Debugging with \texttt{gdb}
Preparing

Source files should be compiled with –g.
• Generates a symbol table
• Turns off optimizations
Starting `gdb`

```plaintext
> gdb gdb_test1
   (gdb)

   – or –

> gdb
   (gdb) target exec gdb_test1
```
Running a Program:

run (r)
> gdb gdb_test1
(gdb) r
[Inferior 1 (process 6260) exited normally]
(gdb)
Specifying Command Line Arguments

> gdb gdb_test1
(gdb) r aaa bbb
aaa
bbb
[Inferior 1 (process 6260) exited normally]
(gdb)

—or—

(gdb) set args mmm nnn
(gdb) r
mmm
nnn
[Inferior 1 (process 6260) exited normally]
(gdb)
help

(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without ...
user-defined -- User-defined commands

Type "help" followed by a class name for a list of ... Type "help all" for the list of all commands.
Type "help" followed by command name for full ...
Type "apropos word" to search for commands related to ...
Command name abbreviations are allowed if unambiguous.
(gdb)
help classname

(gdb) help files
Specifying and examining files.

List of commands:

add-symbol-file -- Load symbols from FILE
cd -- Set working directory to DIR for debugger and ...
core-file -- Use FILE as core dump for examining memory ...
directory -- Add directory DIR to beginning of search ...
dll-symbols -- Load dll library symbols from FILE
edit -- Edit specified file or function
exec-file -- Use FILE as program for getting contents ...
file -- Use FILE as program to be debugged
forward-search -- Search for regular expression (see ...
generate-core-file -- Save a core file with the ...
list -- List specified function or line
load -- Dynamically load FILE into the running program

(gdb)
apropos word

(gdb) apropos display
d -- Delete some breakpoints or auto-display expressions
del -- Delete some breakpoints or auto-display expressions
delete -- Delete some breakpoints or auto-display expressions
delete breakpoints -- Delete some breakpoints or auto-display expressions
delete display -- Cancel some expressions to be...
disable display -- Disable some expressions to be...
display -- Print value of expression EXP each time the ...
enable display -- Enable some expressions to be...
focus -- Set focus to named window or next/prev window
fs -- Set focus to named window or next/prev window
info auxv -- Display the inferior's auxiliary vector
info display -- Expressions to display when program stops
info threads -- Display currently known threads
info w32 selector -- Display selectors infos
info w32 thread-information-block -- Display thread information block
info w32 tib -- Display thread information block
info win -- List of all displayed windows
...
(gdb)
Setting Breakpoints: break (b)

break line #
break file.c:line #

(gdb) b 35
Breakpoint 1 at 0x40125e: file gdb_test2a.c, line 32.
(gdb) b gdb_test2b.c:13
Breakpoint 2 at 0x40134e: file gdb_test2b.c, line 13.
(gdb) r
Starting program: /cygdrive/f/Isilon - C Programming ...
[New Thread 5816.0x1494]
[New Thread 5816.0x17fc]
test1, line 1
test1, line 2
test1, line 3

Breakpoint 1, test2 (param=50) at gdb_test2a.c:35
35 int inx = param * 2;
(gdb)
Continuing after a Breakpoint

continue (c)

Breakpoint 1, test2 (param=50) at gdb_test2a.c:35
35    int inx = param * 2;

(gdb) c
Continuing.
inx: 100
jnx: 25

Breakpoint 1, test2 (param=100) at gdb_test2a.c:35
35    int inx = param * 2;

(gdb)
Examining Values

print expression

Breakpoint 1, test2 (param=100) at gdb_test2a.c:35
35      int inx = param * 2;
(gdb) p param
$1 = 100
(gdb) p param * 1.2
$2 = 120
(gdb)
Exiting `gdb`

```
quit
(gdb) quit
$`
```
Exercises

For these exercises, download the `gdb_test` directory from the class website and execute `Makefile`.

1. Use `gdb` to start `gdb_test2`. Set breakpoints at line 46 in `gdb_test2a.c`, and line 22 in `gdb_test2b.c`. Run the program. When the program stops at line 46 examine the values of `area` and `radius`, and continue. When the program stops at line 22 examine the values of `leg1`, `leg2` and `hypot`, and continue.
Breakpoints (1)

```plaintext
break line #
break filename:line #
break functionname
break filename:functionname

(gdb) b test2
Breakpoint 1 at 0x40125e: file gdb_test2a.c, line 35.
(gdb) b gdb_test2b.c:TEST2_hypot
Breakpoint 2 at 0x401312: file gdb_test2b.c, line 9.
(gdb)
```
Breakpoints (2)

info breakpoints
delete breakpoint #
disable breakpoint #
enable breakpoint #

(gdb) b test2
Breakpoint 1 at 0x40125e: file gdb_test2a.c, line 35.
(gdb) b gdb_test2b.c:TEST2_hypot
Breakpoint 2 at 0x401312: file gdb_test2b.c, line 9.
(gdb) info breakpoints
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0x0040125e in test2 at
gdb_test2a.c:35
2   breakpoint     keep y   0x00401312 in TEST2_hyp ...
(gdb) delete 1
(gdb) disable 2
(gdb) info breakpoints
Num Type           Disp Enb Address    What
2   breakpoint     keep n   0x00401312 in TEST2_hyp ...
(gdb) enable 2
(gdb) info breakpoints
Num Type           Disp Enb Address    What
2   breakpoint     keep y   0x00401312 in TEST2_hyp ...
Breakpoints (3)

break line # if condition
break functionname if condition

(gdb) b TEST2_hypot if leg2 > 5
Breakpoint 1 at 0x40132a: file gdb_test2b.c, line 9.
(gdb) r
Starting program: /cygdrive/l/Isilon - C Programming ...
test1, line 1
test1, line 2
test1, line 3
inx: 100 ...
inx: 2000
jnx: 500
radius: 5, area: 78.5397
radius: 100, area: 31415.9
radius: 1e+101, area: 3.14159e+202
leg1: 3.000000, leg2: 4.000000, hypot: 5.000000

Breakpoint 1, TEST2_hypot (leg1=5, leg2=12) at ...
9     double  sum     = leg1 * leg1 + leg2 * leg2;
(gdb)
Breakpoints (4)

command breakpoint #
> cmd 1
> cmd 2
...
> cmd n
end

(gdb) b 47
Breakpoint 1 at 0x4012ec: file gdb_test2a.c, line 47.
(gdb) command 1
Type commands for breakpoint(s) 1, one per line.
End with a line saying just "end".
>p area
>p radius
>c
>end
(gdb) r
...

Continued on next slide
Breakpoints (4 continued)

...  

Breakpoint 1, test3 (radius=5) at gdb_test2a.c:47
47          printf( "radius: %g, area: %g\n", radius, ...
$1 = 78.539749999999998
$2 = 5
radius: 5, area: 78.5397

Breakpoint 1, test3 (radius=100) at gdb_test2a.c:47
47          printf( "radius: %g, area: %g\n", radius, ...
$3 = 31415.900000000001
$4 = 100
radius: 100, area: 31415.9
...

Changing Variables

set variablename=expression

Breakpoint 1, TEST2_hypot (leg1=3, leg2=4) at ...
9     double sum   = leg1 * leg1 + leg2 * leg2;
(gdb) set leg1=10
(gdb) p leg1
$2 = 10
(gdb) set leg1=10*leg2
(gdb) p leg1
$3 = 40
(gdb)
Exercises

2. Start `gdb` with `gdb_test3`. Set the debugger to break at line 32 only if `rcode` is greater than 1500. Run the executable with no command line arguments.

3. Note that `gdb_test3` will crash if run with a command line argument. Start `gdb` with `gdb_test3`. Set the debugger to break at function `test2` only if `parm2` is equal to 0. Run the executable with one command line argument.

4. Start `gdb` with `gdb_test3`. Set the debugger to break at function `test2` if `parm2` is equal to 0. Use command `breakpoint #` to set `parm2` to 1, print an error message, and continue. Run the executable with one command line argument.
Conditions

condition breakpoint # expression

(gdb) b getSubstr
Breakpoint 1 at 0x804851e: file gdb_test4.c, line 20.
(gdb) cond 1 start > strlen( str )
(gdb) r
Starting program: /home/jack/test/gdb_test/gdb_test4
ck Co
anle

Breakpoint 1, getSubstr (start=7, len=9, str=0x8048633 ... 
at gdb_test4.c:20
20 char *substr = malloc( len + 1 );
(gdb)
Step-over

next [num]
continue to the next [num] line(s) skipping over functions

(gdb) b 9
Breakpoint 1 at 0x40117e: file gdb_test5.c, line 9.
(gdb) r
Starting program: /cygdrive/l/Isilon - C Programming ...

Breakpoint 1, main (argc=1, argv=0x28ac40) at gdb_test5.c:9
9          printf( "%s\n", getSubstr( 3, 5, "Chick Cor ...
(gdb) n
10          printf( "%s\n", getSubstr( 2, 4, "Stanley ...
(gdb) n 3
13 printf( "%s\n", getSubstr( 3, 6, "Tarnation" ) );
14 (gdb)
**Step-into**

step [num]
continue to the next [num] line(s) stepping into functions when indicated

(gdb) b 9
Breakpoint 1 at 0x40117e: file gdb_test5.c, line 9.
(gdb) r
Starting program: /cygdrive/L/Isilon - C Programming ...

Breakpoint 1, main (argc=1, argv=0x28ac40) at gdb_test5.c:9
9           printf( "%s\n", getSubstr( 3, 5, "Chick Cor ... 
(gdb) s
getSubstr (start=3, len=5, str=0x402080 "Chick Corea") ... 
22          char *substr = malloc( len + 1 );
(gdb) s 3
26          return substr;
(gdb)
Finish

finish
continue until the current function returns

(gdb) b 9
Breakpoint 1 at 0x40117e: file gdb_test5.c, line 9.
(gdb) r
Starting program: /cygdrive/L/Isilon - C Programming 2/ ...

Breakpoint 1, main (argc=1, argv=0x28ac40) at gdb_test5.c:9
  9    printf( "\%s\n", getSubstr( 3, 5, "Chick Cor ... 
(gdb) s
getSubstr (start=3, len=5, str=0x402080 "Chick Corea") ...
  22          char *substr = malloc( len + 1 );
(gdb) fin
Run till exit from #0 getSubstr (start=3, len=5, str=0 ...
  at gdb_test5.c:22
0x0040119a in main (argc=1, argv=0x28ac40) at gdb_test5.c:9
  9    printf( "\%s\n", getSubstr( 3, 5, "Chick Cor ... 
Value returned is $1 = 0x80028e28 "ck Co"
(gdb)
Stack Frames (1)

info frame
print information about the current stack frame

(gdb) b funk3
Breakpoint 1 at 0x4011f7: file gdb_test6.c, line 31.
(gdb) r
Starting program: /cygdrive/L/Isilon - C Programming 2/ ...

Breakpoint 1, funk3 (parm=4) at gdb_test6.c:31
31          int inx = 2;
(gdb) info frame
Stack level 0, frame at 0x28ab90:
eip = 0x4011f7 in funk3 (gdb_test6.c:31); saved eip ...
called by frame at 0x28abc0
source language c.
Arglist at 0x28ab88, args: parm=4
Locals at 0x28ab88, Previous frame's sp is 0x28ab90
Saved registers:
ebp at 0x28ab88, eip at 0x28ab8c
(gdb)
Stack Frames (2)

info frame num
print information about the num\textsuperscript{th} frame above the current stack frame

(gdb) info frame 1
Stack frame at 0x28abc0:
  eip = 0x4011e5 in funk2 (gdb_test6.c:25); saved eip ...
called by frame at 0x28abf0, caller of frame at 0x28ab90
source language c.
Arglist at 0x28abb8, args: parm=3
Locals at 0x28abb8, Previous frame's sp is 0x28abc0
Saved registers:
  ebp at 0x28abb8, eip at 0x28abbc
(gdb)
Stack Frames (3)

backtrace
print a back-trace of the frame stack

(gdb) b funk3
Breakpoint 1 at 0x4011f7: file gdb_test6.c, line 31.
(gdb) r
Starting program: /cygdrive/L/Isilon - C Programming 2/ ...

Breakpoint 1, funk3 (parm=4) at gdb_test6.c:31
31          int inx = 2;
(gdb) bt
#0  funk3 (parm=4) at gdb_test6.c:31
#1  0x004011e5 in funk2 (parm=3) at gdb_test6.c:25
#2  0x004011c1 in funk1 (parm=5) at gdb_test6.c:18
#3  0x00401192 in main (argc=1, argv=0x28ac40) at gdb_test6.c:11
(gdb)
Stack Frames (5)

up [num]
down [num]
crawl up or down the frame stack

(gdb) up 2
#2 0x004011c1 in funk1 (parm=5) at gdb_test6.c:18
18 int rcode = parm * funk2(inx);
(gdb) p inx
$1 = 3
(gdb) down
#1 0x004011e5 in funk2 (parm=3) at gdb_test6.c:25
25 int rcode = parm * funk3(inx);
(gdb) p inx
$2 = 4
(gdb)
Examining Source Code (1)

list
List ten lines of code in the neighborhood of the current source line; ...

(gdb) r
Starting program: /cygdrive/L/Isilon - C Programming 2/ ...

Breakpoint 1, funk2 (parm=3) at gdb_test6.c:24
24 int inx = 4;
(gdb) l
19 return rcode;
20 }
21
22 static int funk2( int parm )
23 {
24 int inx = 4;
25 int rcode = parm * funk3( inx );
26 return rcode;
27 }
28 ...
Examining Source Code (2)

list
... multiple list commands successively print the next ten lines of code.

28 ...
(gdb) l
29     static int funk3( int parm )
30     {
31         int inx = 2;
32         int rcode = parm * funk2( inx );
33         return rcode;
34     }
(gdb)
Examining Source Code (3)

Also:

list -
list +
list linenum
list file:linenum
list function
list file:function
Exercises (1 of 2)

5. `gdb_test7` prints out the length of the hypotenuse of various right triangles. Execute it and note that at some point it begins printing invalid values. Open `gdb_test7` in `gdb`. Put a breakpoint at the return statement in `hypotenuse`. Use command to automatically print out the value of `hypot`. Run the program and let it break ten times. Now use the condition command to add if `hypot < 0` to your breakpoint. Resume execution.

6. Open `gdb_test7` in `gdb`, and put a breakpoint at the `puts`. Use `next` to single-step through the for loop several times. Use `step` to step-into `hypotenuse`. Continue to step through `hypotenuse` until you return to `main`. Once again step-into `hypotenuse`. Use `finish` to continue to the end of the function.

7. Open `gdb_test2` in `gdb`, and put a breakpoint at the `printf` in function `print_tri` and the end of `gdb_test2b.c`. Run until the program breaks. Print a backtrace of the call stack. Use `info` to examine the current stack frame; use `info` to examine the stack frame that contains `main`. Crawl up the stack frame one level, and examine the local variables in that frame; crawl up one more level, then down two levels.
Exercises (2 of 2)

8. Open `gdb_test2` in `gdb`, and put a breakpoint at the first line of function `test2` in `gdb_test2a.c`. Run until the program breaks. Use `print` to:
   a) Examine ten lines of source code around your current position.
   b) Examine the next ten lines of source code.
   c) Examine the previous ten lines of source code, then the ten lines before that.
   d) Examine the ten lines of source code around line 15 in the current source file.
   e) Examine the ten lines of source code around line 10 in `gdb_test2b.c`.
   f) Examine the ten lines of source code around function `TEST2_leg` in `gdb_test2b.c`. 
Print Options (1 of 2)

- `p/x variable` - Print in hex
- `p/d variable` - Print as signed integer
- `p/u variable` - Print as unsigned integer
- `p/o variable` - Print in octal
- `p/t variable` - Print in binary (1 byte)
- `p/c variable` - Print integer as character
- `p/f variable` - Print as floating point number
- `p/a variable` - Print as hex address

(gdb) b 41
Breakpoint 1 at 0x4012c2: file gdb_test2a.c, line 41.
(gdb) r
Starting program: /cygdrive/l/Isilon - C Programming 2/ ... test1, line 1
test1, line 2
test1, line 3
inx: 100
jnx: 25
Breakpoint 1, test2 (param=50) at gdb_test2a.c:41
41          return knx;
(gdb) p/x knx
$1 = 0x7d
(gdb) p/o knx
$2 = 0175
(gdb) p/t knx
$3 = 1111101
(gdb) p/a knx
$4 = 0x7d
> gdb gdb_test12
  (gdb) b funk2
Breakpoint 1 at 0x4011f0: file gdb_test12.c, line 27.
  (gdb) b funk1
Breakpoint 2 at 0x4011dc: file gdb_test12.c, line 22.
  (gdb) commands
Type commands for breakpoint(s) 2, one per line.
End with a line saying just "end".
>if ( $INX > 0 )
  >set --$INX
  >c
  >end
>end
  (gdb) set $INX = 5
  (gdb) r
  ...

Conditional Commands, Global Variables (1 of 2)
Conditional Commands, Global Variables (2 of 2)

Breakpoint 1, funk2 () at gdb_test12.c:28
28        puts( "funk2" );
(gdb) c
Continuing.
funk2

Breakpoint 2, funk1 () at gdb_test12.c:23
23        printf( "funk1: %d\n", temp++ );
funk1: 9

Breakpoint 2, funk1 () at gdb_test12.c:23
23        printf( "funk1: %d\n", temp++ );
(gdb)
Custom Commands (1)

```plaintext
define mcmds
> cmd1
> cmd2
. . .
> end

> gdb gdb_test2
(gdb) b TEST2_hypot
Breakpoint 1 at 0x40132a: file gdb_test2b.c, line 9.
(gdb) define test_cmd1
Type commands for definition of "test_cmd1".
End with a line saying just "end".
>info frame
>p leg1
>c
>end
(gdb) r
. . .
```
Custom Commands (2)

Breakpoint 1, TEST2_hypot (leg1=3, leg2=4) at gdb_test2b.c:9
9 double sum = leg1 * leg1 + leg2 * leg2;
(gdb) test_cmd1
Stack level 0, frame at 0x28ac00:
eip = 0x40132a in TEST2_hypot (gdb_test2b.c:9); saved ...
called by frame at 0x28ac20
source language c.
Arglist at 0x28abf8, args: leg1=3, leg2=4
Locals at 0x28abf8, Previous frame's sp is 0x28ac00
Saved registers:
ebp at 0x28abf8, eip at 0x28abfc
$1 = 3
leg1: 3.000000, leg2: 4.000000, hypot: 5.000000

Breakpoint 1, TEST2_hypot (leg1=5, leg2=12) at gdb_test2b.c:9
9 double sum = leg1 * leg1 + leg2 * leg2;
(gdb)
Custom Commands and Arguments (1)

Commands can have up to ten arguments, $\text{arg0}$ through $\text{arg9}$.

(gdb) define px
Type commands for definition of "px".
End with a line saying just "end".
>p/x $\text{arg0}$
>end
(gdb) p temp
$1 = 10$
(gdb) px temp
$2 = 0xa$
(gdb)
Custom Commands and Arguments (2)

$\text{argc tells you how many arguments there are.}$

(gdb) define px
Type commands for definition of "px".
End with a line saying just "end".
>p/x $\text{arg0}$
@end
(gdb) p temp
$1 = 10$
(gdb) px temp
$2 = 0xa$
(gdb) define test
Type commands for definition of "test".
End with a line saying just "end".
>p $\text{argc}$
@end
(gdb) test aaa bbb ccc ddd
$3 = 4$
(gdb)
The shell Command

(gdb) shell printf "\ec"
(gdb) shell less gdb_test12.c
#include <stdio.h>
#include <unistd.h>

static void funk1( void );
static void funk2( void );

int main( int argc, char **argv )
{
  for ( int inx = 0 ; inx < 20 ; ++inx )
  {
    funk1();
  ...

The documentation Command

Documents a user-defined command.

(gdb) define cls
Type commands for definition of "cls".
End with a line saying just "end".
>shell printf "\ec"
>end
(gdb) doc cls
Type documentation for "cls".
End with a line saying just "end".
>Clears the screen via the shell.
>end
(gdb) help cls
Clears the screen via the shell.
(gdb)
Command Files (1)

```
gdb_test2.gdb

b gdb_test2a.c:41
commands 1
  p knx
  if ( param < 1000 )
    c
  end
end
b gdb_test2b.c:TEST2_hypot if leg1 == 5
commands 2
  p leg2
end
```
Command Files (2)

> gdb gdb_test2 -x gdb_test2.gdb
Breakpoint 1 at 0x4012c2: file gdb_test2a.c, line 41.
Breakpoint 2 at 0x40132a: file gdb_test2b.c, line 9.
(gdb) info break
Num Type           Disp Enb Address    What
1    breakpoint     keep y   0x004012c2 in test2 at ...
p knx
   if ( param < 1000 )
   c
   end
2    breakpoint     keep y   0x0040132a in TEST2_hypot...
   stop only if leg1 == 5
   p leg2
(gdb)
Command Files (3)

To load a command file:

- Put a `.gdbinit` file in `$HOME`
- Put a `.gdbinit` file in the directory you execute gdb from
- Use `-x filename` on the command line (you can use multiple `-x` options)
- Use `source filename` on from within gdb.
Sample .gdbinit File

# default lines displayed by list command
set listsize 24
# default prompt
set prompt %
# don't ask me "are you sure you want to do that?"
set confirm off
# save command line history
set history save on
# save command line history in file...
set history file ~/.gdbhistory
define px
    p/x $arg0
end
document px
    Print argument in hex
end
define cls
    shell printf "\ec"
end
document cls
    clear the screen
end
Watchpoints (1)

- `watch expr`: break when `expr` changes
- `rwatch expr`: break when `expr` is read
- `awatch expr`: break when `expr` changes or is read

(gdb) watch controls_.curr_x
Hardware watchpoint 1: controls_.curr_x
(gdb) rwatch controls_.color
Hardware read watchpoint 2: controls_.color
(gdb) r
Starting program: /cygdrive/L/Isilon - C Programming 2/ ... 2/Slides/Unit4/gdb_test/gdb_test8
[New Thread 5312.0xb54]
[New Thread 5312.0x1bb8]
Hardware watchpoint 1: controls_.curr_x

Old value = 100
New value = 150
advance (xxx=50, yyy=0) at gdb_test8a.c:45
45 if ( yyy != 0 )
...

The number of watchpoints you have may be limited by the capabilities of your hardware.
Watchpoints (2)

... (gdb) c
Continuing.
Hardware read watchpoint 2: controls_.color

Value = 16776960
0x00401407 in TEST8_funk2 (controls=0x402000, param=0) ...
15        if ( controls->color > 0 )
Calling Functions

call functionname

(gdb) b 9
Breakpoint 1 at 0x4011b6: file gdb_test9.c, line 9.
(gdb) r
Starting program: /cygdrive/L/Isilon - C Programming 2/ ...
120
720

Breakpoint 1, main (argc=1, argv=0x28ac40) at gdb_test9.c:9
9           printf( "%d\n", factorial( 7 ) );
(gdb) call factorial( 3 )
$1 = 6
(gdb)
Analyzing Core Dumps

gdb executablename corefile

$ gdb gdb_test11 core
...
Core was generated by `./gdb_test11'.
Program terminated with signal 8, Arithmetic exception.
#0 0x080484e6 in funk4 (param=0) at gdb_test11.c:42
42 int result = 12 / param;
(gdb) bt
#0 0x080484e6 in funk4 (param=0) at gdb_test11.c:42
#1 0x080484ce in funk3 (param=0) at gdb_test11.c:35
#2 0x08048498 in funk2 (param=5) at gdb_test11.c:27
#3 0x08048476 in funk1 (param=1) at gdb_test11.c:19
#4 0x0804842b in main (argc=1, argv=0xbfc0c844) at gdb_test11.c:11
(gdb) up
#1 0x080484ce in funk3 (param=0) at gdb_test11.c:35
35 int result = funk4( val );
(gdb) p val
$1 = 0
(gdb)

To get a core dump you may have to configure your shell; try:
ulimit -c unlimited
Exercises (1 of 2)

9. Open gdb_test10 in gdb; put a temporary breakpoint at the `for` statement (`tbreak 9`) and run the program. When the program breaks at your breakpoint, set a write-watchpoint on `test_var1`, and a read-watchpoint on `test_var2` and continue. Each time the program breaks determine which watchpoint caused the break, examine the value of `inx`, and continue.

10. Open gdb_test2 in gdb; put a breakpoint at any statement in function `test1` and run the program. When the program breaks, call `TEST2_hypot` passing arguments 5, 12. The value returned by the function will be stored in an internal variable ($1 if you executed the exercise exactly as described). Print the value of the variable as a signed integer, then in hex and octal.

11. Download `strtok_fail` from the class website. Build and execute `tok_test`; it should crash and dump a core file. Use `gbd` to examine the core file, and determine the cause of the crash. Fix the problem and rerun the test to make sure that the issue is resolved.
Exercises (2 of 2)

12. In your home directory, make a `.gdbinit` file like the one in the slides. Make another `.gdbinit` file in your test directory that contains the commands:

   ```bash
   set logging file gdb.log
   set logging on
   ```

   If you get the error message “... auto-loading has been declined by your ‘auto-load safe-path’...” add
   ```bash
   set auto-load safe-path .
   ```

   to ~/.gdbinit. Start gdb and verify that ~/.gdbinit was correctly read. Set a temporary breakpoint at TEST2_hypot, run the program, and continue to completion. Quit gdb and examine the file gdb.log.

13. Make a command file to set breakpoints at TEST2_hypot and TEST2_leg. Open gdb_test2 in gdb and use the source command to load your command file. Use `info break` to verify that the breakpoints are set.

14. How would you write a shell script to generate a gdb command file to set breakpoints at every function in a list of source files?
When a program communicate via signals, default signal handling may need to be modified.

```c
int main( int argc, char **argv )
{
    puts( "Starting..." );
    if ( signal( SIGHUP, handler ) == SIG_ERR )
        abort();
    funk1();
    puts( "Done..." );

    return 1;
}
static void funk1( void )
{
    raise( SIGHUP );
}
static void handler( int sig )
{
    printf( "Signal %d received\n", sig );
}
```

Signal Handling (2)

$ gdb gdb_test14
...
(gdb) info signals SIGHUP
Signal Stop Print Pass to program Description
SIGHUP Yes Yes Yes Yes Hangup
(gdb) r
Starting program:
/home/jack/isilon/Test/gdb_test/gdb_test14
Starting...

Program received signal SIGHUP, Hangup.
0x00007ffffff7a51425 in raise () from /lib/x86_64-linux-gnu/libc.so.6
(gdb) c
Continuing.
Signal 1 received
Done...
[Inferior 1 (process 27225) exited with code 01]
(gdb)
Signal Handling (3)

(gdb) handle SIGHUP nopass
Signal        Stop      Print      Pass to program Description
SIGHUP        Yes       Yes       No              Hangup
(gdb) (gdb) r
Starting program:
/home/jack/isilon/Test/gdb_test/gdb_test14
Starting...

Program received signal SIGHUP, Hangup.
0x000007ffffff7a51425 in raise () from /lib/x86_64-
gnu/libc.so.6
(gdb) c
Continuing
Done...
[Inferior 1 (process 27231) exited with code 01]
(gdb)
Signal Handling (4)

(gdb) handle SIGHUP pass
Signal Stop Print Pass to program Description
SIGHUP Yes Yes Yes Hangup

(gdb) handle SIGHUP nostop
Signal Stop Print Pass to program Description
SIGHUP No Yes Yes Hangup

(gdb) r
Starting program:
/home/jack/isilon/Test/gdb_test/gdb_test14
Starting...

Program received signal SIGHUP, Hangup.
Signal 1 received
Done...[Inferior 1 (process 27225) exited with code 01]
(gdb)
Signal Handling (5)

When a program communicate via signals, default signal handling may need to be modified.

```c
int main( int argc, char **argv )
{
    puts( "Starting..." );
    if ( signal( SIGHUP, handler ) == SIG_ERR )
        abort();
    funk1();
    puts( "Done..." );

    return 1;
}
static void funk1( void )
{
    raise( SIGHUP );
}
static void handler( int sig )
{
    printf( "Signal %d received\n", sig );
}
```
Signal Handling (6)

> gdb gdb_test15
(gdb) info signals SIGINT
Signal        Stop      Print    Pass to program Description
SIGINT        Yes       Yes      No              Interrupt
(gdb) r
Starting program:
/home/jack/isilon/Test/gdb_test/gdb_test15
Starting...
9
8
^C
Program received signal SIGINT, Interrupt.
0x00007ffffffada820 in nanosleep () from /lib/x86_64-...
(gdb) c
Continuing.
7
Signal Handling (7)

(gdb) handle SIGINT pass
SIGINT is used by the debugger.
Are you sure you want to change it? (y or n) y

Signal Stop Print Pass to program Description
SIGINT Yes Yes Yes Yes Interrupt

(gdb) r
Starting program:
/home/jack/isilon/Test/gdb_test/gdb_test15
Starting...
9
8
^C
Program received signal SIGINT, Interrupt.
0x00007fffffff820 in nanosleep () from /lib/x86_64-

(gdb) c
Continuing.

Program terminated with signal SIGINT, Interrupt.
The program no longer exists.
(gdb)
Signal Handling (8)

For the first 15 signals, a number or symbolic name may be used; above 15 use the symbolic name.

handle sig stop
handle sig nostop
handle sig print
handle sig noprint
handle sig pass
handle sig nopass
Exercises

15. Make a gdb command file that sets signal handling for SIGHUP to stop = no, print = yes and pass = yes, and for SIGILL to stop = yes, print = no and pass = no. Open gdb_test15 in gdb using the –x command line option to load your command file. Use info sig SIGHUP and info sig SIGILL to verify that your command file was read and processed as you intended.
Remote Debugging with **gdbserver** (1)

Start gdbserver on the target machine:

```
gdbserver comm program [args]
```

- `comm`
  - A serial device, or host/tcp address

- `program`
  - The program to debug; must be present on the target machine; may be stripped

- `args`
  - Arguments to pass to program
Remote Debugging with `gdbserver` (2)

`gdbserver /dev/com1 gdb_test2`
Runs over serial device `/dev/com1`; there must be a serial connection between host and target machines, perhaps via a null-modem cable.

`gdbserver blackheart:10000 gdb_test2`
Runs over internet to host ‘blackheart’ on port 10000

`gdbserver localhost:10000 gdb_test2`
Runs on ‘localhost,’ port 10000

`gdbserver :10000 gdb_test2`
Runs on ‘localhost,’ port 10000
Remote Debugging with \texttt{gdbserver (3)}

Run \texttt{gdb} on the host machine; there must be a copy of the program on the host, preferably unstripped:
\begin{verbatim}
> gdb gdb_test2
\end{verbatim}

Within \texttt{gdb}, set the remote target:
\begin{verbatim}
(gdb) target remote /dev/com1
\end{verbatim}

Connects via serial port
\begin{verbatim}
(gdb) target remote blackheart:10000
\end{verbatim}

Connects via internet to host ‘blackheart,’ port 10000
\begin{verbatim}
(gdb) target remote localhost:10000
\end{verbatim}

Connects to ‘localhost,’ port 10000
\begin{verbatim}
(gdb) target remote :10000
\end{verbatim}

Connects to ‘localhost,’ port 10000
Remote Debugging with \texttt{gdbserver} (4)

Set breakpoints, etc., just as you normally would. Note that your program is already running but paused on the target machine; to resume execution use \texttt{continue}:

```
> gdb gdb_test2
(gdb) target remote :10000
Remote debugging using :10000
...  
(gdb) b TEST2_hypot
Breakpoint 1 at 0x40081a: file gdb_test2b.c, line 9.
(gdb) c
Continuing.

Breakpoint 1, TEST2_hypot ... at gdb_test2b.c:9
9 double  sum     = leg1 * leg1 + leg2 * leg2;
(gdb)
```
Remote Debugging with *gdbserver* (5)

*gdbserver* offers features not available in *gdb*, such as *tracepoints*. To take advantage of these features during normal debugging, run *gdbserver* on ‘localhost.’
Introduction to Doxygen
Doxygen

Copyright © 1997-2013 by Dimitri van Heesch.
• Distributed under the GNU General Public License
• http://www.stack.nl/~dimitri/doxygen/index.html
• See also: Graphvis, from Lucent: http://www.graphviz.org/
Doxygen

Sample source markup:

/**
 * \file timer.h
 * \brief Public header file for the TIMER module.
 *
 * \defgroup timer TIMER
 * \brief Documentation for the public API of
 * the TIMER module.
 * \{=*/
Doxygen

Sample source markup:

/**
 * Instantiates a new timer. Returns an ID that represents
 * the timer, to be used in calling all other TIMER
 * functions. When the timer is no longer needed, it should
 * be destroyed by calling TIMER_destroy.
 *
 * \return the ID of the newly created timer
 *
 * \see TIMER_destroy_timer
 */

TIMER_ID_t TIMER_create_timer(
    void
);
Doxygen

Common tags:

\author
\brief
\bug
\deprecated
\file
\param
\return
\see
\warning
\pre
Doxygen

Html markup:

/**
 * \htmlonly <style> \endhtmlonly
 * \htmlinclude sub.css
 * \htmlonly </style> \endhtmlonly
 * \page sources_page Reference Books and Other ...
 *
 * \anchor hands
 * \htmlonly <p class = "reference_book"> 
 * \endhtmlonly
 * <i>C A Reference Manual, Fifth Edition</i><br>
 * by Samuel P. Harbison, and Guy L. Steele Jr.,
 * Prentice Hall, 2002
 * ...
This project consists of a series of modules that encapsulate utilities and data structures for the construction of applications. They include the following:

- This module contains miscellaneous utilities that are used by all the other modules.
- ...

...
Macros:

```c
/**
 * \brief Allocates space for a var of type \a type.
 * \param type The type name ...
 *
 * This is a convenient utility for quickly and
 * clearly allocating space for a value of some type.
 *
 * \b Example
 * \code
 * typedef struct rec_s
 * {
 *     int    id;
 *     char   name[81];
 * } REC_t, *REC_p_t;
 * . . .
 * REC_p_t rec = ISI_NEW( REC_t );
 * \endcode
 *
#define ISI_NEW( type )
    ((type *)ISI_malloc( sizeof(type) ))
* ...
```
Structs:
/**
 * Structure to encapsulate a timer. startTime_ contains
 * the time that the timer was started; a value of 0 means
 * that the timer isn't running elapsedTime_ keeps track of
 * the time so far elapsed while the timer is paused.
 */
typedef struct timer__control_s
{
    /** Start time; 0 when not running or paused. */
    time_t startTime_;
    /** Time elapsed before timer was paused. */
    size_t elapsedTime_;
} TIMER__CONTROL_t, *TIMER__CONTROL_p_t;
Doxygen

Configuration options:

- PROJECT_NAME = ISILIB
- PROJECT_LOGO =
- OUTPUT_LANGUAGE = English
- JAVADOC_AUTOBRIEF = NO
- HAVE_DOT = YES
- CALL_GRAPH = YES
- GENERATE_MAN = NO
- GENERATE_XML = NO
- GENERATE_LATEX = NO
- INPUT_FILTER = ./dox_filter
Doxygen

Executing:

$ doxygen isi.config
$ less docwarnings.log

.../timerp.h:27: warning: Member TIMER__CONTROL_p_t
  (typedef) of file timerp.h is not documented.