MoyValRechercher() à un autre!

Il est possible d'utiliser le « _ » pour séparer les mots plutôt que les majuscules dans les identificateurs, par exemple :
Numero_livre
recherche_val_max()

Les noms comme I, J, A, B sont interdits sauf s'ils sont utilisés comme variables de contrôle de boucles.

BONNES PRATIQUES :

Pas d'écritures de messages d'erreur dans les fonctions de base : la fonction retourne un indicateur ou code d'erreur, que le programme appelant interprète et traite.

Donner pour chaque fonction :

- un lexique complet (pas seulement les paramètres d'entrée/sortie mais aussi toutes les variables locales).
- un exposé du principe (appelé aussi stratégie) et/ou des commentaires explicites.

Pas de return au milieu des fonctions et un seul return par fonction: on gère une variable résultat. Pas de breaks au milieu des boucles, prendre une variable booléenne pour arrêter.

Faire un fichier .h par SDD (exemple un pour liste chaînée, un pour la pile, une file, un arbre).
Faire un fichier source par SDD qui contient toutes les fonctions (primitives) de cette SDD.

Jamais de duplication de code, dès qu'un traitement apparait 2 fois faire une fonction.

Eviter les menus longs et répétitifs.

Ne pas utiliser une variable qui soit à la fois une donnée et un contrôle, en particulier dans les retours de fonction.
Exemple :

Ne pas faire : une fonction retourne un entier et si cet entier vaut 0 cela veut dire qu'il y a eu une erreur.
A faire: une fonction fournit un entier en paramètre de sortie et retourne un code qui vaut 1 si OK et 0 si erreur.
Cas particulier : on peut concevoir des fonctions qui retournent une adresse ou NULL.

Utiliser des fichiers textes pour les jeux d'essais. Organiser les essais de manière à tester rapidement tous les cas mais pas plusieurs fois le même cas !