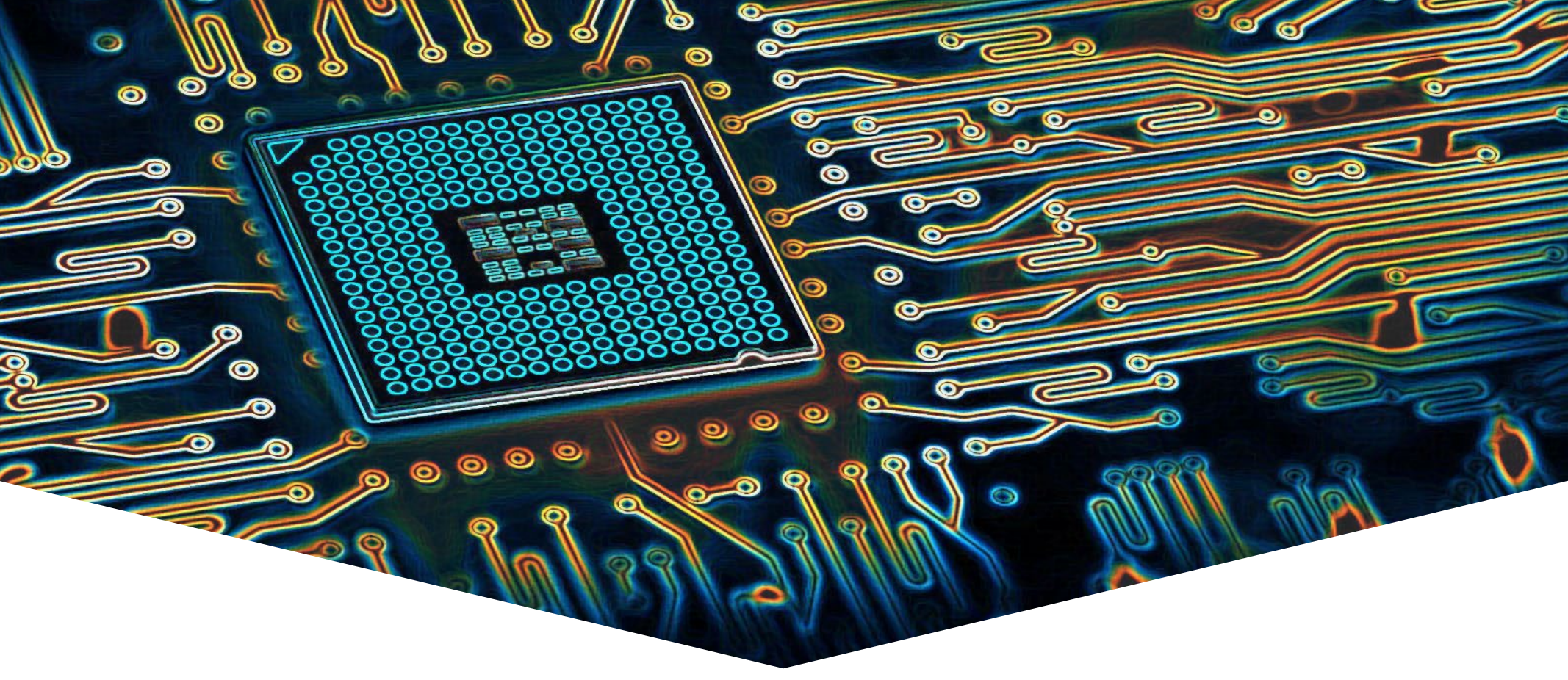


EMS31 Kwartaal 3 Week 5:
Memory tests en test coverage

Memory tests en test coverage

Leerdoelen kwartaal 3 week 5. Je leert hoe je:

- fouten bij het gebruik van dynamische geheugenallocatie zoals geheugenlekken kunt opsporen met speciale tools zoals valgrind;
- weet dat je genoeg testcode hebt geschreven door gebruik te maken van test coverage metingen.

Memory errors

Het gebruik van dynamische geheugenallocatie kan leiden tot lastig op te sporen fouten.

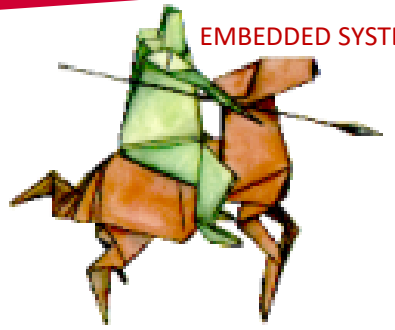
```
while (head != NULL)
{
    free(head);
    head = head -> next;
}
```

Er treedt een

Segmentation fault op!

https://bitbucket.org/HR_ELEKTRO/ems31/raw/master/Programmas/linked_list_dynamic_error/linked_list_dynamic_error.c

Tools om memory errors op te sporen



Linux: Valgind

- https://valgrind.org/docs/manual/quick-start.html
- `valgrind --leak-check=full --show-leak-kinds=all ./linked_list_dynamic_error`

Windows: DrMemory

- https://drmemory.org/

Valgrind voorbeeld


```c
#include <stdlib.h>

int main(void) {
    int *x = malloc(10 * sizeof(int));
    x[10] = 0;
    return 0;
}
```

https://bitbucket.org/HR_ELEKTRO/ems31/src/master/Programmas/valgrindMemcheck/stupid.c

```
// veel output waaronder:
Invalid write of size 4 at 0x109153: main (stupid.c:5)
40 bytes in 1 blocks are definitely lost: malloc in main(stupid.c:4)
ERROR SUMMARY: 2 errors
```

https://bitbucket.org/HR_ELEKTRO/ems31/src/master/Programmas/valgrindMemcheck/stupid-output.txt

Valgrind voorbeeld

https://bitbucket.org/HR_ELEKTRO/ems31/src/master/Programmas/valgrindMemcheck/memcheck.c
https://bitbucket.org/HR_ELEKTRO/ems31/src/master/Programmas/valgrindMemcheck/memcheck-output.txt

Uitgebreider programma met voorbeelden van verschillende mogelijke fouten.

```
Conditional jump or move depends on uninitialised value(s)
   at 0x1091D0: main (memcheck.c:29)
Invalid write of size 4 at 0x1091E8: main (memcheck.c:34)
Invalid write of size 4 at 0x109204: main (memcheck.c:40)
20 bytes in 1 blocks are definitely lost
24 bytes in 1 blocks are possibly lost
40 (24 direct, 16 indirect) bytes in 1 blocks are definitely lost
LEAK SUMMARY:
   definitely lost: 44 bytes in 2 blocks
   indirectly lost: 16 bytes in 1 blocks
     possibly lost: 24 bytes in 1 blocks
   still reachable: 8 bytes in 1 blocks
```

Code Coverage

Wanneer heb je genoeg getest?

Code coverage: % van de uitgevoerde code

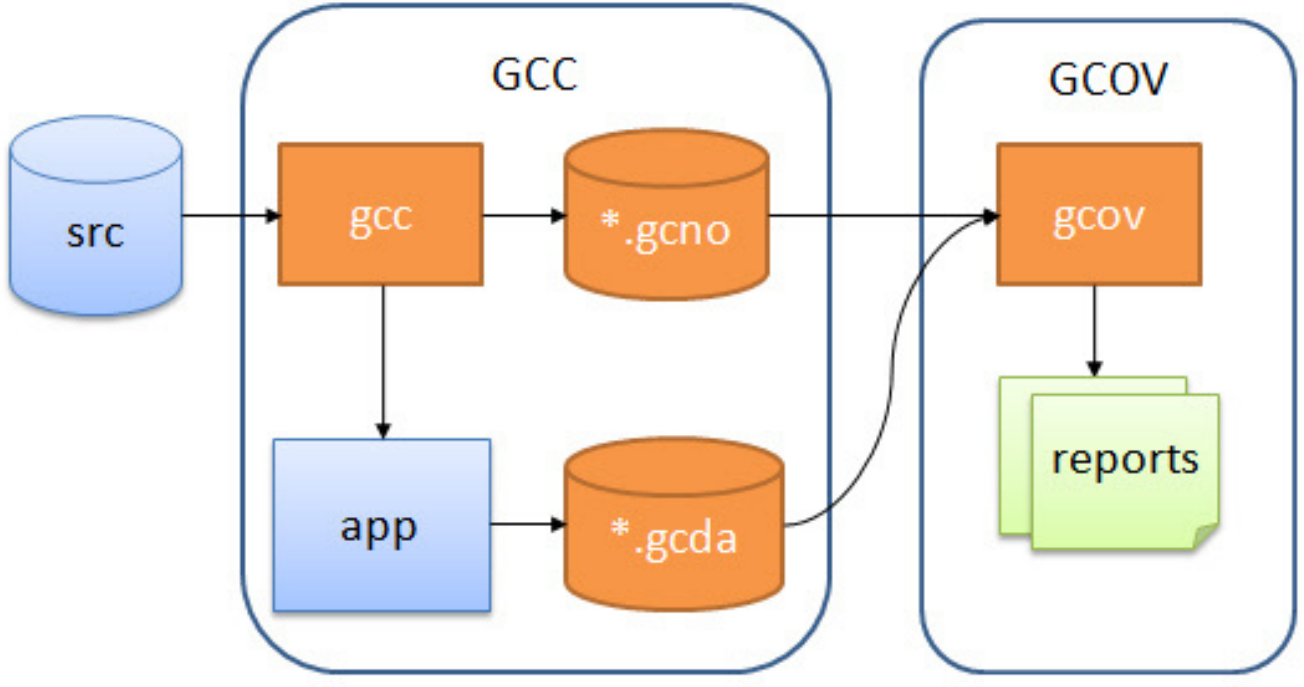
Wat zegt het?

Gebruik het om meer tests te bedenken

- Het percentage zegt niet alles

Is 100% wenselijk / haalbaar?

GCOV: onderdeel van de GCC compiler



LCOV (grafische front-end voor GCOV)

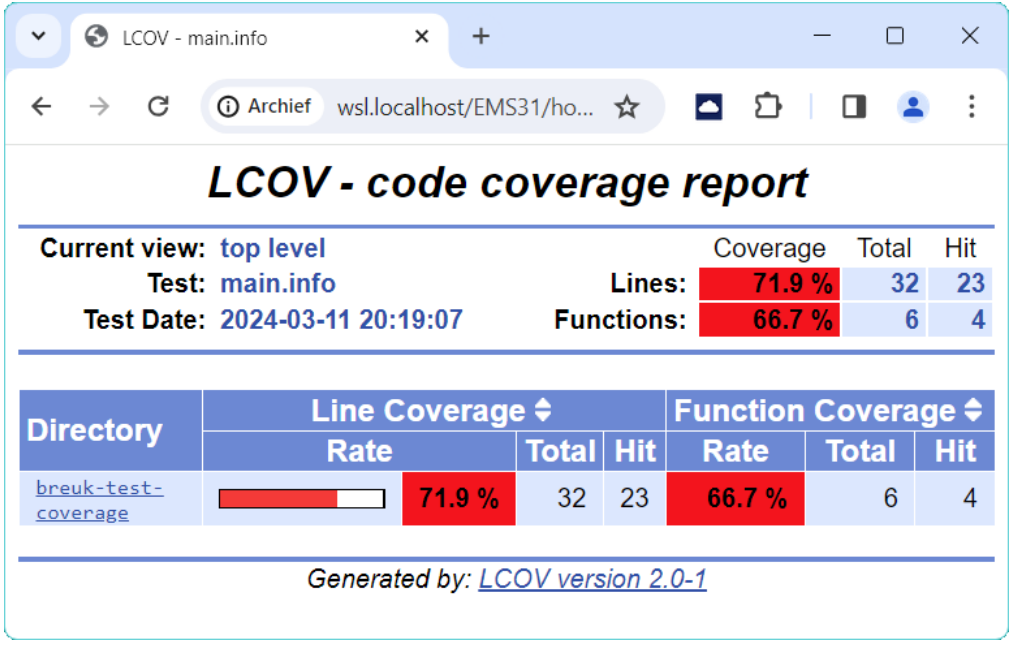
- Installeer `lcov`:
  - `sudo pacman -S lcov`
- Compile met optie `–coverage` en `run`:
  - `gcc --coverage -std=c18 -g3 -O0 prog.c`
  - `./a.exe`
- Run `lcov`:
  - `lcov --capture --directory . --output-file main.info`
  - `genhtml main.info --output-directory html`

Voorbeeld test module breuk

https://bitbucket.org/HR_ELEKTRO/ems31/raw/master/Programmas/breuk-test-coverage.zip

```
$ pwd
/home/ems/Voorbeelden/
$ wget https://bitbucket.org/HR_ELEKTRO/ems31/raw/master/Programmas/breuk-test.zip
...
$ unzip breuk-test-coverage
$ cd breuk-test-coverage
$ mkdir build
$ cd build/
$ cmake ..
-- ...
-- Build files have been written to: /home/ems/Opdrachten/breuk-test-coverage/build
$ make
[ 33%] Building CXX object CMakeFiles/test_breuk.dir/test_breuk.cpp.o
[ 66%] Building C object CMakeFiles/test_breuk.dir/breuk.c.o
[100%] Linking CXX executable test_breuk
[100%] Built target test_breuk
$ ./test_breuk
[==========] Running 3 tests from 1 test suite.
[----------] ...
[  PASSED  ] 3 tests.
$ lcov --capture --directory . --output-file main.info
...
Finished .info-file creation
$ genhtml main.info --output-directory html
...
Overall coverage rate:
  lines......: 71.9% (23 of 32 lines)
  functions......: 66.7% (4 of 6 functions)
```

Output in webbrowser

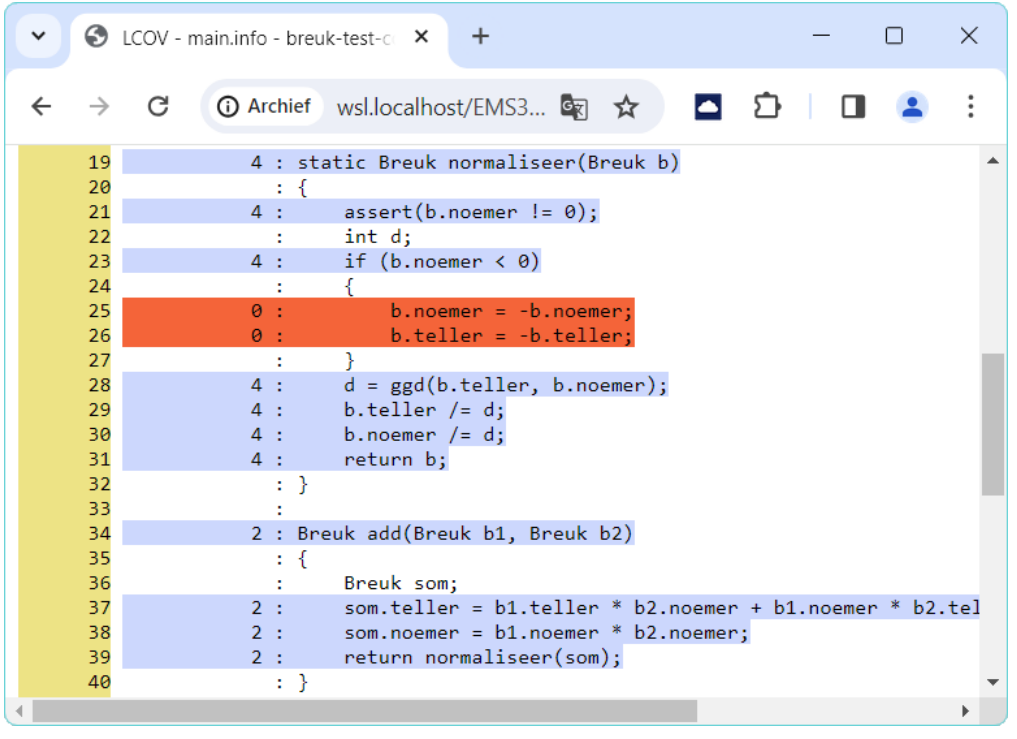


LCOV - code coverage report

			Coverage	Total	Hit
Current view:	top level				
Test:	main.info	**Lines:**	71.9 %	32	23
Test Date:	2024-03-11 20:19:07	**Functions:**	66.7 %	6	4

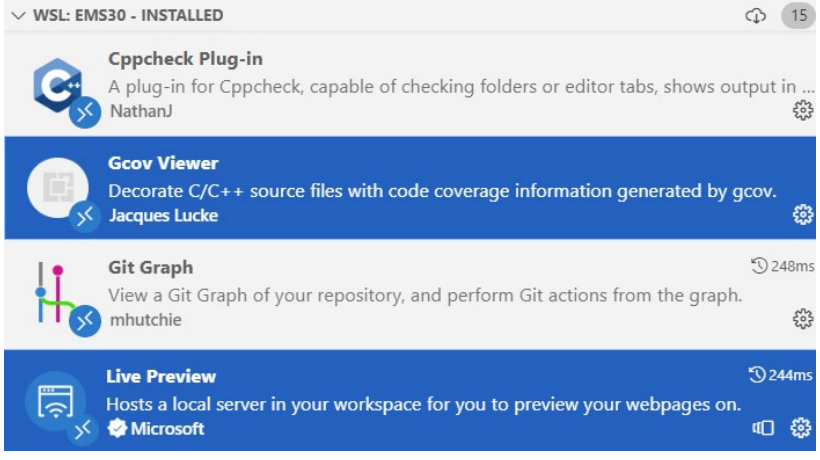
Directory	Line Coverage				Function Coverage		
	Rate		Total	Hit	Rate	Total	Hit
breuk-test-coverage		71.9 %	32	23	66.7 %	6	4

Generated by: LCOV version 2.0-1

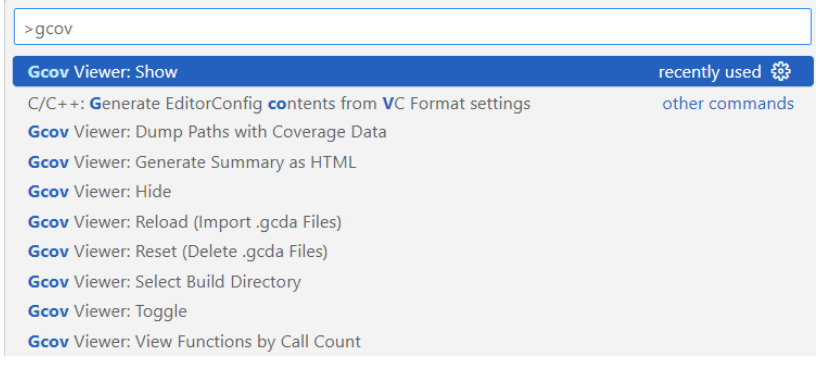



```
19          4 : static Breuk normaliseer(Breuk b)
20            : {
21          4 :     assert(b.noemer != 0);
22            :     int d;
23          4 :     if (b.noemer < 0)
24            :     {
25          0 :         b.noemer = -b.noemer;
26          0 :         b.teller = -b.teller;
27            :     }
28          4 :     d = ggd(b.teller, b.noemer);
29          4 :     b.teller /= d;
30          4 :     b.noemer /= d;
31          4 :     return b;
32            : }
33            :
34          2 : Breuk add(Breuk b1, Breuk b2)
35            : {
36            :     Breuk som;
37          2 :     som.teller = b1.teller * b2.noemer + b1.noemer * b2.tel
38          2 :     som.noemer = b1.noemer * b2.noemer;
39          2 :     return normaliseer(som);
40            : }
```

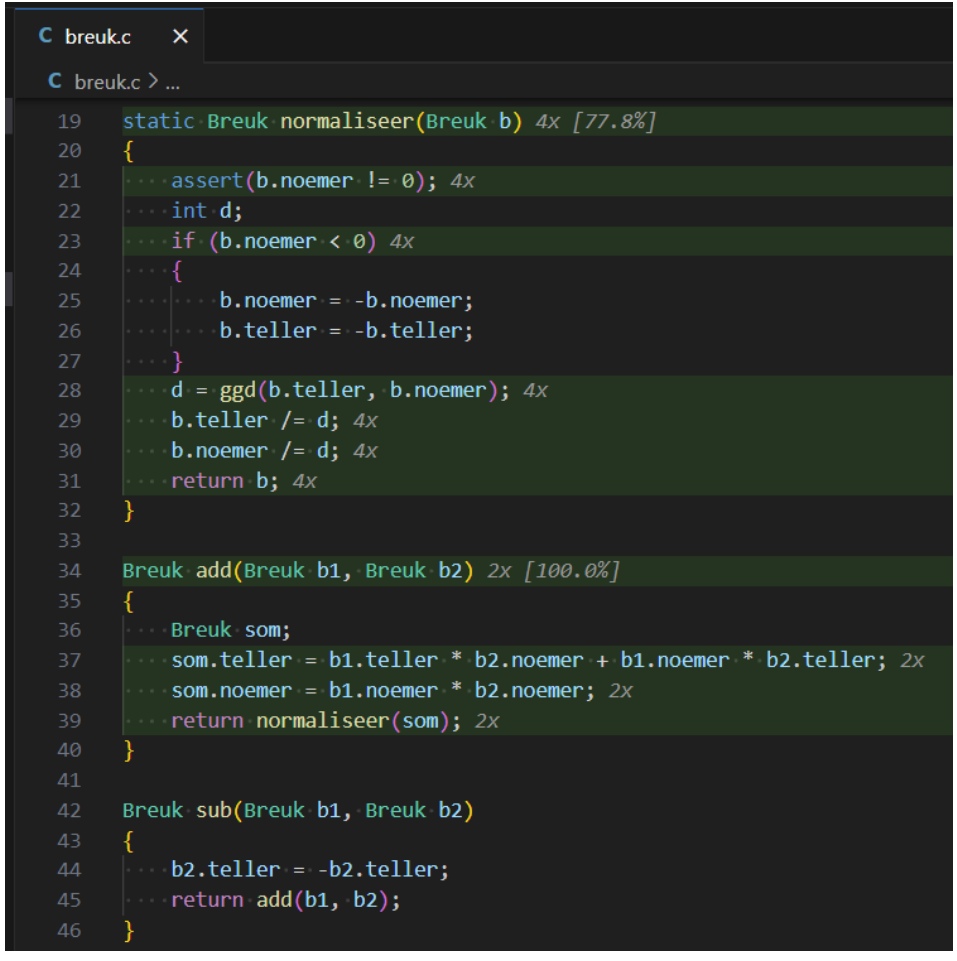
Code coverage in Visual Studio Code



https://bitbucket.org/HR_ELEKTRO/ems31/raw/master/Programmas/breuk-test-coverage.zip



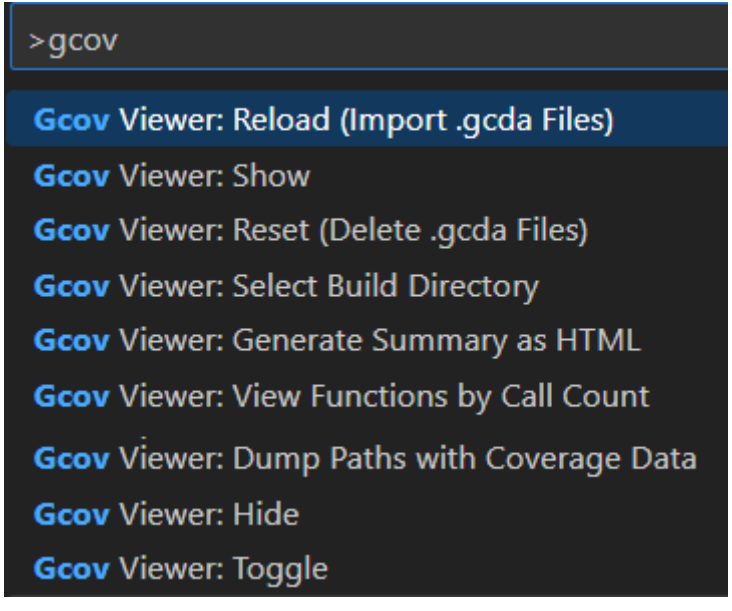
Code coverage in Visual Studio Code



Code coverage in Visual Studio Code

GCOV werkt cumulatief.

- Dus na 2x runnen zijn alle ‘hits’ 2x zo hoog.
- Gcov Viewer ‘update’ niet automatisch na een run.

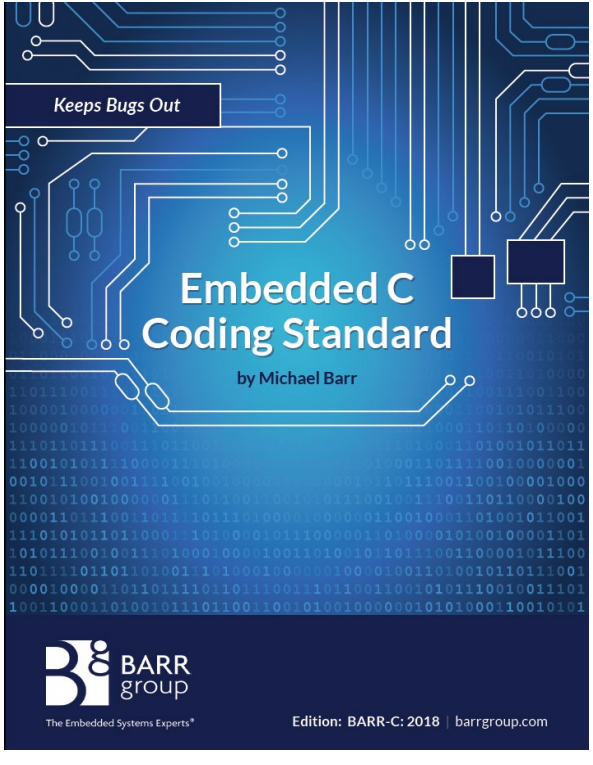


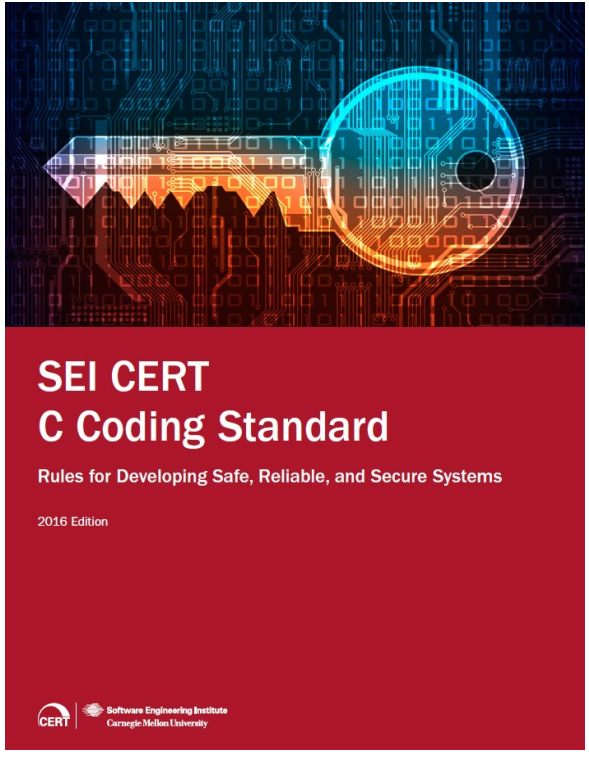
Reload (nodig na elke run)

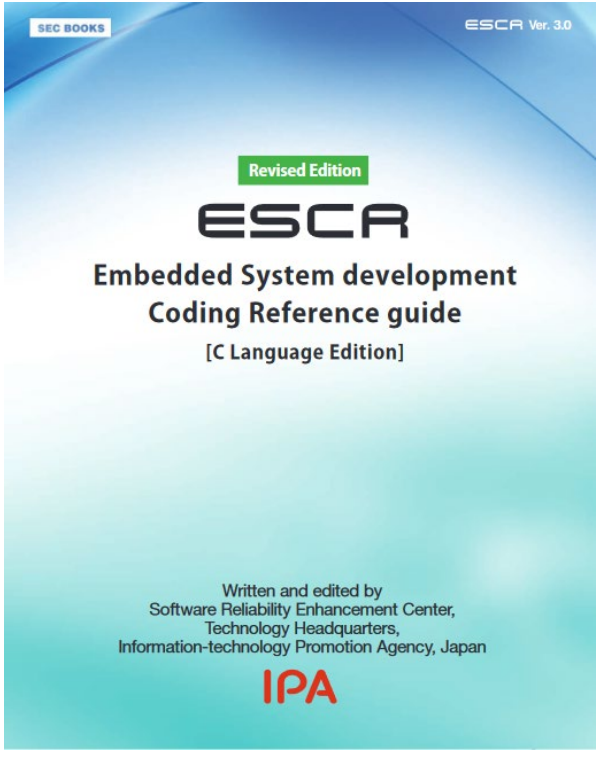
Reset (begin opnieuw met tellen van hits)

Volgende les…

Coding standards en static code analysis







Aan de slag!

Aan de slag met [Opdrachten_Week_3.5.pdf]