|  |  |  |
| --- | --- | --- |
| **ACTIES OP HET SCHERM** | **VOICE-OVER** | **DUUR** |
|  | How do you know your unit tests are complete? Did you forget code paths or functions? This is where code coverage tools come in, they help you to determine exactly that. You have one at your disposal right now, and that is your compiler. |  |
| 1. Compile main.c funcs.c with –fprofile-arcs –ftest-coverage 2. Run a.out 3. Do gcov funcs.c 4. Highlight statistics 5. Show funcs.c.gcov 6. Highlight lines marked with – 7. Hightlight lines with trip count 8. Hightlight lines of fac2 function 9. Highlight line 6 | 1. Compile the code you want a coverage report on with extra command line options, -fprofile-arcs and –ftest-coverage 2. Run the executable. Coverage information is stored on your hard disk in files with extension .gcno and .gcda. 3. To generate a coverage report, run gcov on the source files. 4. This prints some statistics to the screen, which is first indicator for the coverage level. 5. This will also produce an annotated source file. 6. Lines marked with a – are not executable, and don’t concern us here. 7. Lines that have been executed are annotated with the number of times they have been executed. Although this is not of interest in this context, the information may be quite relevant when optimizing an application. 8. Finally, the lines annotated with # are executable, but haven’t been executed. This is the information that tells us whether our tests are complete. The fac2 function has not been executed, so we don’t have any tests for it. 9. Similarly, there is no tests that calls fac with a negative argument, so we fail to test for failure.   Now we can start work on completing the tests. |  |
| **TOTALE DUUR** | | *Maak je screencast niet langer dan ca. 6 minuten.* |