Lab 6: Debuggers

Comp Sci 1585
Data Structures Lab:
Tools for Computer Scientists

Outline

1 Debuggers

2 GDB
Breakpoints
Inspection
TUI mode
Other frontends
Code::Blocks plugin
Miscellaneous

Debuggers

- Step through code one line at a time
- Inspect variables, including structs and classes
- View disassembly
- Check the call stack
- `$ gdb` Command-line debugger

Outline

1 Debuggers

2 GDB
Breakpoints
Inspection
TUI mode
Other frontends
Code::Blocks plugin
Miscellaneous

Using `$ gdb`

- Note: You will want to compile with `$ g++ -g`
- `$ gdb your-program` launches the debugger
- `run arg1 arg2 ...` runs the program, until finish or crash, with command line arguments
- `start arg1 arg2 ...` starts the program, waiting at the beginning of main, with command line arguments
- `start` or `run` are used with no arguments or input files
- `backtrace` or `bt` shows the call stack

Outline

Setting breakpoints with `$ gdb`

If you don't want to just `start` at the beginning of main, then:

- `break filename.cpp:10` will stop execution whenever line 10 in 'filename.cpp' is reached.
- `delete` removes all breakpoints.
- `advance 14` is a temporary breakpoint to 14.
- `https://sourceware.org/gdb/current/onlinedocs/gdb/Breakpoints.html`

Setting breakpoints with `$ gdb`

At your "paused" position/breakpoint, either:

- `step` runs one more line of code.
- `next` runs until execution is on the next line.
- `finish` runs until the current function returns (if you step to deep).
- `continue` resumes running as normal, until program end or crash

Outline

1 Debuggers

2 GDB
Breakpoints
Inspection
TUI mode
Other frontends
Code::Blocks plugin
Miscellaneous

Looking at variables with `$ gdb`

At your "paused" position/breakpoint, either:

- `info locals` shows local variable values
- `info args` shows arguments to current function
- `p variable` prints the contents of 'variable'.
  `p` also works with expressions of just about any sort.
- `set variable myvar = whatever` allows modifying variables during runtime.
  `set myvar = whatever` may work if your myvar is not named something ambiguous
  `set var myvar = whatever` also works

Memory and reverse engineering

- `x address` examines one word memory at a given address.
- `x/2 address` examines two words of memory. `https://sourceware.org/gdb/current/onlinedocs/gdb/Memory.html`
- `info registers` lists all register values.
- `p $regname` prints the value of a register.

Outline

1 Debuggers

2 GDB
Breakpoints
Inspection
TUI mode
Other frontends
Code::Blocks plugin
Miscellaneous

Text User Interface (TUI) mode

To show the source trace as you next/step

- `$ gdb a.out -tui`
  or
  `(gdb) tui enable`
  or
  `(gdb) layout src)`
  For help and info:
- `(gdb) help layout`
- `(gdb) layout`

Outline

1 Debuggers

2 GDB
Breakpoints
Inspection
TUI mode
Other frontends
Code::Blocks plugin
Miscellaneous

Other frontends

- **KDevelop** gdb terminal, easy watches, requires more significant time to set up project, builds, etc., can run with $< input.txt$, really nice code formatter
- **Code::Blocks** gdb terminal, clunky interface; requires a little time to set up project though still easy, can run with $< input.txt$
- **kdgb** no gdb terminal, can set variables in gui, but not arbitrary values in arrays for example; easy without creating project, can run with $< input.txt$, easy
- **Qt-Creator** decent, offers gdb terminal, can run with $< input.txt$
- **cgdb** command line, most streamlined, easy without creating project; like tui mode, but different shortcuts and color, can run with $< input.txt$
- **insight** pretty good all-round, a bit ugly and clunky, but clean and works, gdb terminal, variabl viewer, etc.
- **DDD** if you can stand the ancient GUI...

Code::Blocks plugin

1. Make a project, accepting defaults for debug target
2. Add your files
3. Use the debug menu to run/start/step
4. Right click on variables to watch them

Outline

1 Debuggers

2 GDB
Breakpoints
Inspection
TUI mode
Other frontends
Code::Blocks plugin
Miscellaneous

Miscellaneous

Conditional Breakpoints:

```
condition breakpoint_number expression
```

Editing variables with `$ gdb` :

- `set var VARIABLE_NAME = value` assigns 'value' to 'VARIABLE_NAME'
- `set {int}0x1234 = 4` writes 4 as an integer to the memory address 0x1234

Disassembling code: `disassemble function` prints the assembly for a function.