# Exercises for "Fortran code modernization"



## Schedule

| Time | Day 1 | Day 2 |
|---|---|---|
| 9:00 – 9:30 | Introduction.<br>Source form, Data and type declarations, Execution control, Arrays | Dependency Inversion and Submodules |
| 9:30 – 10:00 | | Exercise Session 4 |
| 10:00 – 10:30 | | |
| 10:30 – 11:00 | Exercise Session 1 | Array processing and performance issues |
| 11:00 – 11:30 | | |
| 11:30 – 12:00 | Procedures and their interfaces (1) | Exercise Session 5 |
| 12:00 – 12:45 | Lunch Break | Lunch Break |
| 12:45 – 13:00 | Procedures and their interfaces (2)<br>Global variables | Exercise Session 5 continued |
| 13:00 – 13:30 | | Extensions to I/O<br>Unit Testing, code coverage, documentation |
| 13:30 – 14:00 | Exercise Session 2 | |
| 14:00 – 14:30 | | Exercise Session 6 |
| 14:30 – 15:00 | Dynamic memory<br>Program configuration control | |
| 15:00 – 15:30 | | Parallel Models: Coarray Basics |
| 15:30 – 16:00 | Exercise Session 3 | |
| 16:00 – 16:30 | The Environment Problem<br>The Object-Oriented Paradigm | Exercise Session 7 |
| 16:30 – 17:00 | | |

# Getting started & general remarks

## Preparing the working environment (15 minutes)

1. Please log in to the front end machine with the command
   `ssh vsc40000@login.hpc.ugent.be`
   **Note:** replace vsc40000 with your own VSC user ID, see https://account.vscentrum.be
2. Please follow the instructions for setting up and connecting to a VNC session that are available at http://hpc.ugent.be/userwiki/index.php/User:VNC (requires login).

Once you have the VNC session running, you can do any of the following:

- Open a terminal
- Use the `nano`, `vi` or `emacs` editors to edit program text
- Load a compiler module:

```
module load intel/2018a            # Intel compilers
module load GCC/7.3.0-2.30         # GCC compilers
module load NAGfor/6.2             # NAG compiler (Fortran only)
```

- Load a tools module:

```
module load Inspector/2018_update2                  # Intel inspector
module load Vtune/2018_update2                      # Intel amplifier
module load Valgrind/3.13.0-intel-2018a             # Valgrind
module load likwid/4.3.2-GCCcore-6.4.0              # LIKWID tool
```

- Execute a batch job on a dedicated node (only needed for reliable performance measurements):

  ```
  qsub -I -l nodes=1:ppn=4 -l walltime=2:0:0 -W x=FLAGS:ADVRES:modern_fortran.16
  ```

  once the shell prompt returns, all commands are run on a worker node, **until the session is exited.** It is a good idea to use a separate terminal from that used for compiling programs. The above command will work **on the first day** only. **On the second day**, please replace `modern_fortran.16` in the above submission command line by `modern_fortran.17`.

The exercises folder for this course is

```
export EXERCISES=/apps/gent/tutorials/fortran/2018
```

Please copy the exercise templates, tools and examples to your HOME directory with

```
cp -a $EXERCISES/skeletons  $HOME
cp -a $EXERCISES/tools      $HOME
cp -a $EXERCISES/examples   $HOME
```

The solutions will be made available after each exercise in the directory `$EXERCISES/solutions`.

The next step is to build some tools (the Intel compiler ifort must be accessible for this):

```
cd tools
make
make install
```

Note that the last command above installs the binaries into the $HOME/bin folder. In order to execute these, you may need to augment your PATH as follows:

```
export PATH=$HOME/bin:$PATH
```

## Fortran standard document

A draft of the Fortran 2008 standard is available in PDF format at

http://www.j3-fortran.org/doc/year/10/10-007.pdf

It is recommended to use this document for reference, e.g., looking up standard intrinsic procedure (these are in section 13.7) as well as syntax definitions. As far as the general language rules are concerned, the document may turn out to be rather hard to read and therefore unsuitable as a guide for learning the language quickly.

# Exercise session 1

## Step 1: Building and running a program from TOMS (15 min)

The folder `skeletons/toms_612` contains a `Makefile` and the fixed-source Fortran files `toms_612.f` and `toms_612_externals.f`. Compile and run these programs with the Intel compiler (you can also use the NAG compiler, but it will need the `-dusty` option to raise its tolerance level), and check the results. Then, correct the observed problems:

- the function `D1MACH` contains platform-dependent machine constants. Select the appropriate ones for the architecture you are running on. What do you need to do to fix the compilation error?
  **Hint:** search for an `EQUIVALENCE` statement – this causes overlapping of the memory for the specified objects and is here used in a non-conforming manner.
- where is the output of the program written to? Is this standard-conforming? Add a suitable I/O statement so the output is written to a file `toms_612.res`.

The solution for this exercise will be made available as `solutions/toms_612_1`.

## Step 2: Conversion to free source form (25 min)

Use one of the following programs to convert the fixed version of `toms_612.f` as well as `toms_612_externals.f` to free source form:

(a) convert.exe
(b) to_f90.exe
(c) nagfor (the NAG compiler)

convert.exe and to_f90.exe read the necessary information from standard input, while the NAG compiler requires using the **=polish** or the **=epolish** option, with additional suboptions. Check how the source looks after conversion and put it into a separate folder. Then, after making appropriate changes to the Makefile rules, attempt to build. Finally, check whether the rebuilt TOMS program still produces the correct results.

**Note:** the source code of convert and to_f90 has some additional documentation on how to use these programs. It cannot be claimed that these tools work perfectly i.e., usually some manual tweaking is needed. This is where commercial products like the NAG compiler often have an edge. Documentation for the NAG compiler is available at https://www.nag.co.uk/nag-compiler#tab-357.

The solution for this exercise will be made available as `solutions/toms_612_2`. You can specifically look at the `run_*.sh` scripts that drive the various tools to see what specific options were used.

# Exercise session 2

## Step 3: Replacing obsolescent syntax in TOMS 612 (25 min)

What happens if a compiler switch is added that requests full standard conformance? For Intel Fortran, you can use the following switches:

```
-standard-semantics -stand f08 -warn errors
```

For the NAG compiler, you can use the `-f2003` switch. Fix the flagged obsolescent features.

Some hints:

- Hollerith descriptors can be trivially converted to character edit descriptors. Some are also handed to subroutines, or appear in DATA statements; these need to be changed to appropriately sized strings.
- After an item has been changed, you may want to temporarily revert to compiling without the switches to check whether the results are still correct.
- Avoid slogging through the changes in `toms_612_externals.f90`. Instead, try to get rid of the dependency on it by replacing calls to D1MACH by modern Fortran intrinsics.

Once your code runs, please also execute the compilation on the `toms_612.f90` source file with the
`-gen-interfaces` switch of Intel Fortran and/or the `=interfaces` option of the NAG compiler, and inspect the newly generated source files that contain explicit interfaces for all procedures. What disadvantages do you observe?

The solution for this exercise will be made available as `solutions/toms_612_3`.

## Step 4: Modularizing the code (25 min)

The following should be performed for `toms_612.f90`:

- Put all procedures into a module stored in `mod_toms_612.f90`. Only the main program and the test integrand function should remain in the original source file.
- Enforce strong typing for all program units. At most three IMPLICIT statements should be needed.
- Convert all COMMON blocks to global variables.
- Replace the use of specific intrinsic functions by their generic names.
- Specify the INTENT at least for all dummy arguments of the module procedure TRIEX.
- Which entity of the module needs to be visible from other program units? Introduce encapsulation statements.

As usual, check whether the correct results are still obtained. Finally, check the generated object file for the symbol names, and compare these with the symbol names for the non-modular object file obtained in step 3. The command

```
nm mod_toms_612.o
```

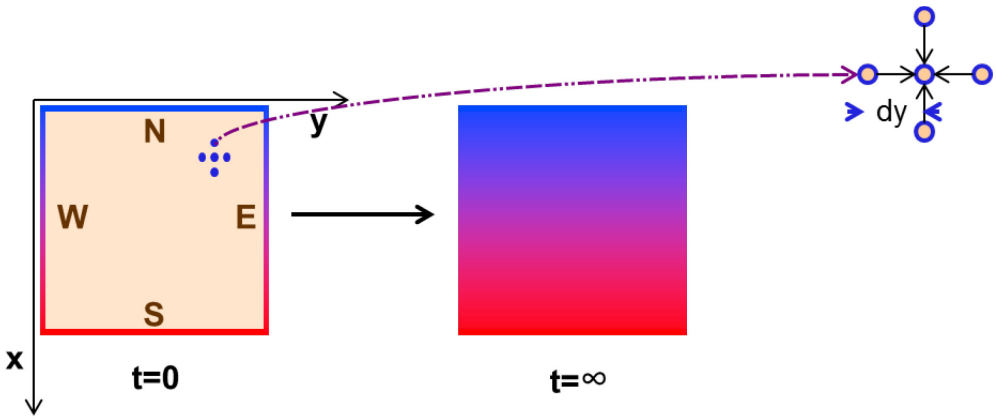
can be used for this purpose.

The solution for this exercise will be made available as `solutions/toms_612_4`.

# Exercises Session 3

## Heat conduction (30 minutes)

Consider a unit square made of a metal, as indicated by the left hand side of the drawing below:




At the edges of the square, it is clamped to heat sources that keep the temperature constant at 0 on the north edge, at some fixed value Φ > 0 at the south edge, and a linear interpolation between 0 and T on the east and west edges, at all times. At time 0, the inside of the square is assumed to have the temperature Φ=0. The temporal change of the temperature field Φ(x, y, t) is described by the partial differential equation

$$\frac{\partial \Phi}{\partial t} = \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2}$$

which over time converges to a stationary solution i.e., one for which the left hand side of the differential equation is equal to zero, as illustrated in the right part of the above figure. The right hand side is also often written as $\nabla^2 \Phi$.

An example program that simulates the above problem can be found in the files `mod_heat_static.f90` and `heat_static.f90` in the `skeletons/heat` folder. The temperature field is simulated by discretizing it along the x and y directions, which requires introducing two `nx` by `ny` arrays that store the temperature values and their updates. The temperature values for time t + δt is obtained from those for time t via incrementation by the temperature differential

$$\delta\Phi = \delta t \cdot \left[\nabla^2 \Phi\left(x_i, y_j\right)\right]_{discretized}$$

1. The main program invokes the iteration procedure repeatedly, until the calculation has converged. Build the executable by running `make`, using a problem size of 200 x 200, and measure the performance. Note that for a small problem size, printout can be generated that permits debugging.
2. Make changes to the source files that allow you to determine the problem size (i.e., the array dimensions `nx` and `ny`) at run time. The following steps will be needed to accomplish this:
   a. Make the necessary changes to the declarations of the fields `phi` and `phinew`. What else is needed to assure the fields are set up correctly?

   b. The x and y directions now may be differently discretized, and the space differentials must also be set up at run time. Use the `ftn_getopt` module discussed in the lecture to enable command line arguments, e.g.
   `./heat_dynamic.exe --nx 50 --ny 100`
   for a problem size 50 x 100.
3. Re-do the performance measurement for the problem size 200 x 200.

The solution for this exercise will be in the files
`examples/solutions/heat/[mod_heat_dynamic.f90, heat_dynamic.f90]`.

# Exercise Session 4

## Returning to TOMS 612

Using the result from Step 4 of the first exercise session (or the provided solution) as a starting point, we note that the argument function `f(x,y)` used by the main driver subroutine still has an implicit interface. Furthermore, the integrand functions use additional parameters. Make the necessary changes for an object-oriented design for this function argument along the lines described in the slides, including introduction of a submodule for the procedure implementation. This will also require making the interface of `f(x,y)` explicit.

# Exercises Session 5

## Some performance tunings for the heat example

Return to the heat conduction example from session 3 and do some performance evaluations.

a. first, add the TARGET attribute to the `phi` and `phinew` arrays and observe the impact on performance for `nx = ny = 200`.
b. Due to copying of `phinew` to `phi`, a factor of two more memory accesses are needed than would be the case if the fields had the POINTER attribute and the pointers are simply switched between iterations. Make a copy of the module within which just that is done and check the performance (possibly for different compilers).
c. Does adding the CONTIGUOUS attribute to the pointer fields improve performance? If not, consider possibilities to eliminate the POINTER attribute for the duration of the calculations.

The solution for this exercise will be in the folder `solutions/heat`, in the files `mod_heat_ptr.f90` and `heat_ptr.f90`.

## Using interoperability features

The folder `skeletons/interop` contains Fortran and C code that makes use of the interoperability features.

1. Write an interoperable Fortran function that corresponds to the C prototype
   `double fun(double x, void *);`
   and call that function (instead of the one provided in the C code) from a C main program. Hint: The Intel compiler requires use of the `-nofor-main` switch in this situation.
2. Write an interface block that corresponds to the C library call
   `float strtof(const char *nptr, char **endptr);`
   and call that function from the Fortran main program, printing out the result. Which other function from the C library will be needed?

The solution for this exercise will be in the folder `solutions/interop`.

# Exercises Session 6

## Using asynchronous I/O

The folder `skeletons/aio` contains source code for a ray tracer. This code performs I/O of the complete picture array at the end of the calculation. The resulting file can be viewed with the `display` command. Convert this program to use asynchronous I/O by putting the data transfer statements inside the outer loop that processes the tiles in the picture. You can reduce the needed amount of storage from `size**2` to `size*nbuf`, where `nbuf` is the number of I/O buffers available for asynchronous I/O. For which picture size do you observe a performance advantage? For good I/O performance it is recommended to use a parallel file system and execute the program there.

**Notes:** For the Intel Compiler, the `-threads` option must be used to activate asynchronous I/O. The current release versions of gfortran compile the programs, but do not actually perform the I/O transfers asynchronously.

The solution will be contained in `solutions/aio`.

## Using pFUnit

The folder `skeletons` contains a subfolder `pfunit_example`, which should be copied as a whole. This is the example used in the slides. Add a subfolder `tests` and produce a suite of unit tests including the build system that fully covers the provided functionality.

# Exercise Session 7

## Extend the Matrix-Vector Multiplication

The folder `skeletons/mv_caf` contains source code for the parallel matrix-vector multiplication. Compile and run the basic version with the gfortran compiler (via the `caf` wrapper) for the case of 1 (nexp=0) and 4 (nexp=1) images. Note that the matrix dimension varies with $2^{nexp}$, so the computational load for 4 images is 4 times as large as for 1 image ("weak scaling").

1. To enable execution of the code for arbitrary image count and arbitrary problem size, introduce allocatable variables instead of the static ones used in the skeleton code. Some conditioning will be required. Remeasure the performance of the code for the cases already treated in the skeleton code.
2. Write a second program that performs the same calculations without using coarrays. Hint: use a suitable collective.

The solution will be contained in `solutions/mv_caf`.