**Projet C Linter**

Description fonctionnelle

Main

Le main se présente sous la forme d’un double while, représentant un menu et un sous menu (validation de sortie ou rechargement de la configuration).

Il s’occupe essentiellement de :

* Charger le/les fichier de configuration
  + Vérification d’existence (+ non vide) du fichier de conf initial via la fonction dispErrMessg (la fonction vérifie le fichier cible via la fonction isExistingFile), la fonction s’occupe de gérer l’affichage des erreurs et sa valeur de retour définis le reste des actions
  + Ensuite, si un héritage est déclaré dans le fichier de conf, on vérifie à nouveau l’existence du fichier cible et cela définira la suite des actions. Qu’aucun héritage ne soit déclaré ou que celui-ci soit inaccessible, les actions sont les mêmes. Les fonctions d’agrégat n’interviennent que dans le cas où un héritage utilisable est déclaré.
  + On décompte ensuite, juste avant le menu, le nombre de règles actives.
* Proposer un menu avec différentes actions
  + Le menu propose un affichage de l’état de la configuration grâce aux différentes valeurs récupérées.
  + Il propose actuellement 4 actions mais seulement deux d’entre elles sont utilisables pour le moment : « 2 – Lister les fichiers ciblées par le Linter » ET « 3 – Exécution du Linter sur un fichier test : ‘exemple.c’ »
  + Le listing exécute un parcours récursif descendant de l’arborescence du dossier en cours.
  + L’exécution sur un fichier permet d’afficher le code source cible en console avec les warning (règles actives) associées à chaque ligne
* Free() les allocations mémoires avant relancement
  + Si l’utilisateur sélectionne « 0 – Quitter », il arrivera ensuite sur une demande de confirmation (Relancer ou Quitter programme). Si la relance est sélectionnée, les allocations mémoires sont libérées et le processus de chargement de la configuration se relance.

Toutes les fonctions ne seront pas décrites, de nombreux processus mis en place se ressemblent nous aborderons les plus complexes uniquement.

Fonction getConfRules(char\* fileContent)

Cette fonction a pour objectif d’extraire les règles présentes dans le fichier de configuration. Pour cela, nous avons mis en place un traitement caractère par caractère.

Ce traitement peut se découper en plusieurs tâches définis :

* On initialise le tableau de règles (16 règles) à -1 partout, afin de pouvoir différencier les règles absentes des fichiers de conf et les règles présentes mais en OFF (valeur 0).
  + Remarque : les valeurs -1 permettent entre autres de savoir quand écraser ou non une valeur existante lors d’un héritage de configuration -> voir agregateRulesValues().
* Viens ensuite une boucle sur chaque caractère du fichier, une variable d’état (searchingState) est utilisée pour repérer la zone du fichier sur laquelle on va extraire les données.
  + Si =rules est repéré, searchingState = 1
  + Si =xxx est repéré (sans espace après le ‘=’, contrairement aux déclarations de règles (‘ = ‘), searchingState = 0
* Lorsque searchingState = 1, on vérifie pour chaque caractère si celui-ci est égal à ‘-‘ et si le suivant est un espace. Si c’est le cas alors on sait que l’on se place en début de ligne d’une règle (i = i + 2 pour se placer sur le caractère juste après l’espace)
* On va ensuite extraire la chaine de caractère relative à la règle, (tant que caractère courant différent d’un espace, extraction)
* Une fois l’extraction terminée, on place un ‘\0’ à la fin de notre extraction et on décale i sur le nombre de caractère que l’on veut (i = i + x + 3, x étant le nombre de caractère de l’extraction et 3 représentant les deux espaces et l’égal séparant le nom de la règle et sa valeur associée)
* On compare ensuite notre chaine extraite avec chaque noms de règle afin de repérer sur laquelle on se place (currRule)
* Viens ensuite l’extraction et l’interprétation de la valeur associée, pour cela le programme repère des on/ON, off/OFF ou des valeurs autres (doivent être numériques pour que cela fonctionne). L’interprétation des valeurs se fait ainsi :
  + On = 1
  + Off = 0
  + Valeur = valeur (on prend la valeur ASCII – 48 et on formate le nombre grâce à des multiplications par 10)
* La fonction retourne ensuite le tableau (pointeur alloué dans la fonction) de valeurs au main.

La logique utilisée dans cette fonction est similaire aux développements mis en place pour les différentes extractions d’informations. La réelle différence se situe dans la syntaxe des informations représentées dans le fichier de configuration (héritage sans le tiret + espace en début de ligne, par exemple).

Les fonctions d’agrégats ont une façon de faire différente selon les éléments, les règles se « fusionnent » tandis que les fichiers exclus « s’additionnent ».

Fonction getNbExtendLayer(char\* fileContent)

Cette fonction a pour objectif de décompter le nombre d’héritage utilisable dans la chaine de fichier de configuration. Elle commence par ouvrir l’héritage présent dans main.lconf (s’il y en a un) puis va aller récupérer chaque nom de fichier de configuration un à un, en vérifiant chaque fois que le fichier existe et n’est pas vide.

Pour cela elle utilisera plusieurs fonctions bien connus de notre programme (dispErrMessg, getConfExtends, getNbFilesExtended, getFileContent).

Fonction getAllExtendedFiles(……)

Cette fonction utilise le nombre décompté par getNbExtendLayer afin de récupérer, dans la limite de l’utilisable, tous les noms de fichiers de conf qui seront utilisés pour établir la configuration du programme (héritage). L’ordre des fichiers dans le tableau de chaînes est inversé par rapport à l’ordre normal pour l’héritage, le traitement des différents fichiers se fera donc de la fin du tableau vers le début.

Fonction lookForType(char\* fileContent, int startInd)

Cette fonction est utilisée dans les repérages de couples { et } pour le parsing du fichier, elle permet de repérer si l’accolade trouvée est relative à :

* Une déclaration de tableau
* Une déclaration de fonction
* La syntaxe du main

Son traitement permet donc de cibler les bonnes accolades dans le traitement, en isolant celles présentes dans le fichier mais n’étant pas pertinente pour notre recherche de niveaux.

**Parsing**

Nous allons maintenant parler un peu du parsing de chaque fichier, les méthodes et logiques utilisées.

Voici la structure de données utilisée, lineLevels :

typedef struct lineLevels lineLevels;

struct lineLevels {

int startLevel;

int startingLine;

int endLevel;

int endingLine;

char\*\*\* declaredVars

int\* nbVars;

lineLevels\* sonStructs;

int nbSons;

};

Le parsing fonctionne actuellement grâce à 5 fonctions :

* getNbPrimaryLevels : décompte le nombre de niveaux primaires du fichier, pour cela nous repérons et décomptons les couples { et } mais sans décompter les couples internes aux couples (appelés primaires)
* getAllLevels : utilise le nombre de niveaux primaires donnés par sa fonction associée, ce nombre est utilisé pour l’allocation mémoire du tableau de structures. La logique de repérage des couples est la même mais la fonction s’occupera également de créer des structures en définissant leurs attributs (caractère de début, ligne de début, caractère de fin, ligne de fin, nombre de fils et tableau des structures filles) – le décompte des variables et leurs extractions est toujours en cours de développement.
* getNbInsidersLevels : décompte le nombre de niveaux primaires à l’intérieur d’une structure, la logique utilisée est la même que pour getNbPrimaryLevels mais la fonction ne traitera que l’intervalle (caractère de début et de fin) du niveau sur lequel elle fait son traitement.
* getInsidersLevels : appelé par getAllLevels pour définir le tableau de structure fille de chaque structure, elle s’appelle elle-même lorsque des structures filles (petites-filles) sont présentes dans la structure fille courante. Ses actions sont les mêmes que getAllLevels mais son action est limitée à l’intervalle de sa structure parente.
* La dernière fonction utilisée est levelVarCount et celle-ci est toujours en cours de développement. La difficulté dans cette fonction va être de repérer de façon exacte les déclarations de variables (logique similaire à onlyOneDeclar) et de les décompter correctement (repérer les déclarations de tableau, multiples ou non, en séparant les noms de vars des chaines contenus dans le tableau par exemple, séparées tout deux par une virgule).
  + Remarque : la difficulté réside également dans le repérage des intervalles d’indices relatifs au niveau (s’occuper du niveau mais pas de ses fils, en s’occupant tout de même des intervalles entre chaque fils)