Goals for today

- Create and use Fortran libraries
- Understand the difference between static and dynamic libraries
- Get to know existing Fortran libraries
- Analyze Fortran code using code analysis software, debuggers, and profilers

What is a (Fortran) library?

- A library is a collection of procedures that perform thematically related tasks. The procedures can be distributed over several modules and files.

- There are dynamic and static libraries:
  - Dynamic libraries are loaded at runtime and usually have the extension .so (shared object)
  - Static libraries are built into the executable program and usually have the extension .a (archive)

- To avoid errors, libraries need interfaces to the calling program:
  - .mod files for modules (from Fortran 90)
  - .inc or .h files for other external procedures

Example code

operations.f90

```fortran
MODULE lines
CONTAINS
  REAL FUNCTION add(a,b)
    IMPLICIT NONE
    REAL, INTENT(IN) :: a, b
    add = a+b
  END FUNCTION add
  REAL FUNCTION subtract(a,b)
    IMPLICIT NONE
    REAL, INTENT(IN) :: a, b
    subtract = a-b
  END FUNCTION subtract
END MODULE lines
```

```fortran
MODULE dots
CONTAINS
  REAL FUNCTION multiply(a,b)
    IMPLICIT NONE
    REAL, INTENT(IN) :: a, b
    multiply = a*b
  END FUNCTION multiply
  REAL FUNCTION divide(a,b)
    IMPLICIT NONE
    REAL, INTENT(IN) :: a, b
    divide = a/b
  END FUNCTION divide
END MODULE dots
```

```
$ gfortran -c operations.f90 ohm.f90 main.f90
$ ls
dots.mod      main.o      ohm.o
lines.mod     ohm.f90     operations.f90
main.f90      ohm.mod     operations.o
```

One .mod file per module, one .o file per .f90 file

ohm.f90

```fortran
MODULE ohm
USE dots
CONTAINS
  REAL FUNCTION I(U,R)
    IMPLICIT NONE
    REAL, INTENT(IN) :: U, R
    I = divide(U,R)
  END FUNCTION I
  REAL FUNCTION R(U,I)
    IMPLICIT NONE
    REAL, INTENT(IN) :: U, I
    R = divide(U,I)
  END FUNCTION R
  REAL FUNCTION U(R,I)
    IMPLICIT NONE
    REAL, INTENT(IN) :: R, I
    U = multiply(R,I)
  END FUNCTION U
END MODULE ohm
```

main.f90

```fortran
PROGRAM main
USE ohm, ONLY: U
IMPLICIT NONE
REAL :: R=2.0, I=3.0
PRINT*, U(I,R)
END PROGRAM main
```

Create and link static and dynamic libraries

Archiver
c: create, r: replace

Static library

- Create:

```
$ ar cr libmod_stat.a ohm.o operations.o
```

- Link:

```
$ gfortran -o a_stat.out main.f90 libmod_stat.a
```

Advantage:

- The executable works safely (even if the library changes)

Position Independent Code:
relative instead of absolute addresses

Dynamic library

- Create:

```
$ gfortran -shared -fPIC -o libmod_dyn.so operations.f90 ohm.f90
```

- Link:

```
$ gfortran -o a_dyn.out main.f90 libmod_dyn.so
```

Advantages:

- Can be easily changed (without recompiling the program)
- The executable program is smaller

```
16920 May  22 14:57 a_dyn.out
17232 May  22 14:57 a_stat.out
```

Available Fortran libraries

There are numerous freely available Fortran libraries for various applications:

- Error handling and testing
- Parallelization
- Mathematics and statistics
- Numerics and scientific computing
- Reading and writing files
- Graphics
- Date and time
- ...

Overview

- http://fortranwiki.org/fortran/show/Libraries
- https://github.com/rabbiabram/awesome-fortran

Basic Linear Algebra Subprograms (BLAS) and Linear Algebra Package (LAPACK)

- BLAS provides elementary operations of linear algebra like vector and matrix multiplications

- LAPACK uses BLAS and includes efficient routines for solving systems of linear equations, eigenvalue problems, least squares, singular value decomposition, ...

Example: System of equations

$$
\begin{aligned}
3x_1 + 2x_2 - x_3 &= 1 \\
2x_1 - 2x_2 + 4x_3 &= -2 \\
-x_1 + \frac{1}{2}x_2 - x_3 &= 0
\end{aligned}
$$

```
a = RESHAPE((/ 3.0, 2.0,-1.0,    &
               2.0,-2.0, 4.0,    &
              -1.0, 0.5,-1.0 /), &
            SHAPE(a), order=(/2,1/))
b = (/1.0, -2.0, 0.0/)

CALL SGESV(3,1,a,3,ipiv,b,3,info)
WRITE(*,'(3(A9,F8.5))') 'x1 =',b(1), 'x2 =',b(2), 'x3 =',b(3)
```

```
     x1 = 1.00000     x2 =-2.00000     x3 =-2.00000
```

NetCDF and ecCodes

- **NetCDF** (Network Common Data Form), **GRIB** (GRIdded Binary) and **BUFR** (Binary Universal Form) are binary file formats for the exchange of scientific data, which are widely used in meteorology and climate science.

- They have several advantages over normal binary files:
  - **Self-description**: They include metadata about the stored data, which include information about variable names, dimensions, units, and other attributes.
  - **Portability**: They can be read and written on various platforms and by different programming languages (e.g. Python netCDF4 or xarray).
  - **Compression**: They support compression techniques to reduce file size while preserving data integrity.
  - **Data subsets**: They provide mechanisms to access subsets of data without reading the entire file

- The NetCDF library reads and writes NetCDF files. The ecCodes library reads and writes GRIB and BUFR files.

NetCDF example

From: https://www.unidata.ucar.edu/software/netcdf/examples/programs/

We are writing 2D data on a 6x12 grid →
ID numbers for files, variables, dimensions →
Data array →
Loop indices and error handling →

Fill data array with integers →

Create file →
Define dimensions →

Define array with IDs of dimensions →

Define variable →
Exit definition mode →

Write data →
Close file →

NetCDF "replace"

NetCDF data type

```
PROGRAM simple_xy_wr
  USE netcdf
  IMPLICIT NONE
  INTEGER, PARAMETER :: ndims=2, nx=6, ny=12
  INTEGER :: ncid, varid, dimids(ndims), x_dimid, y_dimid
  INTEGER :: data_out(ny, nx)
  INTEGER :: i, j, ierr

  DO CONCURRENT (i=1:nx, j=1:ny)
    data_out(j, i) = (i - 1) * ny + (j - 1)
  END DO

  ierr = nf90_create('simple_xy.nc', NF90_CLOBBER, ncid)
  ierr = nf90_def_dim(ncid, 'x', nx, x_dimid)
  ierr = nf90_def_dim(ncid, 'y', ny, y_dimid)

  dimids =  (/ y_dimid, x_dimid /)

  ierr = nf90_def_var(ncid, 'data', NF90_INT, dimids, varid)
  ierr = nf90_enddef(ncid)

  ierr = nf90_put_var(ncid, varid, data_out)
  ierr = nf90_close(ncid)
END PROGRAM simple_xy_wr
```

Datetime

- Datetime provides routines for calculating date and time (similar to datetime in Python)

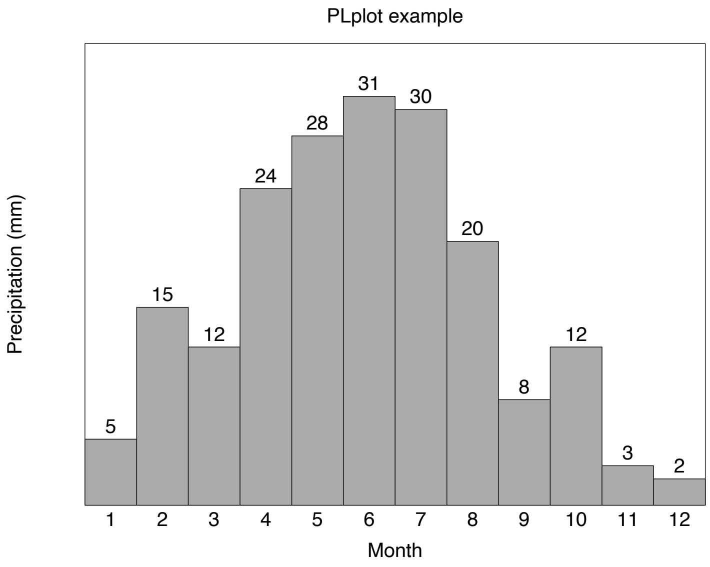
```
PROGRAM today
USE datetime_module, ONLY : timedelta, datetime

IMPLICIT NONE
TYPE(datetime) :: a, b
TYPE(timedelta) :: diff

a = a%now()
b = datetime(a%getYear()-1, 12, 31)
diff = a-b
WRITE(*,'(A,I4,A,I5,A)') 'Today is the', diff%getDays(), &
  '. day of the year', a%getYear(), '.'

END PROGRAM today
```

Data types defined in the module

```
Today is the 144. day of the year 2024.
```

Graphics

- Fortran can also plot, e.g. with these graphics libraries:
  - [PLplot]
  - [NCAR Graphics]
  - [GNUplot]



Example: PLplot

```
CALL plinit()
CALL pladv(0)
CALL plvsta
CALL plwind(1., 13., 0., 35.)
CALL plbox('bc', 1., 0, 'bcnv', 10., 0)
CALL pllab('Month', 'Precipitation (mm)', 'PLplot example')
y0 = (/ 5, 15, 12, 24, 28, 31, 30, 20, 8, 12, 3, 2 /)
DO i = 1, 12
    CALL plcol1(0.0)
    CALL plfbox(REAL(i), y0(i))
    WRITE(string, '(I0)') INT(y0(i))
    CALL plptex(i+0.5, y0(i)+1., 1., 0., 0.5, string)
    WRITE(string, '(I0)') i
    CALL plmtex('b', 1., (i-0.5)/12., 0.5, string)
END DO
CALL plend
```

Load library from another directory

- The directories for the libraries and interfaces can be defined with `-L` and `-I`, respectively. Libraries are specified with `-lname` in this case.

- The compiler then searches
  - in the `-L` directory for `libname.a` or `libname.so` (this only works if the library name starts with `lib`)
  - in the `-I` directory for `.mod`, `.h`, or `.inc` files

- Alternatively, a path can be specified for each library.

- Caution: For dynamic libraries, the program must also know the directory at runtime. There are two ways to do this:
  - Add path to environment variable `LD_LIBRARY_PATH`
  - Include path in executable program with `-rpath` when linking

Example Makefile 1

```
FC = gfortran
FFLAGS = -O2 -Wall
LIBPATH = ./lib
INCPATH = ./include

a_dyn.out: main.o
	$(FC) -o $@ $< -L$(LIBPATH) -lmod_dyn

main.o: main.f90
	$(FC) $(FFLAGS) -c $< -I$(INCPATH)

clean:
	rm -f main.o a_dyn.out
```

Path to the library (libmod_dyn.so)

Path to the interface (ohm.mod)

```
$ ls
include  lib  main.f90  Makefile
$ ls include/
ohm.mod
$ ls lib/
libmod_dyn.so
```

Define new LD_LIBRARY_PATH

```
$ make
gfortran -O2 -Wall -c main.f90 -I./include
gfortran -O2 -Wall -o a_dyn.out main.o -L./lib
-lmod_dyn
$ ./a_dyn.out
./a_dyn.out: error while loading shared libraries:
libmod_dyn.so: cannot open shared object file: No
such file or directory
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./lib
$ ./a_dyn.out
   6.00000000
```

Example Makefile 2 (with NetCDF)

The NetCDF library is installed in the magic environment on the Jupyter Hub.

```
FC = gfortran
FFLAGS = -O2 -Wall
SRC = kinds.f90 findiff.f90 poisson.f90 main.f90
OBJ = $(SRC:.f90=.o)

LIBPATH = -L/headless/envs/magic/lib
INCPATH = -I/headless/envs/magic/include

conv_model: $(OBJ)
	$(FC) -o $@ $(OBJ) $(LIBPATH) -lnetcdff

%.o: %.f90
	$(FC) $(FFLAGS) -c $< $(INCPATH)

findiff.o: kinds.o

poisson.o: kinds.o

main.o: kinds.o findiff.o poisson.o

clean:
	rm -f *.o *.mod conv_model
```

Code analysis

To better understand programs and/or ensure their quality, they need to be analyzed. There are two types of code analysis:

Static code analysis

- Source code is analyzed and checked for errors without running the program
- Can be done by humans or tools (e.g. compilers)

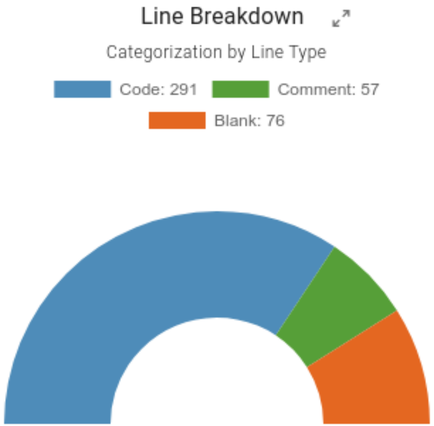
Dynamic code analysis

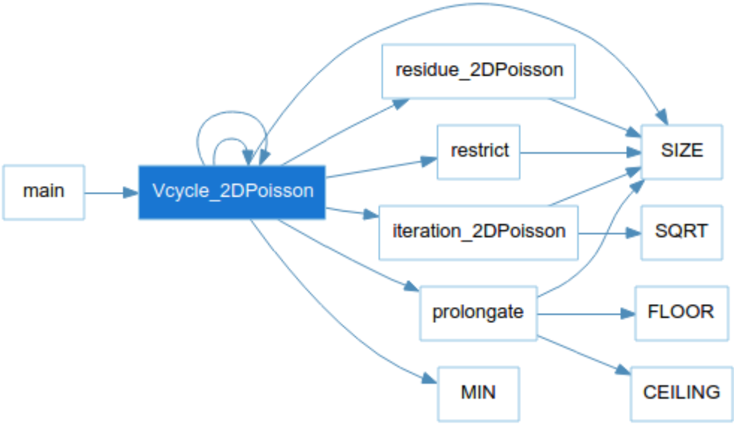
- Analysis takes place during the execution of the program
- Types of dynamic code analysis:
  - <u>Debugging</u>: the program is run step by step, variables are displayed
  - <u>Testing</u>: the program is run with the aim of finding errors
  - <u>Profiling</u>: runtime data of the program are measured, e.g. number of calls of procedures, runtime of single procedures

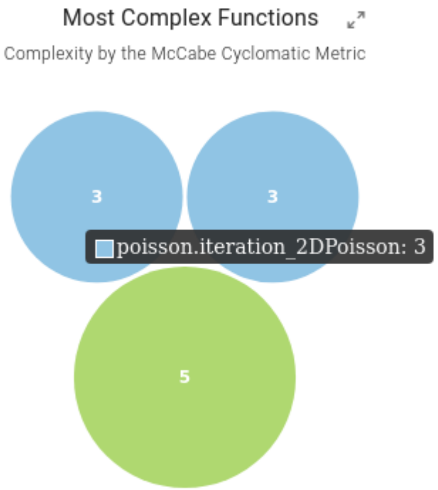
Understand (Scitools)

- Understand is an integrated development environment that enables static code analysis through a set of visualizations, reports, and metrics

- Free licenses are available for students and teachers: https://licensing.scitools.com/student

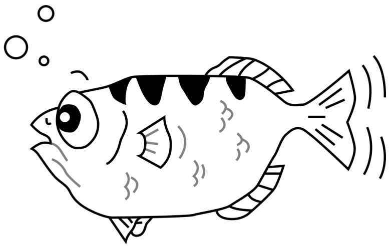






GNU Debugger (GDB)

- GDB enables tracing, display of variables, and intervention in the execution of programs.
- Standard debugger on Linux systems



Compile program with -g

```
$ gfortran -g kinds.f90 poisson.f90 findiff.f90 main.f90
$ gdb ./a.out
```

Important GDB commands

Command	Description
b(reak)	Set breakpoint
c(ontinue)	Continue
d(elete)	Delete breakpoint
fin(ish)	Continue to the end of the function
i(nfo) b(reakpoints)	Show breakpoints
l(ist)	Show source code
p(rint) var	Show variable var
q(uit)	Exit GDB
r(un)	Run program
s(tep)	Execute next line

GNU Profiler (Gprof)

- Gprof is a profiling program that measures runtimes in a program. It shows where a program spends how much time, and which functions / subroutines are called how often. → Helps to find bottlenecks

```
$ gfortran -O2 -pg kinds.f90 poisson.f90 findiff.f90 main.f90
$ ./a.out
$ gprof ./a.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 63.59     14.85    14.85  2307240     6.44     6.44  __poisson_MOD_iteration_2dpoisson
  7.93     16.70     1.85                             MAIN__
  7.33     18.41     1.71    79560    21.51    21.51  __poisson_MOD_prolongate
  7.20     20.09     1.68    59568    28.22    28.22  __findiff_MOD_deriv1_centered
  5.57     21.39     1.30    39712    32.75    32.75  __findiff_MOD_get_vgrad_upwind
  4.50     22.44     1.05    39712    26.46    26.46  __findiff_MOD_get_nabla2
  2.31     22.98     0.54    79560     6.79     6.79  __poisson_MOD_residue_2dpoisson
  1.11     23.24     0.26    19890    13.08   878.83  __poisson_MOD_vcycle_2dpoisson
  0.51     23.36     0.12    79560     1.51     1.51  __poisson_MOD_restrict
```

1. Compile program with -pg
2. First run normally
3. Then run with gprof

Summary

- Libraries are collections of procedures that perform related tasks.

- Static libraries are built into the executable program, dynamic libraries are loaded at runtime.

- LAPACK, NetCDF, ecCodes, Datetime, PLplot are a selection of many freely available Fortran libraries.

- Code analysis software helps in the development of programs, e.g. for static code analysis, debugging, or profiling.