|  |  |  |
| --- | --- | --- |
| **ACTIES OP HET SCHERM** | **VOICE-OVER** | **DUUR** |
|  | Working with arrays is really convenient in Fortran, but what to do if you only know the size of an array at runtime, and not when you’re writing your code? This is where dynamic memory management comes in. Your application can ask the operating system for memory to store its data, and return it when it is no longer required. This adds flexibility to your application, because when you do it right, it will only use as much memory as required, and not more. |  |
| 1. Show program 2. Show declaration 3. Show init\_values | 1. Discuss call to init\_values 2. Discuss declaration of values 3. Discuss init\_values |  |
| 1. Compile 2. Run 3. Run using valgrind | 1. Compile 2. Run 3. Run with valgrind, notice memory leak. |  |
| 1. Switch to video | 1. Discuss memory deallocation by runtime |  |
| 1. Add deallocate | 1. Fix memory leak by single deallocate |  |
| 1. Compile 2. Run with valgrind | 1. Run with valgrind, no more memory leak |  |
| 1. Change allocatable to pointer in program 2. Change allocatable to pointer in init\_values | 1. Runtime can not do everything, pointers won’t be handled. 2. Briefly explain pointers, refer to later material for more in-depth discussion. |  |
| 1. Compile 2. Run with valgrind | 1. Compile and run with valgrind, serious memory leak! |  |
| 1. Fix memory leak by adding deallocate in do-loop | 1. Fix memory leak by deallocate in loop, this causes double deallocation and segfault. |  |
| 1. Compile 2. Run with valgrind | 1. Point out segfault |  |
| 1. Remove deallocate | 1. Fix segfault by removing last deallocate |  |
| 1. Compile 2. Run with valgrind | 1. Show it works now. |  |
|  | As you can see, dynamic memory management is quite useful, but you have to take care not to introduce memory leaks, or segmentation faults due to double deallocations. |  |
| **TOTALE DUUR** | | *Maak je screencast niet langer dan ca. 6 minuten.* |