gdb

or

what to do when my program segfaults?

gdb

- Gnu debugger, for several languages, incl. C and C++
- It allows you to inspect what a program is doing at points during execution
- Esp. useful to find out the cause of a segmentation fault

- Online manual at

  http://www.gnu.org/software/gdb/documentation/

- To use gdb, you have to compile code with –g option of gcc;
  gcc will then instrument code with the necessary hooks for gdb to interact with the running program

starting gdb

Starts with the command gdb, which provides a prompt

```
> gdb
(gdb)
```

The interactive shell of gdb works much like the normal Linux shell.

Eg you can access history with the arrow keys, use TAB for completion, CTRL-C to interrupt things, etc

To get help (on a particular command), type

```
(gdb) help [command]
```

file and run

With the command **`file`** you load an executable into gdb

With the command **`run`** you run it

```
(gdb) file prog1.x
(gdb) run
```

If the program runs without problems, it runs like it would normally

If the program crashes, you'll get some useful info (eg line number where it crahs, values of parameters, etc)

breakpoints using break

Breakpoints are used to let gdb stop the program during execution

The easiest way it so specify a breakpoint as a line number in a file:

```
(gdb) break file1.c:25
```

Now execution will be halted when it reaches line 25 of **`file1.c`**

You can also set a breakpoint at a function call of a given function

```
(gdb) break some_function
```

Now execution will be halted when **`some_function`** is called

You can set as many breakpoints as you want

Now what? continue, step, and next

After a breakpoint, you can use the commands

- **(gdb) continue**

  to continue execution, until the next breakpoint

- **(gbd) step**

  to step through execution one line at a time

Stepping through the code is very slow; you will also step through any function calls that are done. A faster alternative is

(gbb) next

which steps through execution, but treats function calls as one step

Tip: just pressing Enter will repeat the last command

Now what? inspecting the program

By stepping and continuing you can only find out the program flow
You can also inspect the values of variables

```
(gdb) print my_var
(gdb) print *my_ptr
(gbd) print/x  my_var
```

Here **`print/x`** will print hexadecimal

`(gdb) info args`
prints arguments of the current function call

`(gdb) info locals`
prints all local variables

`(gdb) info variables`
prints all variables

Now what? inspecting the program

(gdb) **`backtrace`**

prints the stack trace that led to the current point of execution

(gdb) **`backtrace n`**

prints the stack trace for the last n function calls only

Getting fed up stepping? conditional breakpoints

For a breakpoint you can also specify a condition

```
(gdb) break file1.c:25  if i>= ARRAY_SIZE

(gdb) break file2.c:100  if ptr == NULL
```

Using conditional breakpoints can avoid a lot of stepping!

watchpoints

Instead of using breakpoints, to interrupt execution at specific program points, you can use watchpoints to interrupt execution whenever a given variable is modified

```
(gdb) watch my_var
```

will halt execution whenever `my_var` is modified.

Warning: if there are multiple variables called `my_var`
then gdb picks the one that is currently in scope

Inspecting the stack

(gdb) **`frame`**

prints some info on the current stack frame

(gdb) **`info frame`**

prints more detailed info

(gdb) **`frame n`**

selects frame n to inspect,

where 0 is the current frame, 1 its caller, 2 its caller's caller, etc

Other useful commands

(gdb) **`info breakpoints`**

lists all breakpoints

(gdb) **`delete some_breakpoint`**

to remove a breakpoint

(gdb) **`finish`**

will continue execution until the current function finishes

(gdb) **`quit`**

exit gdb

For many commands, you can use the first letter as abbreviation

Eg **`q`** for **`quit`**