

Systems Programming Laboratory, Spring 2022

Introduction to gdb

Abhijit Das
Arobinda Gupta

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

January 25, 2022

Why debuggers?

- Writing error-free codes (particularly large ones) is every programmer's dream.
 - Compilation errors
 - Logical errors
 - Runtime errors
- The compiler is “*never*” faulty.
- If your program does not run, question your understanding and your code first.
- Debuggers help you out there.
 - You can step through your code line by line.
 - You can keep on watching variables.
 - No need to write diagnostic printf's.
 - ...
- The GNU debugger (gdb) is a popular choice of this day.
- Several graphic debuggers (like `xxgdb`) are built on top of `gdb`.

First example: Area of a triangle

taea.c

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

int main ()
{
    int x1, y1, x2, y2, x3, y3;
    double area;

    printf("Program to calculate the area of a triangle with integer coordinates\n");

    printf("Enter the coordinates of the first corner: ");
    scanf("%d%d", &x1, &y1);
    printf("Enter the coordinates of the second corner: ");
    scanf("%d%d", &x2, &y2);
    printf("Enter the coordinates of the third corner: ");
    scanf("%d%d", &x3, &y3);

    area = x1 * (y2 - y3);
    area += x2 * (y1 - y3);
    area += x3 * (y1 - y2);
    if (area < 0) area = -area;
    area /= 2;

    printf("Area of the triangle is %lf\n", area);

    exit(0);
}
```

Running codes with gdb

```
$ gcc -Wall -g tarea.c
$ gdb ./a.out
GNU gdb (Ubuntu 9.2-0ubuntu1 20.04) 9.2
Copyright (C) 2020 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:
<http://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 ./a.out...
(gdb) run
Starting program: /home/abhij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out
Program to calculate the area of a triangle with integer coordinates
Enter the coordinates of the first corner: 3 8
Enter the coordinates of the second corner: 6 -7
Enter the coordinates of the third corner: -1 5
Area of the triangle is 16.500000
[Inferior 1 (process 6730) exited normally]
(gdb) quit
$
```

Listing your code

- list: Keep on listing your code, ten lines at a time.
- list m,n: List from Line m to Line n of your code.
- list k: List ten lines with Line k at the center.

```
(gdb) list
1      #include <stdio.h>
2      #include <stdlib.h>
3
4      int main ()
5      {
6          int x1, y1, x2, y2, x3, y3;
7          double area;
8
9          printf("Program to calculate the area of a triangle with integer coordinates\n");
10
(gdb) list
11      printf("Enter the coordinates of the first corner: ");
12      scanf("%d%d", &x1, &y1);
13      printf("Enter the coordinates of the second corner: ");
14      scanf("%d%d", &x2, &y2);
15      printf("Enter the coordinates of the third corner: ");
16      scanf("%d%d", &x3, &y3);
17
18      area = x1 * (y2 - y3);
19      area += x2 * (y1 - y3);
20      area += x3 * (y1 - y2);
```

Listing your code: Continued

```
(gdb) list
21     if (area < 0) area = -area;
22     area /= 2;
23
24     printf("Area of the triangle is %lf\n", area);
25
26     exit(0);
27     }
(gdb) list
Line number 28 out of range; tarea.c has 27 lines.
(gdb) list 15
10
11     printf("Enter the coordinates of the first corner: ");
12     scanf("%d%d", &x1, &y1);
13     printf("Enter the coordinates of the second corner: ");
14     scanf("%d%d", &x2, &y2);
15     printf("Enter the coordinates of the third corner: ");
16     scanf("%d%d", &x3, &y3);
17
18     area = x1 * (y2 - y3);
19     area += x2 * (y1 - y3);
(gdb) list 18,22
18     area = x1 * (y2 - y3);
19     area += x2 * (y1 - y3);
20     area += x3 * (y1 - y2);
21     if (area < 0) area = -area;
22     area /= 2;
(gdb)
```

Set a breakpoint and step through your code

```
(gdb) break 11
Breakpoint 1 at 0x11d0: file tarea.c, line 11.
(gdb) run
Starting program: /home/abhiij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out
Program to calculate the area of a triangle with integer coordinates

Breakpoint 1, main () at tarea.c:11
11     printf("Enter the coordinates of the first corner: ");
(gdb) next
12     scanf("%d%d", &x1, &y1);
(gdb) next
Enter the coordinates of the first corner: 3 8
13     printf("Enter the coordinates of the second corner: ");
(gdb) next
14     scanf("%d%d", &x2, &y2);
(gdb) next
Enter the coordinates of the second corner: 6 -7
15     printf("Enter the coordinates of the third corner: ");
(gdb) next
16     scanf("%d%d", &x3, &y3);
(gdb) next
Enter the coordinates of the third corner: -1 5
18     area = x1 * (y2 - y3);
(gdb) next
19     area += x2 * (y1 - y3);
(gdb) continue
Continuing.
Area of the triangle is 16.500000
[Inferior 1 (process 7195) exited normally]
(gdb) q
```

Some notes about gdb

- **Command shortcuts:** You do not have to type the entire command (like list or next). Certain abbreviations are allowed. Here is a short list.

b	break	h	help
bt	backtrace	i	info
c	continue	l	list
d	delete	n	next
dis	disable	p	print
disp	display	q	quit
do	down	ret	return
en	enable	r	run
f	frame	s	step
fin	finish		

- Command name abbreviations are allowed if unambiguous.
- Hit return to repeat the last command.
- **The *unaltered* source file must be present** for gdb to run correctly.

More about breakpoints

- You can set multiple breakpoints.
- Type **info break** to list all the breakpoints.
- continue stops until the next breakpoint or the end of the program
- Other things that you can do with breakpoints:
 - Disable a breakpoint
 - Enable a disabled breakpoint
 - Delete a breakpoint

```
(gdb) break main
Breakpoint 1 at 0x11a9: file tarea.c, line 5.
(gdb) break 11
Breakpoint 2 at 0x11d0: file tarea.c, line 11.
(gdb) break 18
Breakpoint 3 at 0x1257: file tarea.c, line 18.
(gdb) break 24
Breakpoint 4 at 0x12e5: file tarea.c, line 24.
```

Breakpoint examples

```
(gdb) info break
```

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x000000000000011a9	in main at tarea.c:5
2	breakpoint	keep	y	0x000000000000011d0	in main at tarea.c:11
3	breakpoint	keep	y	0x00000000000001257	in main at tarea.c:18
4	breakpoint	keep	y	0x000000000000012e5	in main at tarea.c:24

```
(gdb) disable 3
```

```
(gdb) info break
```

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x000000000000011a9	in main at tarea.c:5
2	breakpoint	keep	y	0x000000000000011d0	in main at tarea.c:11
3	breakpoint	keep	n	0x00000000000001257	in main at tarea.c:18
4	breakpoint	keep	y	0x000000000000012e5	in main at tarea.c:24

```
(gdb) enable 3
```

```
(gdb) info break
```

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x000000000000011a9	in main at tarea.c:5
2	breakpoint	keep	y	0x000000000000011d0	in main at tarea.c:11
3	breakpoint	keep	y	0x00000000000001257	in main at tarea.c:18
4	breakpoint	keep	y	0x000000000000012e5	in main at tarea.c:24

```
(gdb) delete 3
```

```
(gdb) info break
```

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x000000000000011a9	in main at tarea.c:5
2	breakpoint	keep	y	0x000000000000011d0	in main at tarea.c:11
4	breakpoint	keep	y	0x000000000000012e5	in main at tarea.c:24

```
(gdb)
```

Printing variables and expressions

```
(gdb) break 18
Breakpoint 1 at 0x1257: file tarea.c, line 18.
(gdb) run
Starting program: /home/abhiij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out
Program to calculate the area of a triangle with integer coordinates
Enter the coordinates of the first corner: 3 8
Enter the coordinates of the second corner: 6 -7
Enter the coordinates of the third corner: -1 5

Breakpoint 1, main () at tarea.c:18
18      area = x1 * (y2 - y3);
(gdb) print x1
$1 = 3
(gdb) print x2
$2 = 6
(gdb) print x3
$3 = -1
(gdb) print x1 + x2 + x3
$4 = 8
(gdb) print x1 + x2 * x3
$5 = -3
(gdb) print x1 * (y2 - y3) + x2 * (y1 - y3) + x3 * (y1 - y2)
$6 = -33
(gdb) print $6 / 2
$7 = -16
(gdb)
```

Value history: The printed values are stored as \$1, \$2, ..., and can be accessed later.

Watching variables after every step

```
(gdb) break 16
Breakpoint 1 at 0x123b: file tarea.c, line 16.
(gdb) run
...
Breakpoint 1, main () at tarea.c:16
16     scanf("%d%d", &x3, &y3);
(gdb) n
Enter the coordinates of the third corner: -1 5
18     area = x1 * (y2 - y3);
(gdb) display area
1: area = 6.953355807476115e-310
(gdb) n
19     area += x2 * (y1 - y3);
1: area = -36
(gdb) n
20     area += x3 * (y1 - y2);
1: area = -18
(gdb) n
21     if (area < 0) area = -area;
1: area = -33
(gdb) n
22     area /= 2;
1: area = 33
(gdb) n
24     printf("Area of the triangle is %lf\n", area);
1: area = 16.5
(gdb) continue
Continuing.
Area of the triangle is 16.500000
[Inferior 1 (process 8608) exited normally]
(gdb)
```

Setting variables: The program

setvar.c

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

int main ()
{
    int a, b;

    a = 5;
    b = 8;
    printf("max(%d,%d) = %d\n", a, b, (a >= b) ? a : b);
    printf("max(%d,%d) = %d\n", a, b, (a >= b) ? a : b);
    exit(0);
}
```

Setting variables: Run gdb

```
(gdb) break main
Breakpoint 1 at 0x1169: file setvar.c, line 5.
(gdb) n
The program is not being run.
(gdb) run
Starting program: /home/abhiij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out

Breakpoint 1, main () at setvar.c:5
5      {
(gdb) n
8          a = 5;
(gdb) n
9          b = 8;
(gdb) n
10         printf("max(%d,%d) = %d\n", a, b, (a >= b) ? a : b);
(gdb) n
max(5,8) = 8
11         printf("max(%d,%d) = %d\n", a, b, (a >= b) ? a : b);
(gdb) set var b = 2
(gdb) print b
$1 = 2
(gdb) n
max(5,2) = 5
12         exit(0);
(gdb) n
[Inferior 1 (process 12878) exited normally]
(gdb)
```

Conditions and loops

- Nothing special needs to be done.
- Stepping at the end of the loop goes back to the start of the loop.
- When the loop breaks, the line following the loop is executed.

average.c

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

int main ()
{
    int n, x, sum;

    n = sum = 0;
    printf("Keep on entering non-negative integers. Enter a negative integer to end.\n");
    while (1) {
        printf("Next number: "); scanf("%d", &x);
        if (x < 0) break;
        ++n; sum += x;
    }
    printf("Average of the numbers entered is %lf\n", (double)sum / (double)n);
    exit(0);
}
```

Conditions and loops: Example run

```
(gdb) break main
Breakpoint 1 at 0x11a9: file average.c, line 5.
(gdb) run
Starting program: /home/abhiij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out

Breakpoint 1, main () at average.c:5
5      {
(gdb) n
8      n = sum = 0;
(gdb) n
9      printf("Keep on entering non-negative integers. Enter a negative integer to end.\n");
(gdb) n
Keep on entering non-negative integers. Enter a negative integer to end.
11     printf("Next number: "); scanf("%d", &x);
(gdb) n
Next number: 5
12     if (x < 0) break;
(gdb) n
13     ++n; sum += x;
(gdb) n
11     printf("Next number: "); scanf("%d", &x);
(gdb) n
Next number: 8
12     if (x < 0) break;
(gdb) n
13     ++n; sum += x;
(gdb) n
```


Conditions and loops: Example run (Continued)

```
11         printf("Next number: "); scanf("%d", &x);
(gdb) n
Next number: 4
12         if (x < 0) break;
(gdb) n
13         ++n; sum += x;
(gdb) n
11         printf("Next number: "); scanf("%d", &x);
(gdb) n
Next number: -1
12         if (x < 0) break;
(gdb) n
15         printf("Average of the numbers entered is %lf\n", (double)sum / (double)n);
(gdb) n
Average of the numbers entered is 5.666667
16         exit(0);
(gdb) n
[Inferior 1 (process 10883) exited normally]
(gdb)
```

Function calls: next and step

- **next** does not go into function calls.
- **step** goes into function calls.
- If you want to go into recursive calls, use steps before any such call.
- **finish** or **return** leaves the function without further stepping.
- **finish** continues normally to the end of the function, prints the returned value, and adds this value to the value history.
- **return** leaves the function immediately after you ask gdb to do so.
- You may set an explicit breakpoint at a function name.
- Except for function calls, step and next work the same way.

A program with a non-recursive function

TAREA.c

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

double tarea ( int x1, int y1, int x2, int y2, int x3, int y3 )
{
    double area;

    area = x1 * (y2 - y3);
    area += x2 * (y1 - y3);
    area += x3 * (y1 - y2);
    if (area < 0) area = -area;
    area /= 2;
    return area;
}

int main ()
{
    int x1, y1, x2, y2, x3, y3;
    double area;

    printf("Program to calculate the area of a triangle with integer coordinates\n");
    printf("Enter the coordinates of the first corner: "); scanf("%d%d", &x1, &y1);
    printf("Enter the coordinates of the second corner: "); scanf("%d%d", &x2, &y2);
    printf("Enter the coordinates of the third corner: "); scanf("%d%d", &x3, &y3);
    area = tarea(x1,y1,x2,y2,x3,y3);
    printf("Area of the triangle is %lf\n", area);
    exit(0);
}
```

Skip going inside the function by next

```
(gdb) break main
Breakpoint 1 at 0x124e: file TAREA.c, line 17.
(gdb) r
Starting program: /home/abhiij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out

Breakpoint 1, main () at TAREA.c:17
17      {
(gdb) next
21      printf("Program to calculate the area of a triangle with integer coordinates\n");
(gdb) next
Program to calculate the area of a triangle with integer coordinates
22      printf("Enter the coordinates of the first corner: "); scanf("%d%d", &x1, &y1);
(gdb) next
Enter the coordinates of the first corner: 3 8
23      printf("Enter the coordinates of the second corner: "); scanf("%d%d", &x2, &y2);
(gdb) next
Enter the coordinates of the second corner: 6 -7
24      printf("Enter the coordinates of the third corner: "); scanf("%d%d", &x3, &y3);
(gdb) next
Enter the coordinates of the third corner: -1 5
25      area = tarea(x1,y1,x2,y2,x3,y3);
(gdb) next
26      printf("Area of the triangle is %lf\n", area);
(gdb) next
Area of the triangle is 16.500000
27      exit(0);
(gdb)
```

Step into the function

```
...
(gdb) next
Enter the coordinates of the third corner: -1 5
25     area = tarea(x1,y1,x2,y2,x3,y3);
(gdb) step
tarea (x1=32767, y1=-7504, x2=21845, y2=1431655248, x3=32767, y3=-136046822) at TAREA.c:5
5     {
(gdb) next
8     area = x1 * (y2 - y3);
(gdb) print area
$1 = 6.953355807476115e-310
(gdb) next
9     area += x2 * (y1 - y3);
(gdb) print area
$2 = -36
(gdb) next
10    area += x3 * (y1 - y2);
(gdb) print area
$3 = -18
(gdb) finish
Run till exit from #0 tarea (x1=3, y1=8, x2=6, y2=-7, x3=-1, y3=5) at TAREA.c:10
0x000055555555531c in main () at TAREA.c:25
25     area = tarea(x1,y1,x2,y2,x3,y3);
Value returned is $4 = 16.5
(gdb) n
26     printf("Area of the triangle is %lf\n", area);
(gdb) n
Area of the triangle is 16.500000
27     exit(0);
(gdb)
```

Forced return from a function

```
...
(gdb) next
Enter the coordinates of the third corner: -1 5
25     area = tarea(x1,y1,x2,y2,x3,y3);
(gdb) step
tarea (x1=32767, y1=-7504, x2=21845, y2=1431655248, x3=32767, y3=-136046822) at TAREA.c:5
5     {
(gdb) next
8     area = x1 * (y2 - y3);
(gdb) print area
$1 = 6.953355807476115e-310
(gdb) next
9     area += x2 * (y1 - y3);
(gdb) print area
$2 = -36
(gdb) next
10    area += x3 * (y1 - y2);
(gdb) print area
$3 = -18
(gdb) return
Make tarea return now? (y or n) y
#0 0x000055555555531c in main () at TAREA.c:25
25     area = tarea(x1,y1,x2,y2,x3,y3);
(gdb) n
26     printf("Area of the triangle is %lf\n", area);
(gdb) n
Area of the triangle is -18.000000
27     exit(0);
(gdb)
```

Program with a recursive function

fib.c

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

int Fib ( int n )
{
    if (n < 0) return -1;
    if (n <= 1) return n;
    return Fib(n-1) + Fib(n-2);
}

int main ()
{
    int n, f;

    n = 8;
    f = Fib(n);
    printf("F(%d) = %d\n", n, f);
    exit(0);
}
```

Program with a recursive function: Line numbers

```
(gdb) 1
1      #include <stdio.h>
2      #include <stdlib.h>
3
4      int Fib ( int n )
5      {
6          if ( n < 0 ) return -1;
7          if ( n <= 1 ) return n;
8          return Fib(n-1) + Fib(n-2);
9      }
10
(gdb) 1
11     int main ()
12     {
13         int n, f;
14
15         n = 8;
16         f = Fib(n);
17         printf("F(%d) = %d\n", n, f);
18         exit(0);
19     }
(gdb)
```


Function call with next

```
(gdb) break main
Breakpoint 1 at 0x11b6: file fib.c, line 12.
(gdb) run
Starting program: /home/abhiij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out

Breakpoint 1, main () at fib.c:12
12      {
(gdb) n
15      n = 8;
(gdb) n
16      f = Fib(n);
(gdb) n
17      printf("F(%d) = %d\n", n, f);
(gdb) n
F(8) = 21
18      exit(0);
(gdb) n
[Inferior 1 (process 12316) exited normally]
(gdb) q
```

Function call with only one step

```
(gdb) break main
Breakpoint 1 at 0x11b6: file fib.c, line 12.
(gdb) run
Starting program: /home/abhiij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out

Breakpoint 1, main () at fib.c:12
12      {
(gdb) n
15      n = 8;
(gdb) n
16      f = Fib(n);
(gdb) step
Fib (n=21845) at fib.c:5
5      {
(gdb) n
6      if (n < 0) return -1;
(gdb) n
7      if (n <= 1) return n;
(gdb) n
8      return Fib(n-1) + Fib(n-2);
(gdb) n
9      }
(gdb) n
main () at fib.c:17
17      printf("F(%d) = %d\n", n, f);
(gdb) n
F(8) = 21
18      exit(0);
(gdb)
```

Enter recursive calls with step

```
Breakpoint 1, main () at fib.c:12
12      {
(gdb) n
15      n = 8;
(gdb) n
16      f = Fib(n);
(gdb) s
Fib (n=21845) at fib.c:5
5      {
(gdb) n
6      if (n < 0) return -1;
(gdb) n
7      if (n <= 1) return n;
(gdb) n
8      return Fib(n-1) + Fib(n-2);
(gdb) s
Fib (n=0) at fib.c:5
5      {
(gdb) n
6      if (n < 0) return -1;
(gdb) n
7      if (n <= 1) return n;
(gdb) n
8      return Fib(n-1) + Fib(n-2);
(gdb) s
Fib (n=0) at fib.c:5
5      {
(gdb)
```

Set breakpoint with function name

```
(gdb) break main
Breakpoint 1 at 0x11b6: file fib.c, line 12.
(gdb) break Fib
Breakpoint 2 at 0x1169: file fib.c, line 5.
(gdb) run
Starting program: /home/abhij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out

Breakpoint 1, main () at fib.c:12
12      {
(gdb) n
15      n = 8;
(gdb) n
16      f = Fib(n);
(gdb) n

Breakpoint 2, Fib (n=21845) at fib.c:5
5      {
(gdb) n
6      if (n < 0) return -1;
(gdb) n
7      if (n <= 1) return n;
(gdb) n
8      return Fib(n-1) + Fib(n-2);
(gdb) n

Breakpoint 2, Fib (n=0) at fib.c:5
5      {
(gdb)
```

Stepping into an external function

Stepping in average.c

```
(gdb) break main
Breakpoint 1 at 0x11a9: file average.c, line 5.
(gdb) run
Starting program: /home/abhij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out

Breakpoint 1, main () at average.c:5
5      {
(gdb) s
8          n = sum = 0;
(gdb) s
9      printf("Keep on entering non-negative integers. Enter a negative integer to end.\n");
(gdb) s
__GI__IO_puts (
    str=0x555555556008 "Keep on entering non-negative integers. Enter a negative integer to end."
    at ioputs.c:33
33      ioputs.c: No such file or directory.
(gdb) s
35      in ioputs.c
(gdb) s
36      in ioputs.c
(gdb) s
__lll_cas_lock (futex=0x7ffff7fa74c0 <_IO_stdfile_1_lock>)
    at ../sysdeps/unix/sysv/linux/x86/lowlevellock.h:47
47      ../sysdeps/unix/sysv/linux/x86/lowlevellock.h: No such file or directory.
(gdb)
```

The call stack

- A function call defines a frame.
- Commands to work with frames:
 - frame: List information about the current frame
 - info frame: List more detailed information about the current frame
 - backtrace: Print the entire stack from top to bottom
 - up: Move one step up in the stack (toward the stack bottom, up in the call tree)
 - down: Move one step down in the stack (toward the stack top, down in the call tree)
- Note: up/down does not change the execution to go up or down. You only move inside the stack, and can see the details of the frames in the stack.

Program to demonstrate the call stack

callstack.c

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

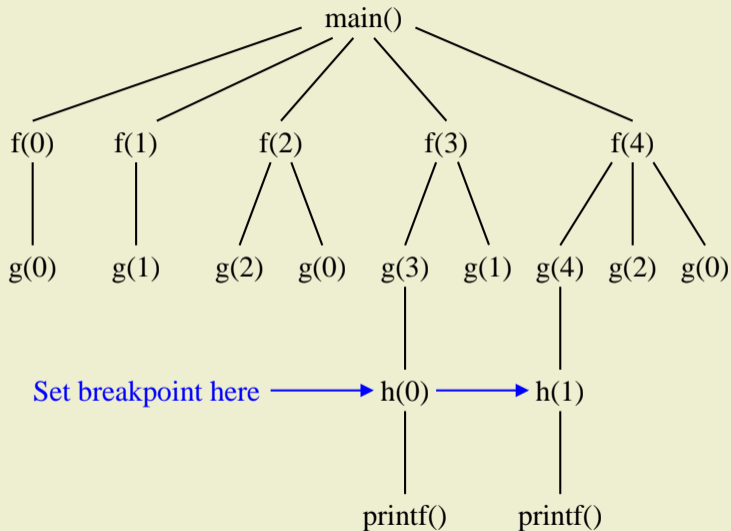
void h ( int n )
{
    printf("+++ Called h(%d)\n", n);
}

void g (int n )
{
    if (n >= 3) h(n-3);
}

void f ( int n )
{
    while (n >= 0) { g(n); n -= 2; }
}

int main ()
{
    int i;
    for (i=0; i<5; ++i) f(i);
    exit(0);
}
```

The call tree



View the call stack

```
(gdb) break h
Breakpoint 1 at 0x1169: file callstack.c, line 5.
(gdb) run
Starting program: /home/abhij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out

Breakpoint 1, h (n=0) at callstack.c:5
5      {
(gdb) backtrace
#0  h (n=0) at callstack.c:5
#1  0x0000555555551b3 in g (n=3) at callstack.c:11
#2  0x0000555555551d1 in f (n=3) at callstack.c:16
#3  0x0000555555551fe in main () at callstack.c:23
(gdb) next
6          printf("+++ Called h(%d)\n", n);
(gdb) step
__printf (format=0x55555556004 "+++ Called h(%d)\n") at printf.c:28
28      printf.c: No such file or directory.
(gdb) backtrace
#0  __printf (format=0x55555556004 "+++ Called h(%d)\n") at printf.c:28
#1  0x00005555555518e in h (n=0) at callstack.c:6
#2  0x0000555555551b3 in g (n=3) at callstack.c:11
#3  0x0000555555551d1 in f (n=3) at callstack.c:16
#4  0x0000555555551fe in main () at callstack.c:23
(gdb) continue
Continuing.
+++ Called h(0)

Breakpoint 1, h (n=1) at callstack.c:5
5      {
(gdb) backtrace
```

View the call stack (continued)

```
#0 h (n=1) at callstack.c:5
#1 0x000055555555551b3 in g (n=4) at callstack.c:11
#2 0x000055555555551d1 in f (n=4) at callstack.c:16
#3 0x000055555555551fe in main () at callstack.c:23
(gdb) up
#1 0x000055555555551b3 in g (n=4) at callstack.c:11
11     if (n >= 3) h(n-3);
(gdb) up
#2 0x000055555555551d1 in f (n=4) at callstack.c:16
16     while (n >= 0) { g(n); n -= 2; }
(gdb) info frame
Stack level 2, frame at 0x7fffffff2a0:
 rip = 0x555555551d1 in f (callstack.c:16); saved rip = 0x555555551fe
 called by frame at 0x7fffffff2c0, caller of frame at 0x7fffffff280
 source language c.
 Arglist at 0x7fffffff278, args: n=4
 Locals at 0x7fffffff278, Previous frame's sp is 0x7fffffff2a0
 Saved registers:
  rbp at 0x7fffffff290, rip at 0x7fffffff298
(gdb) up
#3 0x000055555555551fe in main () at callstack.c:23
23     for (i=0; i<5; ++i) f(i);
(gdb) up
Initial frame selected; you cannot go up.
(gdb) down 2
#1 0x000055555555551b3 in g (n=4) at callstack.c:11
11     if (n >= 3) h(n-3);
(gdb) continue
Continuing.
+++ Called h(1)
```

Call stack in recursive Fibonacci program

```
(gdb) break Fib
Breakpoint 1 at 0x1169: file fib.c, line 5.
(gdb) run
Starting program: /home/abhij/IITKGP/course/lab/SPL/Spring22/prog/gdb/a.out

Breakpoint 1, Fib (n=21845) at fib.c:5
5      {
(gdb) continue 7
Will ignore next 6 crossings of breakpoint 1. Continuing.

Breakpoint 1, Fib (n=0) at fib.c:5
5      {
(gdb) bt
#0 Fib (n=0) at fib.c:5
#1 0x00005555555519e in Fib (n=2) at fib.c:8
#2 0x00005555555519e in Fib (n=3) at fib.c:8
#3 0x00005555555519e in Fib (n=4) at fib.c:8
#4 0x00005555555519e in Fib (n=5) at fib.c:8
#5 0x00005555555519e in Fib (n=6) at fib.c:8
#6 0x00005555555519e in Fib (n=7) at fib.c:8
#7 0x00005555555519e in Fib (n=8) at fib.c:8
#8 0x0000555555551d3 in main () at fib.c:16
(gdb) c
Continuing.

Breakpoint 1, Fib (n=1) at fib.c:5
5      {
```

Call stack in recursive Fibonacci program (continued)

```
(gdb) bt
#0 Fib (n=1) at fib.c:5
#1 0x000055555555551ad in Fib (n=2) at fib.c:8
#2 0x0000555555555519e in Fib (n=3) at fib.c:8
#3 0x0000555555555519e in Fib (n=4) at fib.c:8
#4 0x0000555555555519e in Fib (n=5) at fib.c:8
#5 0x0000555555555519e in Fib (n=6) at fib.c:8
#6 0x0000555555555519e in Fib (n=7) at fib.c:8
#7 0x0000555555555519e in Fib (n=8) at fib.c:8
#8 0x000055555555551d3 in main () at fib.c:16
(gdb) c
Continuing.

Breakpoint 1, Fib (n=2) at fib.c:5
5 {
(gdb) bt
#0 Fib (n=2) at fib.c:5
#1 0x000055555555551ad in Fib (n=3) at fib.c:8
#2 0x0000555555555519e in Fib (n=4) at fib.c:8
#3 0x0000555555555519e in Fib (n=5) at fib.c:8
#4 0x0000555555555519e in Fib (n=6) at fib.c:8
#5 0x0000555555555519e in Fib (n=7) at fib.c:8
#6 0x0000555555555519e in Fib (n=8) at fib.c:8
#7 0x000055555555551d3 in main () at fib.c:16
(gdb)
```

If you are lost/inquisitive, ask for 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 stopping the program.  
user-defined - User-defined commands.
```

```
Type "help" followed by a class name for a list of commands in that class.
```

```
Type "help all" for the list of all commands.
```

```
Type "help" followed by command name for full documentation.
```

```
Type "apropos word" to search for commands related to "word".
```

```
Type "apropos -v word" for full documentation of commands related to "word".
```

```
Command name abbreviations are allowed if unambiguous.
```

```
(gdb) help step
```

```
Step program until it reaches a different source line.
```

```
Usage: step [N]
```

```
Argument N means step N times (or till program stops for another reason).
```

```
(gdb) help set var
```

```
Evaluate expression EXP and assign result to variable VAR.
```

```
Usage: set variable VAR = EXP
```

```
...
```