Scientific Software
Session 1: Fortran 95, Acquaintance

Emil Løvbak, Wouter Baert & Pieter Appeltans October 8, 2020

Introduction

This first session serves as an introduction to Fortran 95. The aim is for you to develop an understanding of the following concepts:

- basic Fortran 95 syntax: variables, programs, functions, subroutines, modules, loops, if-then statements and select-case statements;
- writing to the standard output and reading from the standard input;
- compiling and linking Fortran code and working with makefiles;
- output formatting;
- arrays;
- elemental functions.

Exercises

1. Make a program that reads a number (you may assume it to be an integer) from the standard input and then prints out `Hello, world!` as many times. Add extra checks so that meaningless input such as ridiculously large or negative numbers are handled in a proper way (using if-then statements).
   Extra: Use select-case statements instead of if-then statements.
   Link preparation: Scientific Software Development with Fortran - 2.12, 2.13, 3.1, 5.2, 5.4

2. Put the code from the previous exercise as a subroutine without arguments in a module that is contained in the file `hello_module.f95`. Adjust your previous program so that it calls this subroutine. Save this modified program as `hello_world.f95`.

   Make the executable by first compiling `hello_module.f95` to object code, subsequently compiling `hello_world.f95` and finally composing the executable file (Toledo- Exercise Sessions - Documentation - Compiling using met line commands).

   Answer the following questions:

   - What happens if you try to compose the executable using only one of the two object files?
   - What happens when you try to compile `hello_world.f95` in absence of `hello_module.mod`?

   Link preparation: Scientific Software Development with Fortran - 2.2, 3.2, 3.8

3. Remove the executable and all ".o" and ".mod" files. Download the file named `Makefile` from Toledo (make sure that you save with as name Makefile, so no .txt extension) and compile the program from the previous exercise using

```
$ make hello_world
```

Answer the following questions:

- How does `make` use the mutual dependencies in the `Makefile` to compile as little as possible?
- Modify `hello_world.f95` (for example print your name at the end of the program). What happens if you execute `make hello_world` again?
- Modify `hello_module.f95` (for example print your name instead of `Hello, world!`). What happens if you execute `make hello_world` again?

Finally, try also the other Fortran compilers (ifort and nagfor).
Link preparation: Scientific Software Development with Fortran - 2.2

4. Write a program that contains the subroutine `matrix_stats` that takes as input a real-valued matrix (a two dimensional array) of arbitrary dimension and prints the following properties of this matrix:

   - the number of rows, the number of columns and the number of elements;
   - the smallest and largest value in the matrix (in scientific notation with four decimal places - *Extra:* Would it be useful to print more decimal places?);
   - the ranges of the indices for the rows as well as for the columns;
   - the sum of each row and the sum of each column (in exponential notation with 5 decimal places - *Extra:* Print the result with 9 decimal places. Do you get the result you expected? Why (not)?).

   The output of your program should look as follows:

```
Number of rows:  4, number of columns:  4, number of elements:  16.
Smallest value:  6.2500E-02, largest value:  1.0000E+00.
Range row indices:  1- 4, range column indices:  1- 4.
Sum rows:  0.21250E+01 - 0.21250E+01 - 0.21250E+01 - 0.21250E+01
Sum columns:  0.21250E+01 - 0.21250E+01 - 0.21250E+01 - 0.21250E+01
```

   Test your subroutine in the main program using the two matrices on Toledo. Print also the ranges of the indices in your main program. Why do you get different values compared to inside the subroutine?
   Link preparation: Scientific Software Development with Fortran - 4.8, 4.9, 5.3; Fortran 2003 standard - 13.5.7, 13.5.12

5. (a) In Fortran there are two ways in which a procedure can be defined: either as a `subroutine` or as a `function`. What are the differences between those two in calling the procedure and returning the result? When would you use a `subroutine` and when a `function`?

   (b) Next, write a procedure that takes as input an integer $n > 1$ and has as output a boolean (logical) that indicates whether n is prime. Implement this procedure both as a `subroutine` and as a `function`.

   (c) In this course we will see two special procedure types, namely `elemental` procedures (this week) and `recursive` procedures (next week). Elemental procedures can be used to apply a procedure element wise to a (multi-dimensional) array. For example, the intrinsic functions `sin` and `cos` are elemental. This means that we can apply them to arrays. As such, the following program

```
program
print '(2f10.7)', sin(reshape((/0, 0.1,0.2,0.3 /),(/2,2/)))
end program
```

has as output

```
 0.0000000   0.0998334
 0.1986693   0.2955202
```

Now make your subroutine and function elemental. For the subroutine, add `elemental` before the `subroutine` specifier. For the function, add `elemental` before the declaration of the type of the return value of the function. Test both the elemental subroutine and function on the following two dimensional array

```
  3   5   7   9  11  13  15  17  19  21
 23  25  27  29  31  33  35  37  39  41
```

Extra: How can you avoid copy-pasting this array? Which Fortran syntax and intrinsic function can you use to compose this (two dimensional) array?
Link preparation: Scientific Software Development with Fortran - 3.2, 3.3, 3.4, 3.5