Opdrachten kwartaal 3 week 6 – Coding standards en static code analysis

Tot nu toe hebben we dynamische test-tools ingezet om de kwaliteit van onze C-code te verbeteren. Er zijn ook zogenoemde statische analyse-tools beschikbaar die de kwaliteit van de code beoordelen zonder deze code uit te voeren. In feite heb je die ook al gebruikt, de compiler geeft namelijk waarschuwingen (Engels: warnings) als je bepaalde taalconstructies gebruikt, die geldig zijn in C, maar die vaak tot fouten leiden. Om deze reden laten we de C-compiler zoveel mogelijk warnings genereren[1].

Je leert deze les hoe je:

- gebruik kunt maken van verschillende *C coding standards*;
- gebruik kunt maken van verschillende *static code analysis* tools, waaronder Cppcheck, om de kwaliteit van je C-code te verbeteren.

Bij het ontwikkelen van C-code is het vaak handig of noodzakelijk om gebruik te maken van een zogenoemde codeerstandaard (Engels: coding standard). Een C coding standard is niet meer of minder dan een lijstje do's en don'ts voor het gebruik van de programmeertaal C. Veel bedrijven of bedrijfstakken maken gebruik van een specifieke coding standard. Zo maakt de auto-industrie bijvoorbeeld gebruik van MISRA C[2]. De MISRA C coding standard is helaas niet gratis verkrijgbaar.

Het Software Engineering Institute van de Carnegie Mellon University heeft in 2016 een uitgebreide C coding standaard uitgegeven: *SEI CERT C Coding Standard Rules for Developing Safe, Reliable, and Secure Systems*[3].

Het Information-technology Promotion Agency uit Japan heeft in 2018 een uitgebreide C coding standaard uitgegeven: *Embedded System development Coding Reference guide [C Language Edition]*[4].

---

[1] In week 3.1 hebben we je aangeraden om de compiler opties `-std=c18 -Wall -Wextra -pedantic-errors` te gebruiken.

[2] Zie: https://misra.org.uk/product/misra-c2023/

[3] Zie: https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=454220. Je kunt deze coding standard downloaden vanaf: https://bitbucket.org/HR_ELEKTRO/ems31/wiki/Documenten/SEI_CERT_2016_C_Coding_Standard.pdf.

[4] Zie: https://www.ipa.go.jp/publish/qv6pgp00000011mh-att/000065271.pdf. Je kunt deze coding standard downloaden vanaf: https://bitbucket.org/HR_ELEKTRO/ems31/wiki/Documenten/IPA_2018_Embedded_System_development_Coding_Reference_guide.pdf.

De Barr-group heeft in 2018 een vrij compacte coding standaard uitgegeven speciaal gericht op het ontwikkelen van C-code voor embedded systems: *Embedded C Coding Standard*[5].

Er bestaan veel statische analyse-tools waarmee gecontroleerd kan worden of C-code aan bepaalde coding standards voldoet[6].

In deze les maken we gebruik van Cppcheck[7], een veelgebruikte statische analyse tool voor C en C++. Cppcheck is te gebruiken als losse tool, maar er is ook een plug-in beschikbaar voor Visual studio code[8].

Het programma `foutje.c` bevat de volgende functie:

```
// Bepaal of pointer pi naar een positief getal wijst
bool isPos(int *pi)
{
    return *pi > 0 && pi != NULL;
}
```

De functie `is_pos` geeft **`true`** terug als de integer waar de parameter `pi` van het type **`int`** `*` naar wijst, positief is. De fuctie geeft **`false`** terug als dit niet zo is. Als de als argument meegegeven pointer `NULL` is, dan wordt ook **`false`** teruggegeven (althans dat is de bedoeling van de programmeur). Deze functie bevat een verraderlijke fout. De &&-operatie wordt namelijk altijd van links naar rechts uitgevoerd. De pointer wordt dus eerst gebruikt als verwijzing[9] voordat gecontroleerd wordt of de pointer niet `NULL` is. De operatie `*pi` zal een zogenoemde '*segmentation fault*' opleveren als `pi` gelijk aan `NULL` is en het programma zal worden afgebroken.

De gcc C-compiler geeft hier geen warning voor. Maar Cppcheck waarschuwt je wel voor deze fout, zie figuur 1.

---

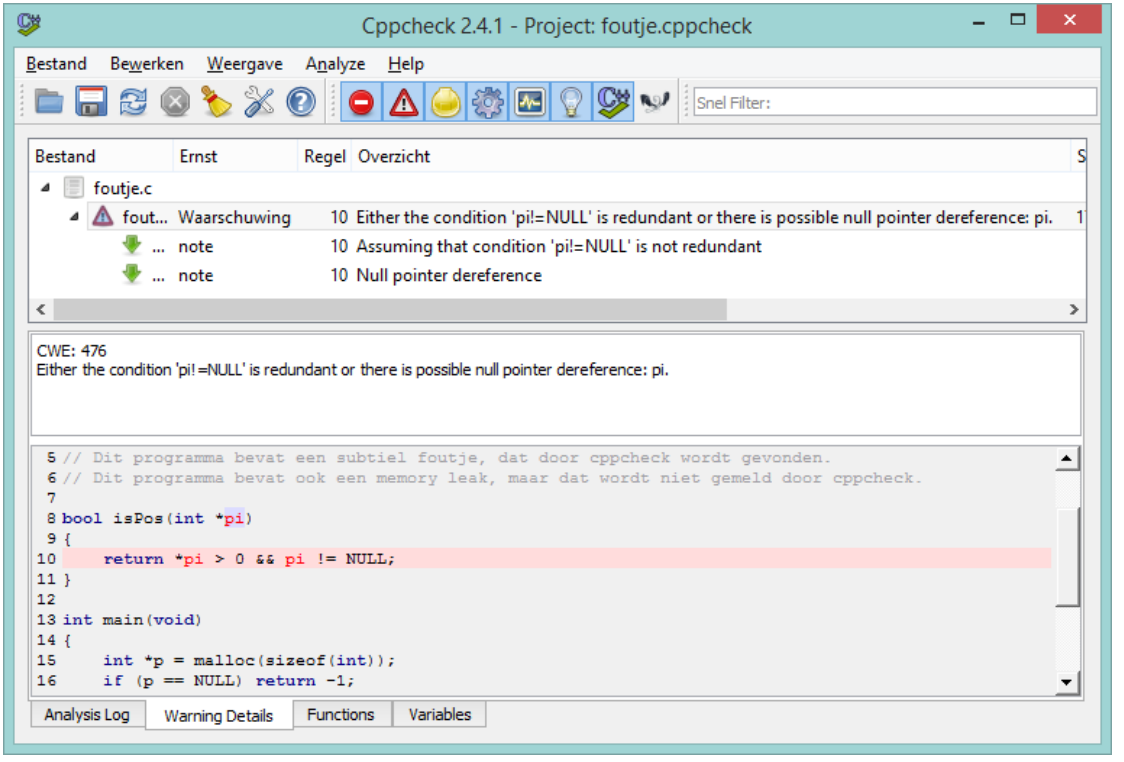
[5] Zie: https://barrgroup.com/embedded-systems/books/embedded-c-coding-standard. Je kunt deze coding standard downloaden vanaf: https://bitbucket.org/HR_ELEKTRO/ems31/wiki/Documenten/Barr_Group_2018_Embedded_C_Coding_Standard.pdf.

[6] Een overzicht kun je hier vinden: https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#C,_C++.

[7] Zie http://cppcheck.sourceforge.net/.

[8] Zie: https://marketplace.visualstudio.com/items?itemName=NathanJ.cppcheck-plugin.

[9] Het gebruik van de *-operator om te verwijzen (refereren) naar de variabele waar de pointer naar wijst wordt in het Engels een 'pointer dereference' genoemd.



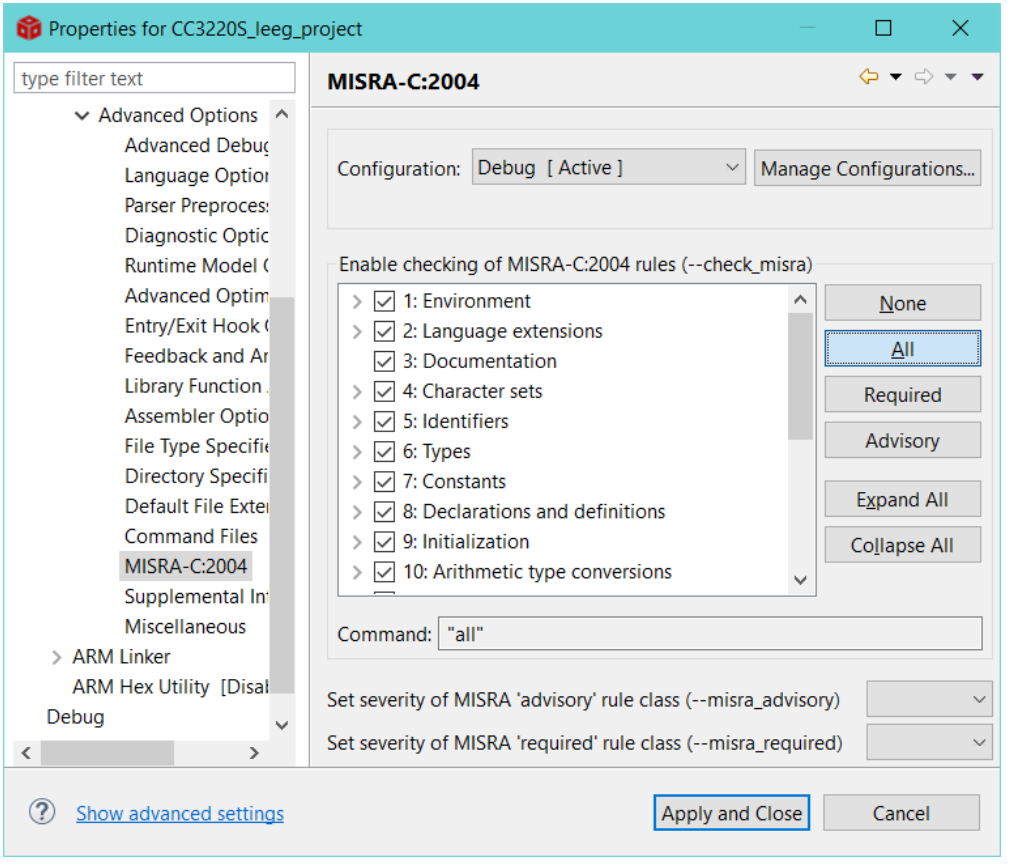
Figuur 1: Het programma Cppcheck waarschuwt voor een mogelijke fout.

De juiste implementatie van de functie `is_pos` is als volgt:

```
bool isPos(int *pi)
{
    return pi != NULL && *pi > 0;
}
```

Als deze code wordt getest door Cppcheck, dan worden geen waarschuwingen meer gegeven. De code is inderdaad correct, want de `&&`-operator moet volgens de C-standaard gebruik maken van zogenoemde 'lazy evaluation'. Dat wil zeggen dat de `&&`-operator niet verder uitgevoerd wordt dan strikt noodzakelijk is. Dus, als de pointer `pi` gelijk is aan `NULL`, dan is de eerste operand van de `&&`-operator (`pi != NULL`) gelijk aan **`false`**. De tweede operand (`*pi > 0`) wordt dan niet meer uitgevoerd, want **`false`** `&&` `iets` is altijd **`false`** dus het is niet nodig om de waarde van `iets` te bepalen. Op deze manier wordt de pointer `pi` dus niet gebruikt als verwijzing, als de waarde van de pointer `NULL` is.

Code Composer Studio bevat een static code analysis tool waarmee gecontroleerd kan worden of de code aan de MISRA-C standaard (versie 2012) voldoet, zie figuur 2. De voorbeeld en library-code die wordt meegeleverd met CCS voldoet echter niet aan deze standaard.



Figuur 2: CCS heeft een ingebouwde tool voor static code analysis.

3.6.1 Analyseer alle code die je tot nu toe hebt ontwikkeld met behulp van Cppcheck. Verbeter de code indien nodig.

Maak deze week al een start met de eerste eindopdracht: https://bitbucket.org/HR_ELEKTRO/ems31/wiki/Opdrachten/Eindopdracht_1.pdf