Consignes pour l'écriture du code v0.3

Ce document décrit les consignes à respecter concernant l'écriture du code.

Modularité

- Par rapport à un code peu modulaire, un code modulaire :
  - est plus simple et facile à comprendre ;
  - est plus facile à tester ;
  - est plus facile à mettre à jour ;
  - est plus facile à maintenir ;
  - possède des fonctionnalités plus facilement utilisables dans un autre projet ;
  - possèdera un nombre inférieur de bugs.
- Le code complet d'une fonction devrait être visible sur un écran en un coup d'oeil. Les fonctions plus longues doivent être rares et leur longueur justifiée (exemple : code à très faible complexité).
- Chaque module doit posséder une grande cohésion et cacher tout ce qui n'est pas nécessaire à son utilisation.
- Un « module » est typiquement défini comme un code accomplissant une fonction particulière. Cela peut être une fonction avec variables associées, ou un ensemble de fonctions travaillant ensemble pour accomplir une fonctionalité ou un ensemble de fonctionalités étroitement liées.
- Propriétés d'un code modulaire :
  - correspondance claire entre le nom du module et l'objet modélisé ;
  - code court ;
  - haute cohésion au sein d'un module : un module réalise une tâche précise et contient uniquement des fonctions liées à cette tâche ;
  - faible dépendance entre modules ;
  - encapsulation :
    - information/état du module protégé ;
    - manipulation uniquement à travers des fonctions dédiées ;
    - bien distinguer les fonctions internes au module et celles permettent d'utiliser le module.

Peu de variables globales

- Eviter d'utiliser des variables globales, sauf si cela est réellement justifié.
- Conséquences de l'utilisation excessive de variables globales :
  - code peu modulaire ;
  - augmentation de la complexité du code ;
  - augmentation du nombre de bugs ;
  - code difficile à maintenir.
- Toute variable devrait si possible être déclarée localement (si besoin avec le mot-clé static).

Propreté du code

- Respecter les mêmes conventions de nommage (et autre) dans **tout** le code. Exemple : pas de mélanges comme "square_the_biggest", "squareTheBiggest", ou "SquareTheBiggest".
- Pas de variables ou code inutilisé.

- Eviter les valeurs numériques littérales « tombant du ciel » ; à remplacer par des constantes.
- Pas de messages de débogage, sauf si un mode debug a été implémenté.
- Code homogène : espaces, indentations, même style et règles partout.
- Toute ligne de code ne doit pas dépasser 120 caractères.
- Indenter toujours le code, autrement celui-ci sera illisible :
  - utiliser des tabulations **ou** des espaces, **mais pas** un mélange des deux.
- Les noms de variables, structures, fonctions, etc. doivent être judicieusement choisis, **explicites** et en **anglais**.

Fichiers headers (.h)

- Un fichier header existe pour chaque module du système qui doit exporter des fonctions (ce n'est pas forcément le cas du fichier source contenant le main() par exemple). Un module implémente un aspect du système et peut comprendre un ou plusieurs fichiers .c.
- Un fichier header contient les prototypes des fonctions décrivant l'interface **publique** d'un module. Ces fonctions sont visibles et utilisables par les autres modules du système.
  - Les prototypes de fonctions sont préfixés par le mot clé extern.
- **Eviter** d'inclure des fonctions, macros, structures, etc. qui sont uniquement utilisées dans le code du module. En effet, le but est de n'exposer que les « objets » publiques. Tout ce qui est privé à un module doit être caché et non visible aux autres modules !
  Exemple : ModuleA doit uniquement appeler des fonctionalités de ModuleB à travers l'interface publique de ModuleB, via ModuleB.h.
- **Eviter** d'insérer le corps de fonctions ou des déclarations de variables dans un fichier header (exception pour les fonctions inline).
- **Eviter** d'exposer (avec le mot clé extern) des variables d'un module dans un fichier header. L'état interne d'un module ne devrait pas être visible/accessible depuis un autre module. L'état interne devrait être représenté par des variables privées déclarées static.

Fichiers sources (.c)

- Un fichier .c ne devrait pas inclure d'autres fichiers .c, à moins d'avoir une très bonne raison de le faire.

Allocation dynamique

- L'allocation dynmique avec malloc est à éviter si possible :
  - risque de leak mémoire dû à l'oubli de libérer la mémoire avec free ;
  - code moins performant : malloc est relativement lent à exécuter ;
  - si la taille de l'espace alloué est statique, malloc est inutile : préférer l'allocation d'un tableau statique sur la pile.

- Toute zone mémoire allouée avec malloc **doit** être libérée avec free à quelque part dans le programme.
- Si possible, tester l'exécution du programme à travers l'outil d'analyse d'allocation dynamique Valgrind.

Gestion d'erreurs

- **Toujours vérifier** qu'une fonction s'est exécutée correctement en gérant la valeur de retour de manière appropriée.

Commentaires

- Les commentaires doivent être en **anglais**. Si vous avez de la difficulté, demandez-moi et/ou utilisez google translate (même si celui-ci n'est pas parfait).
- En début de chaque fichier source (.c), indiquez :
  - le nom de l'auteur ou des auteurs ;
  - la date de dernière modification ;
  - une description générale de ce qui est implémenté dans le fichier.
- Chaque fonction (fichier .c) doit être commentée par un header de fonction :
  - le rôle de la fonction ;
  - le rôle de chaque paramètre ;
  - si la fonction renvoie une valeur : le rôle de celle-ci.
- Le reste du code doit être suffisamment clair, lisible et bien structuré pour ne pas avoir besoin d'explications. Comment faire ?
  - Choisissez des noms explicites pour les variables, structures, fonctions, etc.
  - Modularisez le code : par exemple, remplacez un bloc non trivial par une fonction au nom explicite.
  - Ecrivez des fonctions courtes.
- Au cas où un code obscure subsisterait, celui-ci doit être commenté. Un tel code est cependant à **éviter absolument**.

Conseils

- Compilez votre code avec les options `-Wall` et `-Wextra` puis éliminez tous les messages d'avertissement.