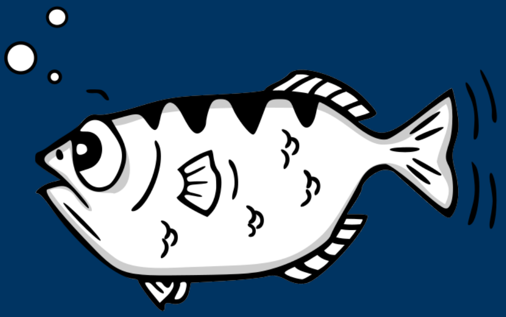
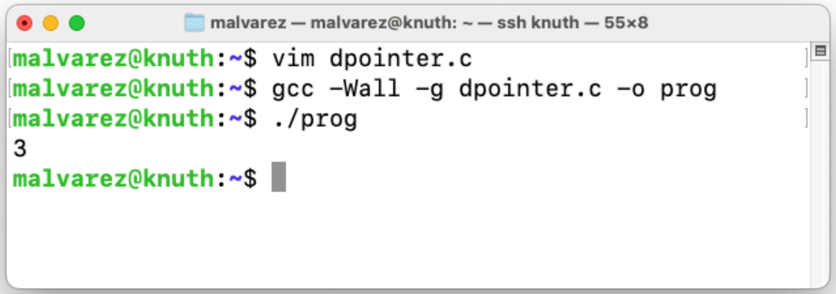
CSC 411

Computer Organization (Spring 2024)
Lecture 6: Debugging (gdb, lldb)

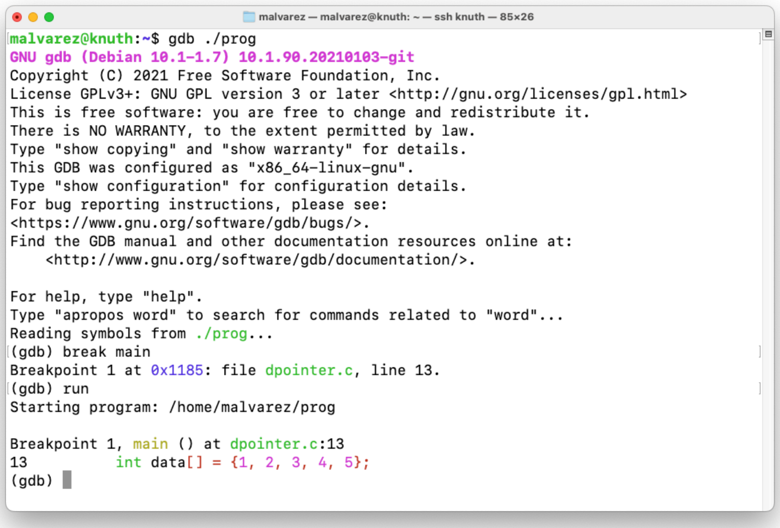
Prof. Marco Alvarez, University of Rhode Island

GDB

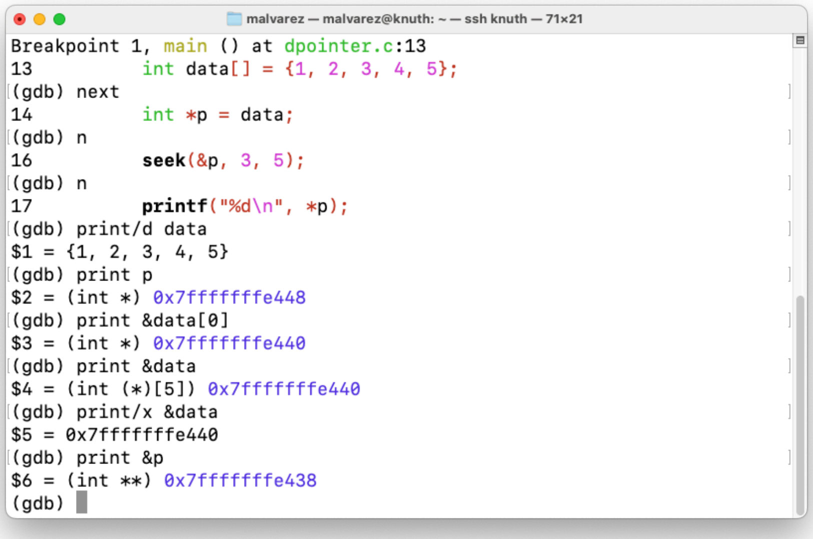




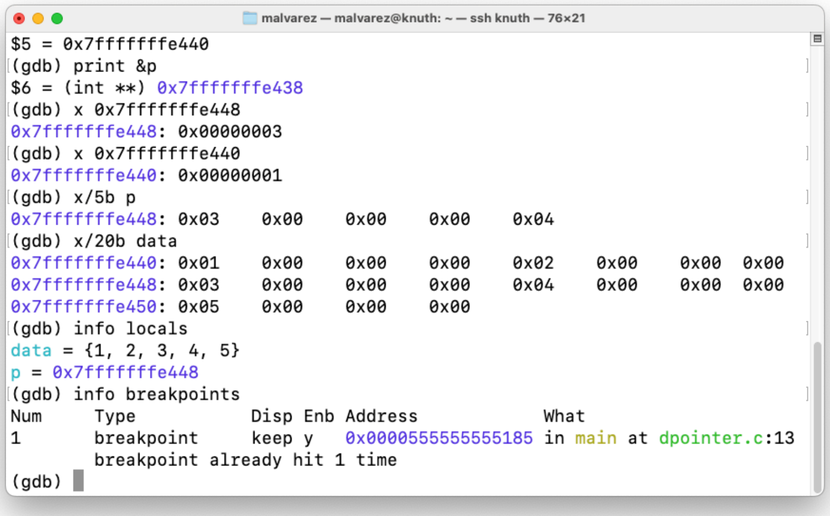
```
malvarez@knuth:~$ vim dpointer.c
malvarez@knuth:~$ gcc -Wall -g dpointer.c -o prog
malvarez@knuth:~$ ./prog
3
malvarez@knuth:~$ 
```



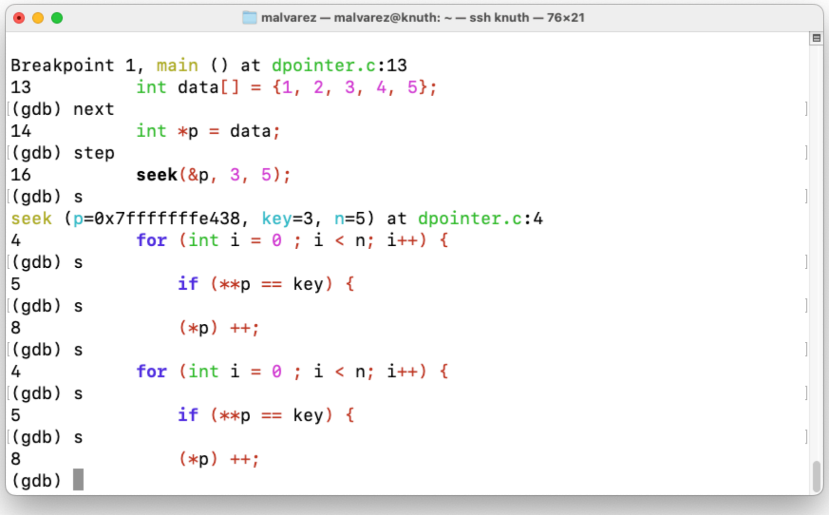
```
malvarez@knuth:~$ gdb ./prog
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./prog...
(gdb) break main
Breakpoint 1 at 0x1185: file dpointer.c, line 13.
(gdb) run
Starting program: /home/malvarez/prog

Breakpoint 1, main () at dpointer.c:13
13          int data[] = {1, 2, 3, 4, 5};
(gdb) 
```



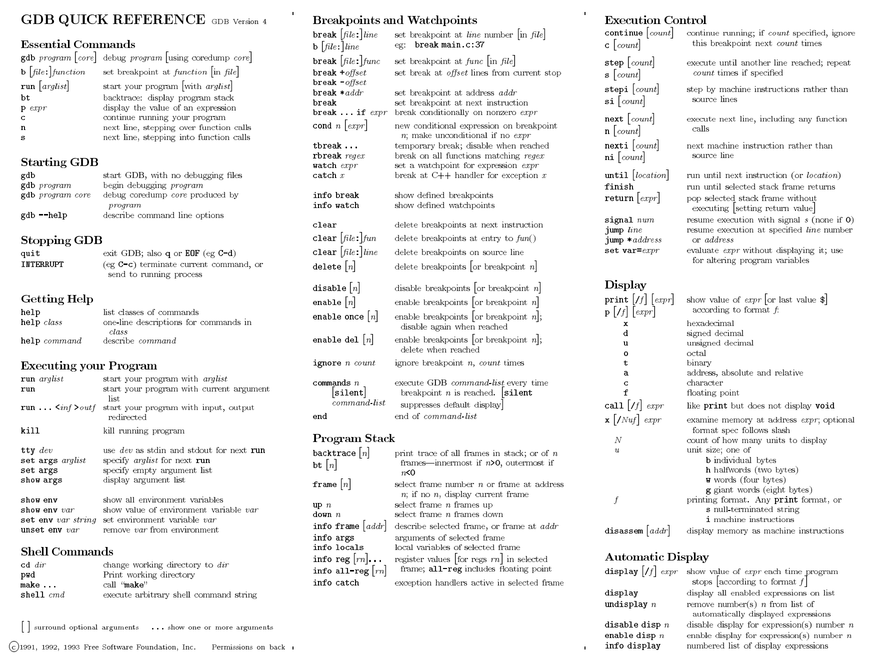
```
Breakpoint 1, main () at dpointer.c:13
13          int data[] = {1, 2, 3, 4, 5};
(gdb) next
14          int *p = data;
(gdb) n
16          seek(&p, 3, 5);
(gdb) n
17          printf("%d\n", *p);
(gdb) print/d data
$1 = {1, 2, 3, 4, 5}
(gdb) print p
$2 = (int *) 0x7fffffffe448
(gdb) print &data[0]
$3 = (int *) 0x7fffffffe440
(gdb) print &data
$4 = (int (*)[5]) 0x7fffffffe440
(gdb) print/x &data
$5 = 0x7fffffffe440
(gdb) print &p
$6 = (int **) 0x7fffffffe438
(gdb) 
```



```
$5 = 0x7fffffffe440
(gdb) print &p
$6 = (int **) 0x7fffffffe438
(gdb) x 0x7fffffffe448
0x7fffffffe448: 0x00000003
(gdb) x 0x7fffffffe440
0x7fffffffe440: 0x00000001
(gdb) x/5b p
0x7fffffffe448: 0x03    0x00    0x00    0x00    0x04
(gdb) x/20b data
0x7fffffffe440: 0x01    0x00    0x00    0x00    0x02    0x00    0x00  0x00
0x7fffffffe448: 0x03    0x00    0x00    0x00    0x04    0x00    0x00  0x00
0x7fffffffe450: 0x05    0x00    0x00    0x00
(gdb) info locals
data = {1, 2, 3, 4, 5}
p = 0x7fffffffe448
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555555185 in main at dpointer.c:13
        breakpoint already hit 1 time
(gdb) 
```



```
Breakpoint 1, main () at dpointer.c:13
13          int data[] = {1, 2, 3, 4, 5};
(gdb) next
14          int *p = data;
(gdb) step
16          seek(&p, 3, 5);
(gdb) s
seek (p=0x7fffffffe438, key=3, n=5) at dpointer.c:4
4           for (int i = 0 ; i < n; i++) {
(gdb) s
5               if (**p == key) {
(gdb) s
8               (*p) ++;
(gdb) s
4           for (int i = 0 ; i < n; i++) {
(gdb) s
5               if (**p == key) {
(gdb) s
8               (*p) ++;
(gdb) 
```



GDB QUICK REFERENCE GDB Version 4

Essential Commands

gdb *program* [*core*]	debug *program* [using coredump *core*]
b [*file*:]*function*	set breakpoint at *function* [in *file*]
run [*arglist*]	start your program [with *arglist*]
bt	backtrace: display program stack
p *expr*	display the value of an expression
c	continue running your program
n	next line, stepping over function calls
s	next line, stepping into function calls

Starting GDB

gdb	start GDB, with no debugging files
gdb *program*	begin debugging *program*
gdb *program core*	debug coredump *core* produced by *program*
gdb --help	describe command line options

Stopping GDB

quit	exit GDB; also **q** or **EOF** (eg **C-d**)
INTERRUPT	(eg **C-c**) terminate current command, or send to running process

Getting Help

help	list classes of commands
help *class*	one-line descriptions for commands in *class*
help *command*	describe *command*

Executing your Program

run *arglist*	start your program with *arglist*
run	start your program with current argument list
run ... <*inf* **>***outf*	start your program with input, output redirected
kill	kill running program
tty *dev*	use *dev* as stdin and stdout for next **run**
set args *arglist*	specify *arglist* for next **run**
set args	specify empty argument list
show args	display argument list
show env	show all environment variables
show env *var*	show value of environment variable *var*
set env *var string*	set environment variable *var*
unset env *var*	remove *var* from environment

Shell Commands

cd *dir*	change working directory to *dir*
pwd	Print working directory
make ...	call "**make**"
shell *cmd*	execute arbitrary shell command string

[ ] surround optional arguments ... show one or more arguments

Breakpoints and Watchpoints

break [*file*:]*line*	set breakpoint at *line* number [in *file*]
b [*file*:]*line*	eg: **break main.c:37**
break [*file*:]*func*	set breakpoint at *func* [in *file*]
break +*offset*	set break at *offset* lines from current stop
break -*offset*	
**break ****addr*	set breakpoint at address *addr*
break	set breakpoint at next instruction
break ... if *expr*	break conditionally on nonzero *expr*
cond *n* [*expr*]	new conditional expression on breakpoint *n*; make unconditional if no *expr*
tbreak ...	temporary break; disable when reached
rbreak *regex*	break on all functions matching *regex*
watch *expr*	set a watchpoint for expression *expr*
catch *x*	break at C++ handler for exception *x*
info break	show defined breakpoints
info watch	show defined watchpoints
clear	delete breakpoints at next instruction
clear [*file*:]*fun*	delete breakpoints at entry to *fun*()
clear [*file*:]*line*	delete breakpoints on source line
delete [*n*]	delete breakpoints [or breakpoint *n*]
disable [*n*]	disable breakpoints [or breakpoint *n*]
enable [*n*]	enable breakpoints [or breakpoint *n*]
enable once [*n*]	enable breakpoints [or breakpoint *n*]; disable again when reached
enable del [*n*]	enable breakpoints [or breakpoint *n*]; delete when reached
ignore *n count*	ignore breakpoint *n*, *count* times
commands *n* [**silent**] *command-list*	execute GDB *command-list* every time breakpoint *n* is reached. [**silent** suppresses default display]
end	end of *command-list*

Program Stack

backtrace [*n*]	print trace of all frames in stack; or of *n*
bt [*n*]	frames—innermost if *n*>0, outermost if *n*<0
frame [*n*]	select frame number *n* or frame at address *n*; if no *n*, display current frame
up *n*	select frame *n* frames up
down *n*	select frame *n* frames down
info frame [*addr*]	describe selected frame, or frame at *addr*
info args	arguments of selected frame
info locals	local variables of selected frame
info reg [*rn*]**...**	register values [for regs *rn*] in selected
info all-reg [*rn*]	frame; **all-reg** includes floating point
info catch	exception handlers active in selected frame

Execution Control

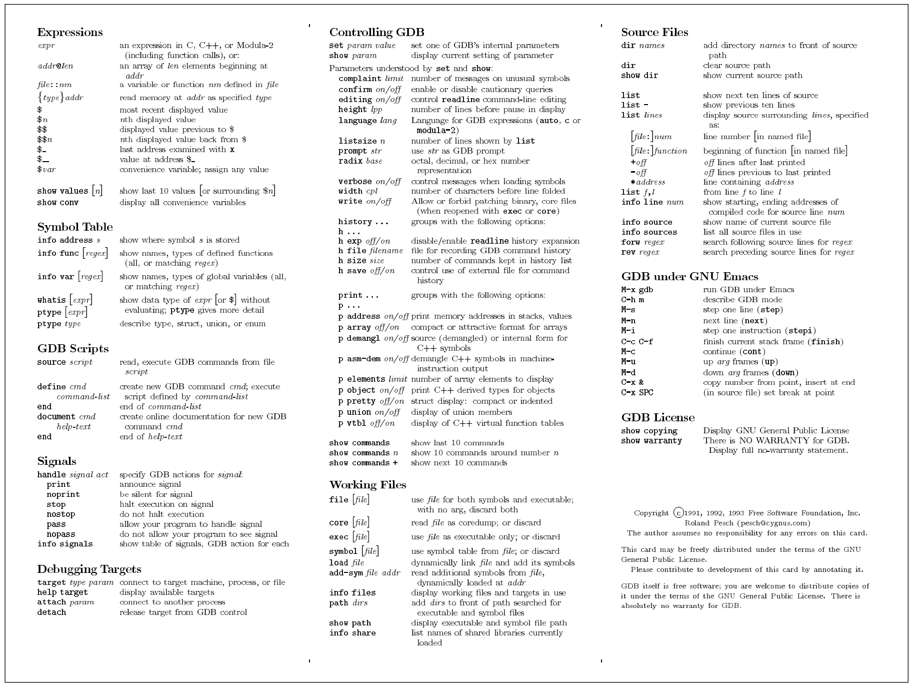
continue [*count*]	continue running; if *count* specified, ignore
c [*count*]	this breakpoint next *count* times
step [*count*]	execute until another line reached; repeat
s [*count*]	*count* times if specified
stepi [*count*]	step by machine instructions rather than
si [*count*]	source lines
next [*count*]	execute next line, including any function
n [*count*]	calls
nexti [*count*]	next machine instruction rather than
ni [*count*]	source line
until [*location*]	run until next instruction (or *location*)
finish	run until selected stack frame returns
return [*expr*]	pop selected stack frame without executing [setting return value]
signal *num*	resume execution with signal *s* (none if 0)
jump *line*	resume execution at specified *line* number
**jump ****address*	or *address*
set var=*expr*	evaluate *expr* without displaying it; use for altering program variables

Display

print [/*f*] [*expr*]	show value of *expr* [or last value **$**]
p [/*f*] [*expr*]	according to format *f*:
x	hexadecimal
d	signed decimal
u	unsigned decimal
o	octal
t	binary
a	address, absolute and relative
c	character
f	floating point
call [/*f*] *expr*	like **print** but does not display **void**
x [/*Nuf*] *expr*	examine memory at address *expr*; optional format spec follows slash
N	count of how many units to display
u	unit size; one of
	b individual bytes
	h halfwords (two bytes)
	w words (four bytes)
	g giant words (eight bytes)
f	printing format. Any **print** format, or
	s null-terminated string
	i machine instructions
disassem [*addr*]	display memory as machine instructions

Automatic Display

display [/*f*] *expr*	show value of *expr* each time program stops [according to format *f*]
display	display all enabled expressions on list
undisplay *n*	remove number(s) *n* from list of automatically displayed expressions
disable disp *n*	disable display for expression(s) number *n*
enable disp *n*	enable display for expression(s) number *n*
info display	numbered list of display expressions



Expressions

expr	an expression in C, C++, or Modula-2 (including function calls), or:
addr@*len*	an array of *len* elements beginning at *addr*
file::*nm*	a variable or function *nm* defined in *file*
{*type*}*addr*	read memory at *addr* as specified *type*
$	most recent displayed value
$*n*	*n*th displayed value
$$	displayed value previous to $
$$*n*	*n*th displayed value back from $
$_	last address examined with **x**
$__	value at address $_
$*var*	convenience variable; assign any value
show values [*n*]	show last 10 values [or surrounding $*n*]
show conv	display all convenience variables

Symbol Table

info address *s*	show where symbol *s* is stored
info func [*regex*]	show names, types of defined functions (all, or matching *regex*)
info var [*regex*]	show names, types of global variables (all, or matching *regex*)
whatis [*expr*] **ptype** [*expr*]	show data type of *expr* [or $] without evaluating; **ptype** gives more detail
ptype *type*	describe type, struct, union, or enum

GDB Scripts

source *script*	read, execute GDB commands from file *script*
define *cmd* *command-list*	create new GDB command *cmd*; execute script defined by *command-list*
end	end of *command-list*
document *cmd* *help-text*	create online documentation for new GDB command *cmd*
end	end of *help-text*

Signals

handle *signal act*	specify GDB actions for *signal*:
print	announce signal
noprint	be silent for signal
stop	halt execution on signal
nostop	do not halt execution
pass	allow your program to handle signal
nopass	do not allow your program to see signal
info signals	show table of signals, GDB action for each

Debugging Targets

target *type param*	connect to target machine, process, or file
help target	display available targets
attach *param*	connect to another process
detach	release target from GDB control

Controlling GDB

set *param value*	set one of GDB's internal parameters
show *param*	display current setting of parameter

Parameters understood by **set** and **show**:

complaint *limit*	number of messages on unusual symbols
confirm *on/off*	enable or disable cautionary queries
editing *on/off*	control **readline** command-line editing
height *lpp*	number of lines before pause in display
language *lang*	Language for GDB expressions (**auto**, **c** or **modula-2**)
listsize *n*	number of lines shown by **list**
prompt *str*	use *str* as GDB prompt
radix *base*	octal, decimal, or hex number representation
verbose *on/off*	control messages when loading symbols
width *cpl*	number of characters before line folded
write *on/off*	Allow or forbid patching binary, core files (when reopened with **exec** or **core**)
history ... **h ...**	groups with the following options:
h exp *off/on*	disable/enable **readline** history expansion
h file *filename*	file for recording GDB command history
h size *size*	number of commands kept in history list
h save *off/on*	control use of external file for command history
print ... **p ...**	groups with the following options:
p address *on/off*	print memory addresses in stacks, values
p array *off/on*	compact or attractive format for arrays
p demangl *on/off*	source (demangled) or internal form for C++ symbols
p asm-dem *on/off*	demangle C++ symbols in machine-instruction output
p elements *limit*	number of array elements to display
p object *on/off*	print C++ derived types for objects
p pretty *off/on*	struct display: compact or indented
p union *on/off*	display of union members
p vtbl *off/on*	display of C++ virtual function tables
show commands	show last 10 commands
show commands *n*	show 10 commands around number *n*
show commands +	show next 10 commands

Working Files

file [*file*]	use *file* for both symbols and executable; with no arg, discard both
core [*file*]	read *file* as coredump; or discard
exec [*file*]	use *file* as executable only; or discard
symbol [*file*]	use symbol table from *file*; or discard
load *file*	dynamically link *file* and add its symbols
add-sym *file addr*	read additional symbols from *file*, dynamically loaded at *addr*
info files	display working files and targets in use
path *dirs*	add *dirs* to front of path searched for executable and symbol files
show path	display executable and symbol file path
info share	list names of shared libraries currently loaded

Source Files

dir *names*	add directory *names* to front of source path
dir	clear source path
show dir	show current source path
list	show next ten lines of source
list -	show previous ten lines
list *lines*	display source surrounding *lines*, specified as:
[*file*:]*num*	line number [in named file]
[*file*:]*function*	beginning of function [in named file]
+*off*	*off* lines after last printed
-*off*	*off* lines previous to last printed
*****address*	line containing *address*
list *f*,*l*	from line *f* to line *l*
info line *num*	show starting, ending addresses of compiled code for source line *num*
info source	show name of current source file
info sources	list all source files in use
forw *regex*	search following source lines for *regex*
rev *regex*	search preceding source lines for *regex*

GDB under GNU Emacs

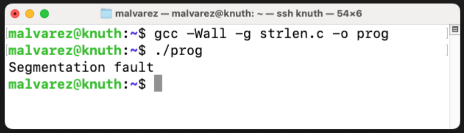
M-x gdb	run GDB under Emacs
C-h m	describe GDB mode
M-s	step one line (**step**)
M-n	next line (**next**)
M-i	step one instruction (**stepi**)
C-c C-f	finish current stack frame (**finish**)
M-c	continue (**cont**)
M-u	up *arg* frames (**up**)
M-d	down *arg* frames (**down**)
C-x &	copy number from point, insert at end
C-x SPC	(in source file) set break at point

GDB License

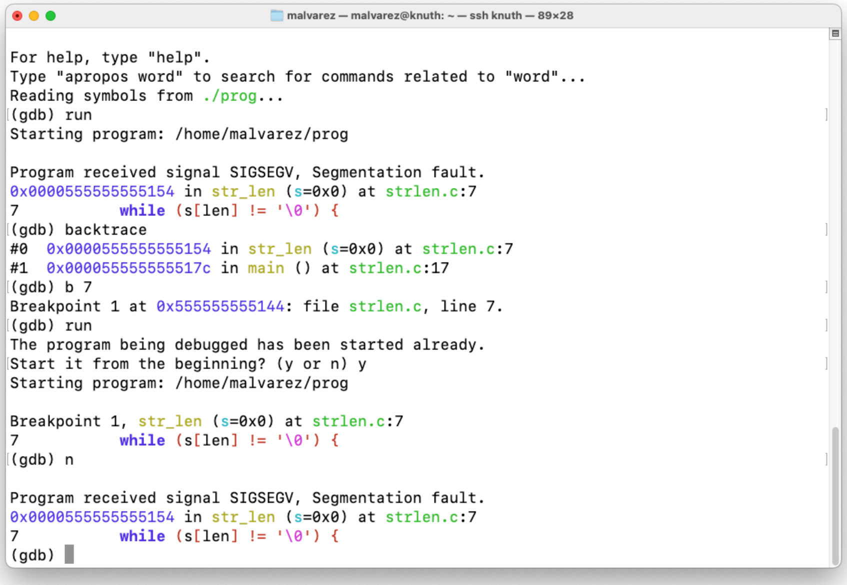
show copying	Display GNU General Public License
show warranty	There is NO WARRANTY for GDB. Display full no-warranty statement.

Example 1

```c
#include <stdio.h>
#include <stdint.h>

uint32_t str_len (const char *s) {
    uint32_t len = 0;

    while (s[len] != '\0') {
        len ++;
    }

    return len;
}

int main () {
    char *str = NULL;

    printf ("Length = %u\n", str_len(str));

    return 0;
}
```



```
malvarez@knuth:~$ gcc -Wall -g strlen.c -o prog
malvarez@knuth:~$ ./prog
Segmentation fault
malvarez@knuth:~$ 
```



```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./prog...
(gdb) run
Starting program: /home/malvarez/prog

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555154 in str_len (s=0x0) at strlen.c:7
7           while (s[len] != '\0') {
(gdb) backtrace
#0  0x0000555555555154 in str_len (s=0x0) at strlen.c:7
#1  0x000055555555517c in main () at strlen.c:17
(gdb) b 7
Breakpoint 1 at 0x555555555144: file strlen.c, line 7.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/malvarez/prog

Breakpoint 1, str_len (s=0x0) at strlen.c:7
7           while (s[len] != '\0') {
(gdb) n

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555154 in str_len (s=0x0) at strlen.c:7
7           while (s[len] != '\0') {
(gdb) 
```

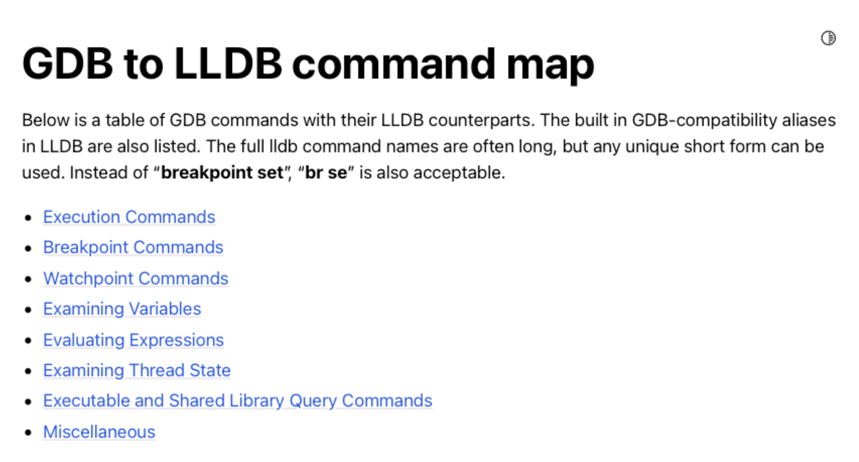
Example 2

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

u_int32_t str_len(const char *s) {
    u_int32_t len = 0;
    while(s[len] != '\0') {
        len ++;
    }
    return len;
}

void str_reverse(const char *src, char *tgt, uint32_t n) {
    u_int32_t start = 0;
    u_int32_t end = n - 1;

    while(end >= 0) {
        tgt[end] = src[start];
        end --;
        start ++;
    }

    tgt[start] = '\0';
}

int main() {
    char str[] = "C for System Programming";
    char *reversed;
    u_int32_t len = str_len(str);

    reversed = malloc(len + 1);
    str_reverse(str, reversed, len);

    printf("%s\n", reversed);

    free(reversed);
    return 0;
}
```

Practice

- Complete the following tasks and submit a report to gradescope in text format
  - 1) compile the program with -Wall and -g, report any warnings/errors
  - 2) run the program in the shell, report the output
  - 3) start gdb
    - 3.1) run the program with no breakpoints, report the output of this command
    - 3.2) print the backtrace and report the function that is causing the problem
    - 3.3) set a breakpoint at the first line of the problematic function, run the program, making sure it stops at the breakpoint, then inspect the local variables with `info locals`, report and explain the result
    - 3.4) for each of the local variables, execute the `watch <local>` command, report the output of each watch command
    - 3.5) run each line at a time with the `step` command, paying attention to the output generated by the watch commands, until the program crashes, then report your findings and explain what is the exact cause of the crash
    - quit gdb
  - 4) report a possible solution to the problem

LLDB



GDB to LLDB command map

Below is a table of GDB commands with their LLDB counterparts. The built in GDB-compatibility aliases in LLDB are also listed. The full lldb command names are often long, but any unique short form can be used. Instead of "**breakpoint set**", "**br se**" is also acceptable.

- Execution Commands
- Breakpoint Commands
- Watchpoint Commands
- Examining Variables
- Evaluating Expressions
- Examining Thread State
- Executable and Shared Library Query Commands
- Miscellaneous

https://lldb.llvm.org/use/map.html