Règles de programmation
ING1 – 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/C++, Java, ... , de manière propre et compréhensible.
Ce document n'est pas exhaustif.

2 Organisation des programmes

Les programmes C/C++, Java, ... 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 `Pile.cpp`. 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)
```

ou

```
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 exportées, ...

3 Compilation

L'utilisation d'un `Makefile` est obligatoire. Il s'appellera obligatoirement `Makefile`, et comportera au minimum les cibles `all`, `clean`, `doc`, `save` et `restore`.

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.

— 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.

— 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, fonctions, ... sont en minuscules. Si le nom est composé, les noms élémentaires seront collés, et la première lettre de chaque mot sera en majuscule, sauf le premier mot (ex : `gestionDesES, saisieNomFamille`, ...).

— Les noms représentant des acronymes peuvent être en majuscules.

— Les noms de paramètres formels et noms de variables locales utilisent les mêmes règles que ci-dessus.

— Les noms de variables sont systématiquement préfixés par une lettre ou un groupe de lettre représentant leur type (`int_NbTour` pour un entier, `str_NomFamille`, ...). Ce préfixe sera en minuscule. Pour les structures, le développeur trouvera un préfixe significatif de cinq lettres maximum.

— Les noms de variables et de fonctions sont *significatifs*. Les noms `i`, `j`, `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`).

— 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
```

4.4 Règles de codage

— Toutes les fonctions sont explicitement typées.

— Les déclarations de type 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 programmeur 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 l'instruction `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'erreur sont testés et traités.

— Une fonction retournant un code d'erreur doit retourner une valeur négative.

— La valeur d'un indice de boucle n'est pas réutilisée en dehors de la boucle.

4.5 Blocs, accolades et indentation

Un bloc est toute partie d'un source C encadrée par des accolades.
On demande de respecter la présentation suivante :

— Une accolade ouvrante doit être seule sur une ligne ou sur la ligne précédente, si elle se trouve à la fin de elle-ci, est séparée par un espace du reste du code.

— Une accolade fermante doit être seule sur une ligne, sauf si elle s'inscrit dans le schéma suivant `} else {`.

— La ligne l_2 qui suit une ligne l_1 constituée d'une accolade ouvrante doit commencer une tabulation plus loin que la ligne l_1.

— La ligne l_2 constituée d'une accolade ouvrante qui précède une ligne l_1 doit commencer une tabulation moins loin que la ligne l_1 .

— Toute ligne l_2 autre qu'une accolade ouvrante ou fermante précédée d'une ligne l_1 doit commencer au même niveau que la ligne l_1.

4.6 Cartouches et commentaires

Un commentaire commente ce que fait le code, et non pas le code lui-même. Il n'est en aucun cas la traduction du code, mais la traduction de la pensée du développeur.
Tout fichier doit commencer par un cartouche.
Un cartouche, ou entête, est constitué :

— d'une explication sur le contenu du fichier;

— du nom du ou des auteurs;

— de la date de création du fichier;

— de la date de dernière modification du fichier;

— et d'un numéro de version.

Toute fonction doit être commentée pour pouvoir plus facilement être :

— déboggée (correction des erreurs)

— appelée par une autre fonction de bonne façon.

Dans cette optique, on doit insérer un commentaire juste avant l'entête de fonction. Ce commentaire doit comprendre :

— La signification de chaque paramètre.

— Des préconditions : ensemble des conditions (evt. vide) que doivent vérifier les paramètres pour que la fonction s'exécute dans le cadre prévu par le programmeur de cette fonction.

— Ce que fait la fonction.

— Ce que retourne la fonction si elle retourne un résultat.

5 Déverminage

Le déverminage est l'affaire du programmeur mais en aucune manière celui de l'utilisateur. Vous ne devez donc en aucun cas laisser des messages de déverminage lors de la remise d'un programme.