HEIG-VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Prof. Yann Thoma

Exercices du cours VSE
Exercices de vérification SystemVerilog
Couverture fonctionnelle

semestre automne 2024 - 2025

Pour pouvoir effectuer cet exercice il faut avoir `clang-tidy`, `clang-format` et `scan-build` à disposition. Pour ce faire vous pouvez les installer avec

```
sudo apt install clang-tidy
sudo apt install clang-format
sudo apt install clang-tools
```

La documentation de `clang-tidy` est disponible ici : `https://clang.llvm.org/extra/clang-tidy/` et la liste des vérifications disponibles ici : `https://clang.llvm.org/extra/clang-tidy/checks/list.html`.

Exercice (scan-build)

Créez un répertoire `build2` et placez-vous dedans.
Exécutez la commande

```
scan-build cmake ..
scan-build make
```

Ensuite, vous pouvez ouvrir le fichier `index.html` généré ou utiliser

```
scan-view /tmp/scan-build-...
```

Essayez d'ajouter des bouts de code que vous savez incorrects et voyez si `scan-build` les détecte.

Exercice (clang-tidy)

Reprenez le code fourni, qui exploite la double liste chainée et l'arbre syntaxique.
Trois scripts sont fournis :

1. `checkcode.sh` : Il effectue une vérification définie dans le script lui-même
2. `checkcodeimplicite.sh` : Il effectue les vérifications définies dans le fichier `.clang-tidy`
3. `checkcodefull.sh` : Il effectue les vérifications définies dans le script lui-même, sans les noms, mais plus de contraintes

Les scripts génèrent un fichier `clang-warnings.txt`, et deux résumés dans `clang-warnings-shorts-h.txt` et `clang-warnings-shorts-cpp.txt`
Observez leur contenu après le lancement de chacun des scripts. Etes-vous d'accord avec tous ces *warnings* ? En voudriez-vous d'autres ? La liste des vérifications possibles vous montre l'ensemble des possibles.

Exercice (clang-format)

Lorsque plusieurs personnes travaillent sur un même code source, il est difficile de garantir que tout le monde suive les lignes directrices de codage. `clang-format` permet de vérifier si le code est compatible avec les conventions, et peut également corriger le code automatiquement.
Pour faire un essai, copiez le répertoire `code` dans `code2`, puis placez-vous dans `code2`.
Lancez la commande

```
clang-format --dry-run *.h *.cpp
```

Celle-ci vous donner en sortie les erreurs repérées.
Vous pouvez ensuite faire modifier automatiquement le code avec la commande suivante :

```
clang-format -i *.h *.cpp
```

Pour voir les modifications vous pouvez utiliser un logiciel de comparaison comme `meld` par exemple, afin de comparer le contenu de `code` et `code2`.
Si vous disposez d'un fichier `.clang-format` dans un répertoire racine, alors vous pouvez exploiter la commande suivante pour appliquer les corrections sur tous les fichiers pertinents :

```
find . -regex '.*\.\(cpp\|hpp\|cc\|cxx\|h\)' -exec clang-format -style=file -i {} \;
```

A titre gracieux voici un fichier `pre-commit` que vous pouvez placer dans `.git/hooks`, si vous travaillez avec git et C++. Il permet de lancer automatiquement une analyse au moment du commit et interdit celui-ci si le code n'est pas compatible. Ceci part toutefois du principe que vous disposez d'un fichier `.clang-format` à la racine de votre arborescence.

```
#!/bin/sh
# This hook checks if the files formating is valid.
# If not, it displays the warnings as errors and refuses the commit

OK=true
for FILE in $(git diff --cached --name-only | grep -E '.*\.(c|cpp|h|hpp)')
do
if clang-format --dry-run -Werror -i $FILE; then
  :
else
  OK=false
fi
done

if $OK; then
  echo "clang-format successful"
  exit 0
else
  echo "clang-format errors. Please modifiy your code to meet the requirements"
  exit 1
fi
```