DPST1092 23T2 — Debugging

https://www.cse.unsw.edu.au/~dp1092/23T2/

Debugging Tools

Please watch this pre-recorded video to accompany these lecture slides

Debugging Lecture video

Debugging Tools

Debugging Tools

Debugging is like detective work ...

Examine the crime scene

Examine the output for the given inputs

Form a hypothesis
(what happened? whodunnit?)

Form a hypothesis
(what might have caused this runtime behaviour?)

Look for clues
(to strengthen hypothesis)

Look at code
(to strengthen hypothesis)

Gather evidence

Observe program behaviour**

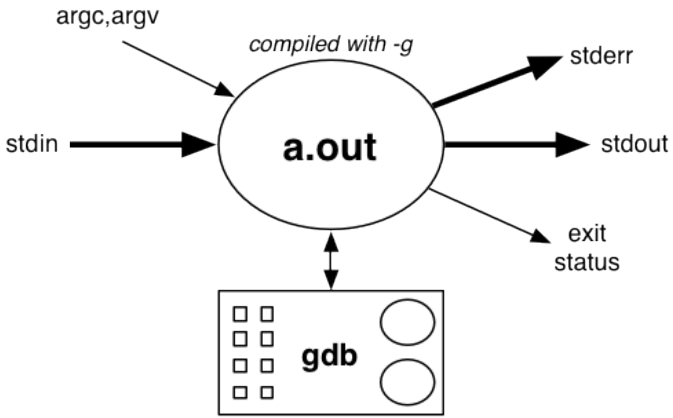
GDB: The Gnu Debugger

gdb provides facilities to

- control execution of program
  (step-by-step execution, breakpoints)
- view intermediate state of program
  (values stored in data structures, control stack)

Using GDB

Program must be compiled using **-g** option.



gdb provides control of execution, monitoring of state

Using GDB

Executing program under gdb control:

```
$ gdb myProg
(gdb) run < dataFile
... crashes, displaying line of code
(gdb) where
... stack trace
(gdb) list
... show code around current location
(gdb) print expr
... display value of expression
(gdb) help
... documentation
(gdb) quit
```

Basic GDB Commands

`quit` -- quits from gdb

`help` `[CMD]` -- on-line help

- Gives information about CMD command.

`run` ARGS -- run the program

- ARGS are whatever you normally use, e.g.

  ```
  $ xyz < data
  ```

  is acheived by

  ```
  (gdb) run < data
  ```

GDB Status Commands

`where` -- stack trace

- find which function the program was executing when it crashed.
- stack may also have references to system error-handling functions.

`up` `[N]` -- move down the stack

- allows you to skip to scope of particular function in stack

`list` `[LINE]` --- show code

- displays five lines either side of current statement.

`print` EXPR -- display expression values

- EXPR may use (current values of) variables
- special expression `a@1` shows all of the array a

GDB Execution Commands

`break` `[FUNC|LINE]` - set break-point

- stop execution and return control to gdb
  on entry to function FUNC or on reaching line LINE

`next` - single step (over functions)

- execute next statement
  if statement is function call, execute whole function

`step` - single step (into functions)

- execute next statement
  if statement is function call, go to first statement in function body

`continue` - resume program execution

- continue to execute statements until a break point is reached or the program terminates

For more details see gdb's on-line help.

Exercise: Monitoring Program Execution

Use GDB to examine the execution of the following:

- iterative factorial function (`fac0.c`)
- recursive factorial function (`fac.c`)
- iterative list traversal (`List.c`)

Do each of the following:

- set a breakpoint
- run the program with command line arguments
- check the stack
- print the values of variable
- step though the next line of code
- continue execution after the breakpoint

valgrind

valgrind is a tool that can

- Find memory leaks (memory you malloced but did not free)
- Find memory errors (bugs where you illegally tried to access memory)

Program must be compiled using **-g** option.

Can be run like:

```
$ valgrind ./a.out
```

Or for more information about memory leaks:

```
$ valgrind --leak-check=full ./a.out
```

Exercise: Finding Memory Leaks

Use valgrind to examine the execution of the following:

- iterative list traversal (`testList.c and List.c`)

Do each of the following:

- Fix any memory errors
- Fix any memory leaks