Interactive Debugging
An introduction to using gdb

Prof Matt Probert
http://www-users.york.ac.uk/~mijp1

First steps - getting started

- To enable debugger use must set compile flags:

```
$ gfortran -g -O0 -Wall -fcheck=all my_program.f90
```

- To launch debugger:

```
$ gdb ./a.out
```

- To enter debugging mode:

```
$ (gdb) run
```

- After encountering an error:

```
$ (gdb) backtrace
```

Breakpoints and navigating code

- Can run code and stop at a specified point

```
(gdb) break my_prog.f90:20
(gdb) run/continue
```

- Or just navigate through codes:

```
(gdb) list
(gdb) step 20
(gdb) reverse-step 20
(gdb) next 20
```

(skips over functions)

Printing values

- Print current state of specific variables

  (gdb) **p**rint my_var

- Print current state of ‘var’ that is defined in module ‘mod’

  (gdb) **p**rint mod::var

- Print first 10 elements of a double allocatable array

  (gdb) **p**rint *((double *)my_arr)@10

- Print 2nd element of an integer allocatable array

  (gdb) **p**rint *((integer *)my_arr+1)

- Print current state of **all** variables

  (gdb) **i**nfo locals

  (gdb) **i**nfo args

Watchpoints

- You can put a watchpoint to see when variables change:

```
$ (gdb) watch my_var
```

- You can get rid of watchpoints (or breakpoints)

```
  (gdb) disable my_var
$ (gdb) delete my_var
```

More?

- Many more gdb commands available – see my `gdb_fortran_tutorial.pdf` for more
- You can also embed gdb into your IDE (e.g. VSCode or emacs) to make it less scary to use – see my `gdb_fortran_tutorial.pdf` for more!