Twelve Basic Gdb Debugging Commands

This handout describes twelve useful debugging commands in the `gdb` debugger. In order to use the debugger, a C program to be debugged must be compiled with the "-g" option. For example, to compile the a program `main.c` to run under `gdb`, use the command:

```
gcc -ansi -pedantic -Wall -Werror -g main.c
```

The 101 handout on gdb basics has further information on get started with `gdb`, including how to run `gdb` inside emacs.

Topic	Command	Description
Starting	gdb *prog*	To invoke the debugger from the UNIX shell, type the name of the debugger followed by the name of the executable object program to debug. Once inside the debugger a prompt appears, at which any of the commands listed below can be entered.
Running	r *[args]*	To run the program under the debugger program, use the `gdb` r(un) command. The run command can have optional command line arguments. These are the same arguments that would be provided at the top-level of UNIX when executing the program outside of the debugger. For example, if the program being debugged is named "prog1", then the command "r prog1 < infile" executes prog1 with redirected input from "infile".
Breakpoints	b *line*	A breakpoint is a line within the program at which the debugger will stop whenever program execution reaches that line. When the breakpoint is encountered, program execution is temporarily suspended, and the debugger command prompt reappears. While the program is suspended, program variables can be examined using the print commands described below. Program execution can be resumed after a breakpoint, using the `continue` command, also described below. A breakpoint can be set at either a numeric line number, or by giving a function name that designates the first executable line of the function. For example, "b 10" sets a breakpoint at line 10, "b func" sets a breakpoint at the first executable line of the function named "func". In a multi-file program, a breakpoint can be qualified by the file name, as in "b main.c:10"; this allows a breakpoint to be set in a file other than the currently selected file.
Back Trace	bt	The back trace command lists function calls that are pending after a program has stopped execution. The command is only valid after execution has been suspended at a breakpoint, or after the program has exited abnormally with some form of runtime error (e.g., segmentation fault). The command is particularly useful after a runtime error, since it indicates the function in which the error occurred, in addition to all pending function calls at the time of the error.

Printing	p expr display expr	The print command is used to examine the value a program variable, or any expression that includes program variables. The command is only valid after execution has been halted at a breakpoint, or after a runtime error. In general, the "expr" argument can be any legal C expression. Gdb can handle just about any legal C expression, including expressions involving function calls. The display command does the same kind of printing as as the print command, but display runs automatically whenever a breakpoint is reached. This is a shortcut for re-typing the print command. For example, you can set a breakpoint in the middle of a program loop, and use the display command to print the value of the loop index variable. Every time the breakpoint is reached, the variable will automatically be printed out. To turn off auto-display, type the gdb command "undisplay".
Continuing	c	Use this command to continue execution after a breakpoint has been encountered.
Single Step	step,next	After a breakpoint, a program can be single stepped through source lines one at a time. Two commands are used for single stepping: "step" and "next". "step" executes all lines, including into functions that are called. "next" skips over function calls, without going into each line within a called function. Either way, you can run the program line-by-line, to get a detailed trace of its execution.
Trace Search	up,down	After a breakpoint, the program context can be moved up and down in the list of pending calls. This is useful for examining variables in different levels of calls in the trace list.
File Select	list *file*:1,1	In a program composed of more than one source file, the debugger's attention must be focussed on one of the source files for the purposes of setting breakpoints and examining lines within the file. In gdb, the "list" command is used to list the first line of the program, which also sets the listed file as the current working file.
Directory Select	dir *dir*	In a large program where source files are in more than one directory, the debugger's attention must be focussed on one of the directories. In gdb, the command is "dir".
Memory Dump	x/fmt addr	It is sometimes necessary to examine the contents of program memory using a machine address rather than a symbolic expression. This is an advanced command that most 101 students most likely won't use, but it's described here in case you want to check it out. Dumping memory in gdb is done with the "x" command The "fmt" argument is a one-character formatting code that specifies how the value at the given address is printed. The codes are: o(octal), x(hex), d(decimal), u(unsigned decimal), f(float), a(address), i(instruction), c(char) and s(string). In gdb, multiple locations can be dumped by using a count prefix in the format specification. E.g., "x/10x M" dumps 10 hex words starting at memory location M.
Help	help	The help command lists help on all topics, and it can be specialized to a particular topic. The UNIX the man page for gdb has additional information.
Quitting	q	Exit gdb. (OK, so technically, this a 13th command.)