# **Document de Livraison**

Ce document porte sur le contexte du projet Linter C donné suite à la semaine thématique sur le langage C.

GROUPE : 4

|  |  |
| --- | --- |
| Nom | Prénom |
| Maubert | Allan |
| Cassoly | Guillaume |
| Théral | Hugo |

# **Contexte du projet**

## **Sujet du projet**

Le projet porte sur la création d’un Linter (un outil de linting) qui va analyser du code statique en langage C. Le linter analyse le code selon des règles activable dans un fichier de configuration qui porte l’extension. lconf .

Ce projet a été entièrement développé en langage C.

Il y a 3 types de règles :

|  |  |  |  |
| --- | --- | --- | --- |
| **Règles de conventions** | | | |
| **Règle** | **Valeurs possibles** | **Description** | **Disponible** |
| Array-bracket-eol | on | off | L’accolade doit se trouver en fin de ligne pour les fonctions, if, boucles, … | Oui |
| Operators-spacing | on | off | Il doit avoir un espace de chaque côté entre tous les opérateurs binaires. | Non |
| Comma-spacing | on | off | Il doit avoir un espace à droite d’une virgule. | Oui |
| indent | n | off | L’indentation doit être respectée entre les différents blocs, n correspond au nombre d’espaces du décalage. | Oui |
| comments-header | on | off | Test la présence d’un commentaire multi-ligne en entête de fichier. | Oui |
| max-line-numbers | n | off | Les lignes ne doivent pas dépasser n caractères. | Oui |
| max-file-line-numbers | n | off | Les fichiers ne doivent pas dépasser n lignes. | Oui |
| no-trailing-spaces | on | off | Il ne doit pas avoir d’espace en fin de ligne. (en dehors de l’indentation classique) | Oui |

|  |  |  |  |
| --- | --- | --- | --- |
| **Règles sur les variables et fonctions** | | | |
| **Règle** | **Valeurs possibles** | **Description** | **Disponible** |
| no-multi-declaration | on | off | Il ne doit pas avoir plusieurs déclarations de variable sur une même ligne. | Non |
| unused-variable | on | off | Il ne doit pas avoir de variable inutile dans le code. | Non |
| undeclared-variable | on | off | Il ne doit pas avoir de variable non déclarée dans le code et manipulée quand même. | Non |
| no-prototype | on | off | Les fonctions doivent toutes avoir un prototype dans le fichier. | Non |
| unused-function | on | off | Il ne doit pas avoir de fonction inutile dans le code. | Non |
| undeclared-function | on | off | Il ne doit pas avoir de fonction non déclarée dans le code et déclenchée quand même. | Non |

|  |  |  |  |
| --- | --- | --- | --- |
| **Règles sur les Vérifications de type** | | | |
| **Règle** | **Valeurs possibles** | **Description** | **Disponible** |
| variable-assignment-type | on | off | Il ne doit pas avoir de problème de type lors de l’affectation d’une valeur dans une variable. | Non |
| function-parameters-type | on | off | Il ne doit pas avoir de problème de type lors de passage de paramètres dans une fonction. | Non |

## **Le fichier de configuration**

Les différentes règles sont activables dans un fichier de configuration (extension .lconf).

**Voici le format du fichier :**

=KEY

VALEUR # Valeur de KEY

=KEY2

- VALEUR # Éléments de KEY2

- VALEUR

- VALEUR

=KEY3

- E1 = VALEUR

- E2 = VALEUR

- E3 = VALEUR

KEY, KEY2 et KEY3 sont les clés correspondantes aux options de votre Linter, c’est une

chaine de caractères uniques permettant d’identifier une option.

VALEUR correspond au contenu derrière la première clé, elle peut être soit une chaine de

caractères, soit un entier, soit un boolean.

E1, E2, E3 correspondent aux éléments contenus derrière une autre clé.

**Ci-dessous l’ensemble des clés principales :**

* **extends**: Héritage possible d’un autre fichier lconf (Permet de récupérer des options depuis un autre fichier de configuration) .
* **rules**: Les règles de votre Linter .
* **excludedFiles**: Les fichiers à ne pas inclure lors de la lecture du dossier en cours .
* **recursive**: Le parcours se fera de manière récursive, le Linter devra s’exécuter aussi dans les sous dossiers.

# **Aspect Technique**

## **Les Structures utilisés**

La définition et la liste des fonctions manipulant les structures sont dans le fichier structure.h .

### **La structure RuleLinter :**

Cette structure définit une règle et sa valeur.

|  |  |  |
| --- | --- | --- |
| **Sous variable(s)** | **Type** | **Description** |
| key | Char\* | Contient le nom de la règle |
| value | Char\* | Contient la valeur concernant la règle. (on | n | off) |

### **La structure ConfigLinte**r :

Cette structure définit la configuration du linter récupérer pas le(s) fichier(s) de configuration(s) .

|  |  |  |
| --- | --- | --- |
| **Sous variable(s)** | **Type** | **Description** |
| fileExtend | Char\* | Indique le nom du dernier fichier de configuration hériter |
| listRules | RuleLinter\*\* | Contient la liste des règles |
| countRules | int | Contient le nombre de règles |
| llistExcludedFiles | Char\*\* | Contient la liste des fichier(s) source(s) à ignorer |
| countExcludedFiles | int | Contient le nombre de fichier(s) source(s) à ignorer |
| recursive | short | Prend la valeur 1 si le linter doit analyser le(s) sous-dossier(s), sinon prend la valeur 0 |

# **Procédure d’Installation**

Le linter sera livré au format zip, il suffira de le dézipper.

Le linter sera livré avec un une application prête à être utilisé (llinterC.exe) dans le dossier bin. Toute fois si vous souhaité ajouter de modifications (règles ou fonctionnalités dans les fichiers sources) et recompiler le projet, cela est tout à fait possible mais présente des prérequis.

Pour cela, il vous faudra installer plusieurs outils :

* **GCC** : le compilateur pour le langage C.
* **make** : un outil de compilation de projet/ application.
* **cmake** : un outil de génération de fichier de construction standard, il va générer les fichiers nécessaire à la compilation d’une application.

Je vous invite à consulter au moin les pages wikipédia pour en savoir un peu plus sur ces outils :

GCC : <https://fr.wikipedia.org/wiki/GNU_Compiler_Collection>

make : <https://fr.wikipedia.org/wiki/Make>

cmake : <https://fr.wikipedia.org/wiki/CMake>

## **Sous Windows**

Vous allez télécharger l’installateur MinGW et suivre les démarches pour installer les outils **gcc** et **make** .

Lien De l’installateur MinGW: <https://osdn.net/projects/mingw/releases/>

Tutoriel d’installation MinGW : <https://www.youtube.com/watch?v=DHekr3EtDOA>

Pour **CMake**.

Lien de l’installateur : <https://cmake.org/download/>

Tutoriel d’installation CMAKE : <https://www.youtube.com/watch?v=8_X5Iq9niDE>

**Ajout des outils dans la variable d’environnement PATH.**

Selon les versions de Windows cette étape peut diverger un peu.

Pour les utilisateurs de Windows 10 : <https://www.youtube.com/watch?v=83SccoBYSfA>

Pour les utilisateurs de Windows 8, 8.1 : <https://www.youtube.com/watch?v=W9pg2FHeoq8>

Pour les utilisateurs de Windows 7, XP, Vista : <https://www.youtube.com/watch?v=vN2RZfkrqKc>

## **Sous Linux**

Pour les outils **gcc** et **make** .

Un **apt-get install build-essential (installe les compilateurs GCC, G++ ; installe make ; installe également les Library standards pour le développement C).**

Pour l’outil **CMake.**

Un **apt-get -y install cmake.**

**Ajout des outils dans la variable d’environnement PATH.**

Lien tutoriel écrit : <https://www.cyberciti.biz/faq/set-environment-variable-linux/>

## **Compilation du projet**

Pour la compilation du projet, un script est à construire.

Pour **Windows** :

Il s’agit du fichier compilation.cmd (il est déjà construit).

Pour **LINUX** :

Il faut construire un script shell qui reprend les mêmes commandes que compilation.cmd.

Attention :

La ligne cmake .. -G "MSYS Makefiles».

Doit être remplace part : cmake .. -G "Unix Makefiles".

Votre projet devrait être compiler, il vous suffit de lancer l’application qui se trouve dans le dossier bin.

# **Procédure d’utilisation**

Vous devez préciser en argument au linter, le dossier contenant le(s) fichier(s) source(s) C a analysé.

Dans ce dossier, il devra se trouver un fichier de configuration. lconf pour activer les règles que vous souhaités vérifier.

Exemple sous Windows : linterC.exe C:/Dosssier1/DossierContenantDesFichiersAAnalyser



# **Bilan du projet**

Ressentit du projet par élève.

## **Allan MAUBERT**

### **Difficultés techniques et humains rencontrées**

**Le Code** :

La construction des fichiers de builds, ne sachant pas comment construire un makefile pour build un projet, je me suis documenté en conséquence, néanmoins je n’arrivais pas à faire ce que je voulais. J’ai donc opté pour l’utilisation de cmake et avec l’aide d’un script de gérer les fichiers de build, le binaire générer et les sources. La partie de build, m’a pris un temps assez conséquent (entre 15 et 20 heures) pour arriver à un résultat peu satisfaisant.

La récupération des informations du fichier de configuration dans une structure m’a posé pas mal de problème à cause notamment de l’allocation dynamique de mémoire.

La difficulté des règles de la partie 2 était abordable, mais celles de la partie 3 étaient d’un tout autre niveau car il fallait faire beaucoup d’étapes (fonctionnalité) pour arriver à un résultat.

Un exemple concret : il fallait impérativement distinguer toute la liste des fonctions dans un fichier et leur contenu en mémoire pour pouvoir les analyser une par une. Cette fonctionnalité était un pré-requis (assez difficile) pour les règles de la partie 3. Je ne parlerai pas de la partie 4 car on a même pas réussi une seule règle de la partie 3.

**Gestion du temps** :

Le temps a été très mal gérer puisque la totalité des membres du groupes à commencer à travailler au bout d’1 mois et qqe jours après le début du projet.

Un outil de planification des tâches avec des deadline ou des mini objectifs à atteindre aurait très certainement changer la donne.

**Le groupe** :

La communication a été assez difficile au sein du groupe, car on ne sait pas assez concerter au début et surtout mis d’accord sur une façon de procéder suffisamment claire pour démarrer le projet sur de bonne condition.

Le workflow n’a pas été respecté, nous voulions opter pour un workflow type **pull request** avec git et github car tous les membres ne savent pas utiliser suffisamment ces outils.

Quant à la disponibilité des membres : très insatisfaisant, sachant que l’un des membres à des problèmes avec son téléphone portable depuis quelque mois. La communication n’a pas été notre fort, nous avons, à peine, fait une ou deux réunions sur la planification de projet.

A l’avenir se sera l’un des plus grands axes à corriger.

**Résumé :**

Je suis globalement insatisfait du résultat car l’objectif (toutes les fonctionnalités du Linter) n’a pas été atteint et de très loin. Le temps que j’ai passé sur ce projet n’a pas été assez productif (à mon avis). L’implication des membres de mon équipe ne se voyait pas suffisamment (Partager des infos sur ces fonctionnalités et mettre au courant les autres sur l’avancement de son travail de façon régulière est important !).

Néanmoins, j’en tire aussi beaucoup de bonne chose, j’ai appris beaucoup sur comment on construit un projet en langage C, le build, l’environnement de travail, Le scripting pour l’organisation des fichiers de builds ; je sais désormais ce qu’il ne faut pas refaire comme erreur et ce que je peux améliorer au sein d’un travail d’équipe.

## **Guillaume CASSOLY**

### **Difficultés techniques et humains rencontrées**

## **Hugo THERAL**

### **Difficultés techniques et humains rencontrées**