3ETI, Développement logiciel en C

Règles de codage.

Règles générales d'organisation et de lisibilité:

Chaque niveau **d'abstraction** est contenu dans un fichier distinct.
Les **prototypes** (ou signature) des fonctions sont écrites dans un fichier **.h**
L'implémentation de ces fonctions sont écrites dans un fichier **.c** du même nom.

Afin de privilégier la **lisibilité**, les variables seront déclarées le plus **localement** possible. De préférence juste avant leur zone d'utilisation (et non toutes au début de fonction, *norme C99*).

Initialisation des variables:

Afin de minimiser l'impact des erreurs, toutes les variables seront initialisées dès leur déclaration.
- Soit à leur valeurs finale si celle-ci est connue.
- Soit à une valeur par défaut (0 pour un entier, `NULL` pour un pointeur).

Cas particulier important:
A tout endroit de votre programme, tout pointeur doit soit pointer vers une zone valide, soit pointer vers NULL.

ex1. INCORRECT
```
int *g;
ma_fonction(g);
```

ex2. CORRECT
```
int *g=NULL;
ma_fonction(g);
```

Commentaires:

Chaque fichier d'en-tête contient un commentaire introductif décrivant le **rôle de l'abstraction** décrite et son utilité dans le jeu d'échec.

Chaque fonction est décrite en détail dans le fichier .h, au niveau de sa signature (ce fichier faisant partie de la librairie visible par l'utilisateur finale).
Ce commentaire doit suivre s'inscrire dans le cadre d'une **programmation par contrat** (ou assertion).

Il doit en l'occurrence décrire:
- Le **rôle** de cette fonction.
- Les **suppositions** sur les paramètres d'entrées (ces suppositions seront traduites par des assertions). Il conviendra au programmeur de les respecter.

- Les **garanties** obtenues après exécution de la fonction si les assertions sont vérifiées.

Ex. Commentaire sur la déclaration de la fonction écrivant les coordonnées (x,y) dans la structure pièce:

```
/**
 * Fonction piece_ecrire_coordonnee:
 * *********************************
 *   Ecrit les coordonnees (x,y) dans la structure de la piece_du_jeu
 *
 *   Necessite:
 *     - Un pointeur vers une struct de type piece non NULL
 *     - Deux coordonnes x et y entieres comprises sur [0,7]
 *   Garantie:
 *     - La piece est mise a jour avec les coordonnees donnees en parametres.
 */
```

Dans **l'implémentation** de la fonction (fichier .c), chaque fonction contiendra des commentaires correspondant à une méthode de **programmation par algorithme**.

Pour cela, on retrouvera dans le corps de la fonction:

- L'algorithme global suivi.
- Potentiellement, des commentaires supplémentaires devant chaque bloc non trivial.

Passage par paramètres:

Tous les types “**plain old data**” (int, float, …) non modifiés par la fonction seront passés en paramètres par **copies**.

Toutes les **structures** spécifiées par l'utilisateur (struct) seront passées par **pointeurs**.

- Si la structure n'est pas modifiée par la fonction, elle sera qualifiée par le mot clé `const`.
- Si la structure est modifiée par la fonction, elle ne sera pas qualifiée par `const`.

Nommage de variables/fonctions:

- Il est impératif d'utiliser des noms de fonctions **précis et signifiants**.

- Afin de bien différencier les **niveaux d'abstraction** dans l'appel des fonctions, la dénomination du fichier (ou de la struct sur laquelle la fonction agit) est répété dans la première partie du nom de la foncton.

Ex. Pour une fonction qui spécifie les coordonnées (x,y) d'une pièce, on aura la dénomination suivante:

```
piece_ecrire_coordonnee();
```

- Lorsqu'une fonction agit spécifiquement sur une struct passée en paramètre, on passera cette structure en tant que **premier argument**.

Ex. Fonction ecrivant les coordonnees (x,y) dans la structure piece:

```
void piece_ecrire_coordonnee(piece *piece_du_jeu,int
coordonnee_x,int coordonnee_y);
```

- Les noms de fonctions utilisent une action ou un verbe à l'infinif pour décrire leur rôle.

Ex. fonction d'invalidation d'une pièce du jeu :

```
void piece_invalider(piece* piece_du_jeu);
```