LECTURE 7: INTRODUCTION TO DEBUGGING WITH GDB

TWO WAYS TO DEBUG

Using printf statements
Insert printf statements at various points to provide information as your program executes.
Easy to do.
Doesn't require a special executable file.
Using gdb (GNU DeBugger)
Allows you to set breakpoints and execute the program in steps so that you can examine the status before resuming execution.
Lets you inspect the entire program state, not just the values you thought to print out in advance.
Provides a more comprehensive view of values throughout the execution of the program.

gdb HELP

For a listing of gdb help topics or information about a specific gdb command:

(gdb) h	help: lists help topics
(gdb) h topic	help on named topic

COMPILING FOR GDB

To use gdb, you must compile with the -g option:

```
$ gcc -g fileName.c -o fileName
```

This creates an executable file that contains debugging info, e.g., enhanced symbol table.

RUNNING YOUR PROGRAM WITH GDB

Two ways to start gdb with your file and run it:

1.
```
$ gdb fileName
[copyright/help info]
Reading symbols from fileName...done.
(gdb)
(gdb) run
```

2.
```
$ gdb
[copyright/help info]
(gdb)
(gdb) file fileName
Reading symbols from fileName...done.
(gdb)
(gdb) run
```

If there are no problems with your program, it should run normally here.

If there are problems, gdb will usually provide some useful information about why.

note:	gdb has an interactive shell, much like the one you use to interact with Linux.
	up arrow will bring up history
	tab will autocomplete

You can abbreviate many gdb commands.	
(gdb) file vt [...] (gdb) r < vt.in	Runs vt with vt.in as stdin.

SETTING BREAKPOINTS

breakpoint	A way to tell gdb to pause execution of your program at certain points.
	Allows you to step through your code in increments and find where things went wrong.

Two basic ways to set a breakpoint:

1. Set a breakpoint at a line number.
   Execution will pause before executing the specified line:
   `(gdb) break 54`

2. Set a breakpoint on a function:
   Suppose we have the function:
   `int functionName(int a, int b);`
   Execution will pause before executing this function:
   `(gdb) break functionName`
   You can start by setting a breakpoint at main():
   `(gdb) b main`

You can see information about the breakpoints you've set with the info command:

(gdb) info breakpoints	(gdb) i b

CONDITIONAL BREAKPOINTS

The same as normal breakpoints, but you specify some condition that must be met for the breakpoint to trigger.

Example:
`(gdb) break 54 if i >= ARRAYSIZE`

The program will pause before line 54 if the variable i is greater than or equal to the constant ARRAYSIZE.

RESUMING AFTER A BREAK

We've hit a break point and execution paused.
How do we resume execution?

continue	Resume normal execution.	(gdb) continue (gdb) c
step	Single-step (execute just the next line of code)	(gdb) step (gdb) s
next	Similar to step, but will treat a subroutine as one instruction rather than stepping through it.	(gdb) next (gdb) n

REMOVING BREAKPOINTS

(gdb) clear 54 (gdb) clear main (gdb) cl 54	The clear command lets you remove a breakpoint based on where it is in the program (e.g., line number, func. name)
(gdb) delete 1 (gdb) d 1	The delete command lets you remove a breakpoint by specifying its breakpoint number.

SETTING WATCHPOINTS

You can also interrupt execution of the program based on a variable rather than a specific line or function.

watch	
The program will pause whenever the watched variable's value is modified.	(gdb) watch myVar
Note:	Be careful if you are watching a variable whose name is used more than once in your program.

EXAMINING VALUES

We can interrupt and resume execution. Now what?

print	Print the value of a specified variable.	(gdb) print myVar (gdb) p myVar
print/x	Print the value in hexadecimal	(gdb) print/x myVar (gdb) p/x myVar

Variations on print and some other command examples:

(gdb) p i = 2	set the variable i to 2 and print it
(gdb) p 3 * i	print value of expression 3 * i
(gdb) set variable i = 5 (gdg) set var i = 5	set the variable i to 5 without printing
(gdb) info local (gdb) i lo	info: provides values of all local variables

We can also check line numbers with the list command.	
(gdb) l 10	prints the 10 lines around line 10 (lines 5 through 14)
(gdb) l (after listing)	prints the next 10 lines

BACKTRACE

Produces a stack trace of the function calls that lead to a segmentation fault.

THE STACK

When a computer program is executed, it becomes a process.

A process is provided a piece of memory called its virtual memory space.

The virtual memory of a process contains several sections, and we're interested in the stack.

As a program executes, each call to a function sets aside an area of memory called a stack frame that sets aside space for the information the function needs to execute.

This stack grows as functions are called, and as each function returns, it disappears from the stack and indicates where to go next.