|  |  |  |
| --- | --- | --- |
| **ACTIES OP HET SCHERM** | **VOICE-OVER** | **DUUR** |
|  | A recursive procedure is a procedure that is defined in terms of itself. In other words, the definition of the procedure uses at least one call to the procedure itself. A recursive function probably reminds you of mathematical proofs by induction. You have a base case you know to be true, and the reduce a complex case to a simpler case, ultimately ending up with the base case.  This makes recursive procedures often easier to implement than their non-recursive counterparts, but recursive implementation may have performance issues. |  |
| 1. Show start situation 2. Add recursive call 3. Save file | 1. The Fibonacci function called with argument n is defined in terms of itself, called with values n – 1 and n - 2. We also need a base case, here for n equal to 0 and 1 in which case the result is 1. |  |
| 1. Switch to terminal 2. Compile code | 1. When we compile and run the code, we get an error message telling us that the function can not be called recursively. |  |
| 1. Add recursive | 1. In Fortran, a procedure can only be called recursively if it has been declared as such, so we add the keyword “recursive” to the function signature. |  |
| 1. Switch to terminal 2. Compile code 3. Run application 4. Recompile with -g 5. Start gdb 6. Type r 7. Type bt 8. Type b 13 9. Type r (and y) 10. Type c 11. Type c 5 | 1. When we compile and run the application, everything works out nicely… but we get a segmentation fault. Not so good. 2. Lets recompile our application with debugging symbols included, so we can run it using a debugger to find out what is going on. 3. We now run the application under control of the debugger. 4. Once the debugger is launched, we can run the application using the “run” command. Just typing “r” will do. 5. Again, the application crashes, and gdb shows us this happens in the statement where the result is assigned to the variable “res”. However, it also shows that this happens when “fib” is called with a large negative value. This should not happen, how did we get there? 6. We can have a look at the stack which holds the function calls that have not been completed yet by printing the backtrace, command “bt”. The most recent call is at the top, frame 0, this was called by the function in frame 1 and so on, all the way down. Each function is called with a value one less than the previous one, and we seem to be stuck in an infinite chain of recursive calls. 7. Let’s set a breakpoint in the function at line 13. The execution will halt each time when it hits this line of code, and we’ll have the opportunity to inspect the state of the application. 8. We run the application again and confirm. 9. So now the execution halted in the first call to “fib”, and we expect n to be 5 and that is indeed the case. 10. We can resume execution using “continue” or “c”. Now “fib” was called with 4 as an argument. 11. We can continue for a few more steps, and indeed, we see that “fib” is called with -1 as an argument, and this shouldn’t happen. |  |
| 1. Replace .and. by .or. | 1. Most likely, you already spotted the problem. Our base case is not correct. In practice, a number is rarely equal to both 0 and 1, so if we replace the logical “and” operator by a logical “or”, the problem might be fixed.. |  |
| 1. Switch to terminal 2. Type !gfortran -g fib.f90 3. Type r 4. Type d 1 5. Type c 6. Type q | 1. We can recompile the source code from within the debugger, no need to restart. 2. We run the application using “r”. 3. Of course, the breakpoint is still there. Lets remove it using delete followed by the breakpoint number, 1. 4. When we continue execution, the correct answer 8 is produced. 5. Problem solved, we can quit the debugger. |  |
|  | I’ll confess that this video was intended more as propaganda for using a debugger, rather than to illustrate recursion. However, given that mistakes are easily made when programming in general, having some skills with a debugger never goes amiss. |  |
| **TOTALE DUUR** | | *Maak je screencast niet langer dan ca. 6 minuten.* |