Norme de programmation : NoobZik-Dev

Version : v0.01 (En cours de rédaction)

Cette norme est inspirée de l'école 42. Le but d'une norme est de rendre un code source facilement lisible par tout le monde.

La norme a deux objectifs principaux :

- Uniformiser vos codes afin que tout le monde puisse les lire facilement, membres du groupe, personnes extérieur et profs.
- Ecrire des codes simples et clairs.

Si vous êtes dans mon groupe, vous vous engagez à respecter scrupuleusement cette norme.

Pré-requis

- Connaissances parfaites sur le fonctionnement environnement de développement GIT.
- Connaissances partiels du fonctionnement de Gitkraken.
- Aptitude à savoir être autonome sur la correction des erreurs de compilations débiles.
- Coder sur Atom ou Visual Code (**GEDIT STRICTEMENT BANNI**) Avec son header 42.

Note pour atom : vous devrez récupérer la version modifiée du module header 42 à cette adresse (Il est écrit comment appliquer cette modification.).

Quelques plugins pour simplifier la vie (atom)

- Docblocker (Permet d'écrire les commentaire de manière rapide).
- File icon (Permet de colorer les fichiers, dans le but de switcher rapidement).
- Linter (Pour la compilation en live en enrengistrant le fichier).
- Linter GCC Pour le code c.
- Linter Java pour le code java
- Linter Ocaml

Généralités sur GIT

On prend le soin de rappeler que le but de Git est de retracer l'historique des versions du programme / projet. Il est plus facile de revenir en arrière en cas d'introduction de bug.

Si ce n'est pas le cas, vous devez vous dispenser d'une formation git.

- Un commit = une modification sur une fonction ou fonctionnement d'une fonctionnalité (Pas de 1 commit pour une liste de modification).
- S'il y a une erreur dans un code source du projet que vous ne pouvez pas le régler et qu'il est déjà commit, **vous devrez ouvrir un ticket de rapport de bug sur le répertoire concerné**.
- Toutes les descriptions des commit devront clairement dire les modifications apportées.

Norme de programmation

Généralités

Chaque fichier doit commencer dès la première ligne par le header 42

```
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   BinarySearchTree.h                                 :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: NoobZik <rakib.hernandez@gmail.com>        +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2017/11/28 12:16:51 by NoobZik           #+#    #+#             */
/*   Updated: 2017/12/03 15:21:45 by NoobZik          ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */
```

Chaque ligne ne devra pas dépasser 80 colonnes y compris tabulation + espace !

Une tabulation compte pour 2 espaces et non pour 2 tabulation

Commentaires

- Il ne doit pas y avoir de commentaires dans le corps des fonctions.
- En revanche, chaque fonction doit être commenté de la façon suivante :

```
/**
 * [function name]
 * [description]
 * [parameters description]
 * @param [type] [description]
 * @return       [description]
 */
```

Les choses interdits

Il est strictement interdit utiliser les mots-clées suivantes :

- for (car c'est tomber de face)
- do - while
- switch
- case
- goto

Identation générale

- L'indenation d'une fonction en général se fait de la façon suivante :

```
// Une espace entre chaque mot clée
// Correct
```

```
int foo (char*) {

}
// Non correct
int foo() {

}

// Egalement non correct
int foo()
{

}
```

- Une ligne vide ne doit pas contenir d'espace ou de tabulation.
- Dans le cas ou le prototypes d'une fonction dépasse 80 colonnes, il est toléré de retirer un espace. Dans ce cas, l'espace entre l'accolade ouvrante et la fin de parenthèse est prioritaire. Ensuite, vient l'espace entre le type et la parenthèse ouvrante.
- Chaque opérateur (binaire ou ternaire) et opérande doivent être séparés par un espace et seulement un.
- Les mots clées suivantes :

```
if else for while
/* et définition des fonctions */
```

contiennent un espace entre l'instruction et la parenthèse ouvrante.

Accolade optionnel :

Dans le cas où il y a une seule instruction, on n'est pas obligé de mettre des accolades. (Cas particulier en C à voir plus loin).

Les variables devront être alignées verticalement.

```
// Correct
int     boujour;
float   boujourr;
double  poo;

// Incorrect
int bonjour;
float bonjourr;
double poo;
```

Vous pouvez retourner à la ligne lors d'une même instruction ou structure de contrôle, mais vous devez rajouter une indentation par parenthèse ou opérateur d'affectation. Les opérateurs doivent être en début de ligne.

```
// Correct
while (jesuisunevariable < jesuisuneautrevariable && bruh ^ okay
       || voila) {

}
// Incorrect
while (jesuisunevariable < jesuisuneautrevariable && bruh ^ okay ||
        voila) {

}
```

Dès qu'il y a une instruction if - else avec une seule instruction chacune, on doit utiliser la condition ternaire.

```
// Incorrect
if (voila) {
  return true;
}
else {
  return false;
}

// correct
return (voila) ? true : false;
```

Cas particulier

Il existe un cas particulier auquel cette norme ne s'applique pas. Si l'instruction est *break* ou *continue*. Elle ne s'applique pas, car ce sont des mots clés et non une instruction. Essayez vous aurez une erreur.

Forme de la boucle itérative

Puisque les boucles for sont strictement interdit, votre boucle itérative sera donc.

```
int i = -1;
while (++i < max){
  /* instrictions */
}
```

Code C

Compilation

- Votre code devras être compilé avec tous les warnings activés.

```
-Wall -Wextra -pedantic -Wshadow -Wpointer-arith -Wcast-align
-Wwrite-strings -Wmissing-prototypes -Wmissing-declarations
-Wredundant-decls -Wnested-externs -Winline -Wno-long-long
-Wuninitialized -Wconversion -Wstrict-prototypes
```

- Le code devra être compilé avec la dernière norme connue (C11) en plus des optimisations.

```
-std=c11
-O3
```

Dossiers et emplacements des fichiers C / H / divers

- Tous les fichiers .c devront être dans le dossier "Sources".
- Tous les fichiers .h devront être dans le dossier "Headers".
- Les définitions des structures devront être dans le dossier "Headers/Structures".
- Les diverses documents devront être dans le dossier Ressources.

Inclusions des fichiers

- Tous les includes de .h doivent se faire au début du fichier (.c ou .h).
- Il est strictement interdit d'inclure des fichiers .c dans un .c
- Les fichiers .h doivent être protégé contre la multiple inclusion. La définition d'un macro sera témoin de cette protection.

Pour un fichier header contenant les déclarations des fonctions.

```
#ifndef _(NOM_DU_PROJET)_(NOM_DU_FICHIER)_H_
#define _(NOM_DU_PROJET)_(NOM_DU_FICHIER)_H_
#endif // _(NOM_DU_PROJET)_(NOM_DU_FICHIER)_H_
```

Pour un fichier header contenant les structures

```
#ifndef _(NOM_DU_PROJET)_STRUCTURE_(NOM_DU_FICHIER)_H_
#define _(NOM_DU_PROJET)_STRUCTURE_(NOM_DU_FICHIER)_H_
#endif // _(NOM_DU_PROJET)_STRUCTURE_(NOM_DU_FICHIER)_H_
```

Formatage générale du C

- Les étoiles des pointeurs doivent être collés au nom de la variable.
- Une seule déclaration de variable par ligne.
- Les déclarations doivent être en début de fonctions et doivent être séparées de l'implémentation par une ligne vide.
- Aucune ligne vide ne doit être présente au milieu des déclarations ou de l'implémentation.
- Une fonction qui ne prend pas d'argument doit explicitement être prototypée avec le mot void comme argument.

Cas spécifique : Fonction à une seule instruction

- Toute fonction contenant au plus une seul instruction doit avoir le mot clée inline.

```
inline void freeChar(char *b) {
  free(b);
}
```

Définition d'une structure et diverses types

- Une structure doit commencer par s_ et se termine par _s
- Un type énuméré doit commencer par e_ et se termier par _e.

- Un type union doit commencer par u_ et se terminer par _u.
- **Une variable globale doit commencer par g_ et se terminer par _g**
- Après l'accolade fermante, cette structure doit commencer par t_ et se terminer par _t.
- Tous les macros seront écrite en anglais et en majuscule.
- Les variables et les noms doivent être alignée verticalement
- Une émunération = retour à la ligne.

```
typedef struct           s_structfoo_s {
  int                    a;
  float                  b;
  struct s_structfoo_s  \*c; //Sans le \
}                        t_structfoo_t;

typedef enum e_steet_e{
  RUE,
  BOUVEVARD,
  AVENUE,
  ALLEE
}            e_street_e;
```