Debugging

Gdb

To attach gdb to a program currently running:

```
gdb --pid <pid>
```

The program is stopped after loading in gdb, use the command cont to make it go on.

To run a program with the debugger:

```
gdb <program>
run <args>
```

If the program terminates with an error, it is possible to get the backtrace of the **stack** with the command bt.

To put a **breakpoint** in a source file, use the following gdb command:

```
breakpoint <source file>:<line number>
```

The program stops when the breakpoint is reached and it is possible to execute step by step using the commands step and next. It is also possible to print a variable value with print <var>.

On Windows

- gdb on msys requires PYTHONPATH to point to the main $PYTHONHOME/Lib directory:

  ```
  export PYTHONPATH="$PYTHONPATH:$PYTHONHOME/Lib"
  ```

- then gdb anatomist will start, but python paths will be messed-up: it will not be able to load python modules (cannot import numpy, typically)
- the solution is to run anatomist "normally", then attach gdb on it
- for this we have to get the process id of the program to be debugged
  - open procmon (process monitor) **with admin rights (important!)**, by right-clicking on the procmon icon and selecting "execute as administrator"
  - fill in admin login/passwd (.\admin-local for our VM)
  - in procmon, select the menu "Tools/process tree"
  - close the tree "wininit" if needed, and look for the process you need (inside a shell somewhere), note its PID.
- then run gdb --pid <pid>

- Stopping a program that hangs using Ctrl-C does not work: it closes gdb also...
- see: http://www.mingw.org/wiki/Workaround_for_GDB_Ctrl_C_Interrupt
- this program is compiled on our machine in /c/bv_issues/debugbreak/debugbreak.exe
- while the process is running and attached to gdb, using "debugbreak <pid>" will stop it and let gdb control it

Valgrind

This tool enables to track memory problems: invalid use of the memory, memory leaks...

To debug a program with Valgrind:

```
valgrind -v --show-reachable=yes --num-callers=20 --leak-check=full --tool=memcheck '<program>' >&
 /tmp/valgrind.log
```

In this example, the results of the Valgring analysis will be stored in the file */tmp/valgrind.log*.