Règles de Programmation C

1 Introduction

Ce document fixe un certain nombre de règles élémentaires dans le but d'aider le développeur à écrire des programmes C de manière propre et compréhensible.

Ce document est non exhaustif.

2 Organisation des programmes

Les programmes C doivent être écrits selon le principe de la programmation modulaire. Est appelé un module l'ensemble <fichier interface – fichier implémentation> regroupant les fonctions traitant d'un seul et même sujet (manipulation d'une structure, thème, …). Par exemple, le module *Pile* est constitué d'un fichier interface *Pile.h* et d'un fichier implémentation *Pile.c*. Un fichier indépendant contiendra la fonction main() qui lancera l'application. Celle-ci sera obligatoirement de la forme suivante :

```
int main(int argc, char *argv[])
```

Le fichier interface contiendra les définitions des structures de données, les définitions des constantes symboliques (#define), les prototypes des fonctions, etc.

3 Compilation

L'utilisation d'un *Makefile* est obligatoire.

4 Règles d'écriture

4.1 Règles de présentation

- Un module commence toujours par un cartouche qui le décrit.
- Chaque fonction est précédée d'un cartouche décrivant son rôle, ses paramètres d'entrée et sa valeur de retour.
- Un bloc d'instructions est toujours placé entre accolades (même s'il n'y a qu'une instruction).
- Pas plus d'une instruction par ligne.
- Aucun commentaire ne se trouve sur la même ligne qu'une instruction, excepté pour commenter une déclaration.
- Les commentaires décrivent les fonctions du code et non le code lui-même.
- Au maximum une variable est déclarée par ligne.
- Les commentaires sont tous dans la même langue.
- Au maximum un champ de structure est déclaré par ligne.
- Un commentaire suit chaque champ de structure.
- Un commentaire suit chaque déclaration de variable sauf si le nom de la variable est assez clair (ce qui est recommandé).
- Une valeur retournée par la fonction `return` est placée entre parenthèses.

4.2 Règles de nommage

- Les noms des identificateurs et des fichiers utilisent la même langue.
- Les noms des modules, variables, sous-programmes, … sont en minuscules sauf le premier caractère qui est en majuscule (ex: `Pile`, `Lire`, `Donnee`, …). Si le nom est composé, les noms élémentaires seront collés et la première lettre der chaque mot, hormis le permier, sera en majuscule (ex: `gestionDePiles`, i`TailleDe`, iNombreDeFiles, …).
- Les noms représentant des acronymes peuvent être en majuscules (ex: gestionDesES).
- Les noms de paramètres formels et noms de variables locales utilisent les mêmes règles que ci-dessus.

- Les noms de variables sont systématiquement préfixées par une lettre ou un groupe de lettre représentant leur type (**i**NbTour pour un entier, strChaine pour une chaîne de caractère, cLettre pour un caractère,…). Pour les structures, le développeur trouvera un préfixe significatif de 3 lettres maximum.
- Les noms de variables et de fonctions sont **significatifs**. Les noms i, j ou k sont tolérés pour les boucles uniquement.

4.3 Règles de structuration

- Un module est toujours composé d'une partie interface (`.h`) et d'un corps (`.c`).
- Un module ne dépasse pas 1000 lignes de code.
- Toutes les variables ou fonctions exportées par un module sont déclarées dans son interface.
- Le corps d'un module fait explicitement référence à son interface.
- L'interface d'un module est auto-protégée contre les inclusions multiples :

```
#ifndef   INTERFACE_PILE
#define   INTERFACE_PILE
...
   interface de l'objet
...
#endif    INTERFACE_PILE
```

4.4 Règles de codage

- La norme ISO/ANSI est utilisée.
- Toutes les fonctions sont explicitement typées.
- Les déclarations de types et de variables sont effectuées séparément.
- Les dimensions d'un tableau sont des constantes symboliques.
- L'utilisation de variables globales est interdite à moins que le développeur n'ait d'autre solution.
- L'instruction **`goto`** est interdite.
- L'affectation multiple est interdite.
- Chaque unité de traitement d'un **`switch`** est terminée par un **`break`**.
- Toute instruction **`switch`** contient une branche **`default`** même vide.
- L'instruction **`switch`** est préférée à la structure **`if..elseif..else`** si les deux sont applicables.
- L'instruction **`for`** est préférée à l'instruction **`while`** pour les traitements itératifs basés sur un compteur.
- Pas d'affectation dans une condition.
- Les fonctions de type différent de **`void`** terminent leur exécution par un **`return(val)`**.
- Les expressions sont bien parenthésées.
- Tous les cas d'erreurs sont testés et traités.
- Les indices sont testés avant d'accéder à un tableau.
- La valeur d'un indice de boucle n'est pas réutilisée en dehors de la boucle.