How to Use GDB

This is a sample debugging session I captured and annotated. You can use the builtin help, Unix manual pages, or "emacs info" pages for more information on gdb. What I typed to the computer will show in **bold**. What the computer responded will show in `courier` font. My annotations appear in *italics* before the line in question.

Compile my program with -g.

```
$ gcc -Wall -Werror -g f2c.c -o f2c
```

Start the symbolic debugger.

```
$ gdb f2c
GDB is free software and you are welcome to distribute copies of it
 under certain conditions; type "show copying" to see the conditions.
There is absolutely no warranty for GDB; type "show warranty" for details.
GDB 4.14 (sparc-sun-solaris2.4),
Copyright 1995 Free Software Foundation, Inc...
```

Get some startup help.

```
(gdb) help
List of classes of commands:

running -- Running the program
stack -- Examining the stack
data -- Examining data
breakpoints -- Making program stop at certain points
files -- Specifying and examining files
status -- Status inquiries
support -- Support facilities
user-defined -- User-defined commands
aliases -- Aliases of other commands
obscure -- Obscure features
internals -- Maintenance commands

Type "help" followed by a class name for a list of commands in that class.
Type "help" followed by command name for full documentation.
Command name abbreviations are allowed if unambiguous.
```

List the program source, first 50 lines.

```
(gdb) list 1,50
1       /*
2        * f2c.c:
3        * Convert temperatures from Fahrenheit to Celsius.
4        * Erez Zadok, W3131, Fall 1995.
5        */
6
7       #include <stdio.h>
8
```

```
9       /*
10       * convert():
11       * Takes a floating number as input (Fahrenheit degrees).
12       * Returns a floating number as output (Celsius degrees).
13       */
14      float convert(float f)
15      {
16          float c;
17
18          c = (f - 32) * 5.0 / 9.0;
19          return c;
20      }
21
22      /*
23       * main():
24       * Iterate over a range of numbers, convert them from Fahrenheit to
25       * Celsius degrees, and print them.
26       */
27      void main()
28      {
29          float min, max, step, f, c;
30
31          min = -40.0;
32          max = 100.0;
33          step = 25.0;
34
35          for (f=min; f<=max; f+=step) {
36              c = convert(f);
37              printf("%6.2ff = %6.2fc\n", f, c);
38          }
39      }
```

Set a breakpoint in the function main().

```
(gdb) break main
Breakpoint 1 at 0x107ec: file f2c.c, line 31.
```

Run the program.

```
(gdb) run
Starting program: /n/shekel/u/zing/ezk/teach/misc/f2c

Breakpoint 1, main () at f2c.c:31
31          min = -40.0;
```

Program stopped at main, right before really starting to run. It shows me the next line that is about to execute before it runs it. Now I want to execute that line and go to the next one using the ‘‘step’’ command.

```
(gdb) step
32          max = 100.0;
(gdb) step
33          step = 25.0;
```

Print the value of variable ‘‘min’’.

```
(gdb) print min
$1 = -40
```

```
(gdb) print max
$2 = 100
```
Print the value of variable ‘‘step’’ not to be confused with the gdb step command.
```
(gdb) print step
$3 = 0
```
Single-step to the next line of source code.
```
(gdb) step
35          for (f=min; f<=max; f+=step) {
(gdb) print f
$4 = -7.84398484e+28
(gdb) step
36              c = convert(f);
(gdb) print f
$5 = -40
(gdb) step
convert (f=-40) at f2c.c:18
18          c = (f - 32) * 5.0 / 9.0;
```
Print the stack trace, showing the calling sequence of functions.
```
(gdb) where
#0  convert (f=-40) at f2c.c:18
#1  0x10848 in main () at f2c.c:36
(gdb) print c
$6 = 0
(gdb) step
19          return c;
(gdb) print c
$7 = -40
(gdb) step
20      }
(gdb) step
main () at f2c.c:37
37              printf("%6.2ff = %6.2fc\n", f, c);
(gdb) where
#0  main () at f2c.c:37
(gdb) step
-40.00f = -40.00c
35          for (f=min; f<=max; f+=step) {
```
Rather than use step, I use ‘‘next’’ which skips over invocations of functions.
```
(gdb) next
36              c = convert(f);
```
Indeed, the next line to run is the one after the convert() function. We did not enter it.
```
(gdb) next
37              printf("%6.2ff = %6.2fc\n", f, c);
```
Just hitting ENTER will re-execute the last gdb command (next in this case).
```
(gdb)
-15.00f = -26.11c
```

```
35          for (f=min; f<=max; f+=step) {
(gdb)
36              c = convert(f);
(gdb)
37              printf("%6.2ff = %6.2fc\n", f, c);
(gdb)
 10.00f = -12.22c
35          for (f=min; f<=max; f+=step) {
```
Set a breakpoint in line 37 of the current source file.
```
(gdb) break 37
Breakpoint 2 at 0x10850: file f2c.c, line 37.
```
Let me see all the breakpoints I've setup.
```
(gdb) info breakpoints
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0x000107ec in main at f2c.c:31
        breakpoint already hit 1 time
2   breakpoint     keep y   0x00010850 in main at f2c.c:37
```
Continue running without stopping or single-stepping, until the next breakpoint is hit.
```
(gdb) continue
Continuing.

Breakpoint 2, main () at f2c.c:37
37              printf("%6.2ff = %6.2fc\n", f, c);
```
Every gdb command has an abbreviation. ''c'' is for ''continue''.
```
(gdb) c
Continuing.
 35.00f =   1.67c

Breakpoint 2, main () at f2c.c:37
37              printf("%6.2ff = %6.2fc\n", f, c);
(gdb)
Continuing.
 60.00f =  15.56c

Breakpoint 2, main () at f2c.c:37
37              printf("%6.2ff = %6.2fc\n", f, c);
(gdb)
Continuing.
 85.00f =  29.44c

Program exited with code 01.
```
Enough of this...
```
(gdb) quit
```