Règles de nommage

✔ Les noms des variables, des fonctions et des fichiers :
- doivent avoir les noms les plus explicites.
- doivent être composées de minuscules, de chiffres ou des caractères '-' et '_'.
- peuvent être sous forme de noms composites mais séparées par '_' (underscore).
- ne doivent pas être mis au pluriel (pas de 's' à la fin) sauf dans le cas de tableau.

Règles de formatage du code

✔ Tous les fichiers sources doivent commencer par un cartouche :

```
/*************************************************************
 *  Nom du projet :  _____________________________
 *  Nom du fichier :  ____________________________
 *  Créé le :        __/__/____  par  _____________
 *  Mis à jour le :  __/__/____  par  _____________
 *************************************************************/
```

✔ Chaque ligne ne peut faire plus de 80 colonnes, commentaires compris. (Attention, une tabulation ne compte pas pour une colonne mais bien pour les 4 espaces qu'elle représente).
✔ Une seule déclaration par ligne.
✔ Une seule instruction par ligne.
✔ Une ligne vide ne doit pas contenir d'espaces ou de tabulations.
✔ Une ligne ne doit jamais se terminer par des espaces ou des tabulations.
✔ Une accolade ouvrante ou fermante doit être seule sur sa ligne avec la bonne indentation.
✔ Vous devez retourner à la ligne à la fin d'une structure de contrôle (if, while, etc.).
✔ Vous devez indenter votre code avec des tabulations qui doivent avoir une taille de 4 espaces (ce n'est pas équivalent à 4 espaces, ce sont bien des tabulations → modifier peut-être la configuration de votre éditeur).
✔ Chaque virgule ou point-virgule doit être suivi d'un espace si nous ne sommes pas en fin de ligne.
✔ Chaque opérateur (binaire ou ternaire) et ses opérandes doivent être séparés par un espace et seulement un.
✔ Les étoiles des pointeurs doivent être collées au nom de la variable et les unes aux autres.
✔ Les déclarations doivent être en début de bloc et doivent être séparées de l'implémentation par une ligne vide.

Les fonctions

✔ Les définitions de fonction doivent être séparées les unes des autres par une ligne vide.
✔ Chaque fonction doit faire au maximum 25 lignes de code (sans compter les accolades du bloc de la fonction).
✔ Un commentaire (placé en début de fonction) doit présenter le rôle de la fonction, ses entrées et ses sorties.
✔ Chaque fonction doit toujours avoir un nom qui a du sens.

Les fichiers headers (.h)

✔ Seuls les inclusions de headers (système ou non), les définitions de structures de données, les *defines*, les prototypes et les macros sont autorisés dans les fichiers headers.
✔ Toute inclusion de header doit être justifiée autant dans un .c / .cpp que dans un .h.
✔ Tous les *includes* de .h doivent se faire au début du fichier (.c ou .h).
✔ Les *headers* doivent être protégés contre la multi inclusion.

Les commentaires

✔ Il doit y avoir des commentaires.
✔ Tous les commentaires doivent être utiles.

Ce qui est interdit

✔ Les variables globales.
✔ L'utilisation de *goto*.
✔ Les noms de variables ou de fonctions en majuscules (les noms de constantes doivent être en majuscules).

Exemple de code

```
/*****************************************************************************
 *  Nom du projet : Jeu de hasard
 *  Nom du fichier : main.cpp
 *  Créé le :           10/06/2020  par  Marcel
 *  Mis à jour le :     19/09/2020  par  Ginette
 *  Auteur(s) : etudiant_de_snir@netocentre.fr
 *****************************************************************************/
#include <iostream> // Requis pour les flux d'E/S cin et cout.
#include <cstdlib>  // Requis par les fonctions srand() et rand()
#include <ctime>    // Requis pour initialiser la fonction srand()

using namespace std ;

int main()
{
    int             nombre_essais_restants = 10 ;
    unsigned short  nombre_secret ;
    unsigned short  nombre_propose = 0 ;
    int retour = 0 ;

    // Initialisation du générateur pseudo-aléatoire (indispensable).
    srand(time(NULL));

    // Génération d'un nombre secret compris entre 1 et 100.
    nombre_secret = rand() % 100 + 1;

    // Tentative du joueur
    cout << "Proposez un nombre : ";
    cin >> nombre_propose;
    if (nombre_propose < nombre_secret )
    {
        cout << "C'est plus." << endl;
        retour = 1 ;
    }
    else if (nombre_propose > nombre_secret)
    {
        cout << "C'est moins." << endl;
        retour = -1 ;
    }
    else
    {
        cout << "Bravo, vous avez gagné !" << endl;
        retour = 0 ;
    }
    return retour;
}
```