|  |  |  |
| --- | --- | --- |
| **ACTIES OP HET SCHERM** | **VOICE-OVER** | **DUUR** |
|  | A user defined type is a data structure defined by the programmer. It consists of multiple elements of various types. User defined types form the building blocks of more sophisticated data structures and can make your life as a programmer considerably easier. We will introduce them using a motivating example. We want to compute descriptive statistics measures for an array, specifically, the mean and standard deviation. |  |
| 1. Add function parameters 2. Add computation 3. Add call and print 4. Save file | 1. Since the procedure has to compute multiple values, we have to use a subroutine. The array representing the data would have intent “in”, and the computed values, so the mean, standard deviation and the number of data items would be arguments with intent “out”. 2. The computation is straightforward, and for simplicity, we don’t check we have enough data to compute the measures. Some error handling should be added. 3. In the program unit, we initialize the data, call the subroutine, and use the results. This is a bit messy since the statistical quantities belong together, but are stored in three distinct variables. If we were interested in more measures, for instance the skew or the median, the situation would deteriorate rapidly. |  |
| 1. Switch to terminal 2. Compile code 3. Run application | 1. When we compile and run the code, we get the expected result. |  |
| 1. Add type declaration 2. Add statistics variable declaration 3. Modify subroutine argument 4. Modify use of mean, stddev and n in subroutine 5. Modify use of mean, stddev and n in program unit 6. Save file | 1. An elegant solution is to introduce a user defined type that represents statistical data. The name of this new type is “statistics”. This type has three elements, two of them, “mean” and “stddev” are real numbers, the third one, “n” is an integer. 2. We can declare a variable of this type using “type(statistics)”. 3. Rather than passing three arguments to the function, we will pass just the variable “stats” to hold all results. 4. The subroutine signature needs to be changed accordingly, it now also takes a second argument of type statistics. 5. The elements of such a variable can be accessed using the percentage sign followed by the name of the element. 6. We make the same changes in the program unit when we print the results. |  |
| 1. Switch to terminal 2. Compile code 3. Run application | 1. When we compile and run the application, we get the same result. However, the code is somewhat less messy. |  |
| 1. Add associate statement 2. Replace stats%n by n | 1. A drawback of this approach is that our formula for the standard deviation gets harder to read. However, that is easy to remedy using an associate statement. We associate the name “n” with the “n” element of the “stats” variable. Now we can simply use “n” directly in the formulas for the mean and standard deviation, as that name is associated with the appropriate element of “stats”. |  |
| 1. Switch to terminal 2. Compile code 3. Run application | 1. When we compile and run the application, we get the same result. |  |
| 1. Change subroutine to function 2. Change procedure call in program | 1. We can go one step further, and replace the subroutine by a function that returns a value of type “statistics”. 2. In the program unit, we change the subroutine into a function call, assigning the result to “stats”. |  |
|  | Of course, user defined types don’t only help us to write code that is more elegant and clear, but we will see that they also form the basis for object-oriented programming in Fortran. |  |
| **TOTALE DUUR** | | *Maak je screencast niet langer dan ca. 6 minuten.* |