Tools of the Trade – Part II: Debugger

Computational Science II (CAAM 520)

Christopher Thiele

Rice University, Spring 2020

Debugging

I wrote some code, the compiler finally stopped complaining, but the code doesn't do what I want it to do.

→ Time for ***debugging***

How not to debug your code

Debugging with `printf()` etc. (see below) is cumbersome, inefficient and possibly misleading!

```
for (int i = 0; i < n; i++) {
  for (int j = i + 1; j < n; j++) {
    if (array[j] < array[i]) {
      tmp = array[i];
      array[i] = array[j];
      array[j] = tmp;
    }
  }
  printf("array[%d] = %d after iteration %d\n",
         i, array[i], i);
}
```

Debugging

Use a proper tool, i.e., a debugger like GDB!

Preparation: For efficient and convenient debugging, compile your code with

- debug symbols (`-g`) and
- possibly less optimization (e.g., `-O0` instead of `-O2`).

```
gcc -g -O0 -o myapp myapp.c
```

Using GDB

To debug your program, run

```
gdb ./myapp
```

or run

```
xterm -e gdb ./myapp
```

to debug in a new terminal.

Using GDB

To print your current position in the code, use the `where` and `frame` commands, e.g.,

```
(gdb) list
```

To view the surrounding source code, use the `list` command.

To find out how you got to the current line of code, use the `backtrace` command.

Note: GDB allows you to use the shortest unambiguous abbreviation for any command, e.g., `l` instead of `list` etc.

Using GDB

To run your program, use the `run` command.

To interrupt execution, use `^C` (Ctrl+C).

To interrupt execution at a specific place, use ***breakpoints***:

```
(gdb) break 123
(gdb) break my_file.c:123
(gdb) break my_function
(gdb) break 123 if my_variable > 42
```

Using GDB

To view breakpoints, enter `info break`.

To delete breakpoints, enter

- `delete` plus the breakpoint ID or
- `clear` plus the location of the breakpoint.

Using GDB

Watchpoints are a special type of breakpoints that interrupt execution whenever the value of a specific variable changes:

```
(gdb) watch my_var
(gdb) run
Hardware watchpoint 1:  my_var

Old value = 1
New value = 2
```

Using GDB

To continue execution, use enter

- `continue` to continue to the next breakpoint,
- `step` to continue to the next line of code, entering called functions,
- `next` to continue to the next line of code, ignoring function calls, or
- `finish` to continue until the current function is left.

Using GDB

To print the value of a variable, use the `print` command.

To examine a block of memory, use the `x` command, i.e.,

```
(gdb) x/nfu address
```

where

- `n` is the number of units to examine,
- `f` is the data format (like `printf()`, e.g., `d` for integers etc.),
- `u` is the unit (`b`, `h`, `w`, `g` for 1, 2, 4, 8 bytes), and
- `address` is the memory address.

Using GDB

You can modify variables using the `set var` command, e.g.,

```
set var i=123
```

To find out the type of a variable, use the `whatis` command.

```
(gdb) whatis i
type = int
(gdb) set var i=123
```

Using GDB

To modify memory directly, use the `set` command, e.g.,

```
(gdb) set {int}0x7fffffffb630=123
(gdb) set {char}0x7fffffffb630='x'
(gdb) set {int}my_int_ptr=123
(gdb) set {char[32]}my_str="hello, world!"
```

Note: In the last two examples, it is easier to use `(gdb)`

```
set var *my_int_ptr=123
(gdb) set var my_str="hello, world!"
```