

GDB stands for GNU Project Debugger and is a powerful debugging tool for C(with other languages like C++). It helps you poke around inside your C programs while they are executing and allows you to see what happens when your program crashes. GDB operates on executable files, which are binary files produced by the compilation process.

The below example is executed on a Linux machine with the below specs for demo purposes.

```
sree@ubuntu:~$ uname -a
Linux ubuntu 4.10.0-19-generic #21-Ubuntu SMP Thu Apr 6 17:04:57 UTC 2017 x86_64
x86_64 x86_64 GNU/Linux
```

Let's learn by doing:-

1. Go to your Linux command prompt and type "gdb."

```
sree@ubuntu:~$ gdb
GNU gdb (Ubuntu 7.12.50.20170314-0ubuntu1) 7.12.50.20170314-git
Copyright (C) 2017 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".
(gdb) |
```

Gdb open prompt lets you know that it is ready for commands. To exit out of gdb, type quit or q.

```
(gdb) quit
sree@ubuntu:~$ |
```

2. Below is a program that shows undefined behavior when compiled using C99.

```
sree@ubuntu:~/debugging$ cat test.c
//sample program to show undefined behaviour
//Author : Sreeraj R
#include<stdio.h>

int main()
{
    int x;
    int a = x;
    int b = x;
    int c = a + b;
    printf("%d\n", c);
    return 0;
}
```

**Note:** If an object with automatic storage duration is not initialized explicitly, its value is indeterminate, where the indeterminate value is either an unspecified value or a trap representation.

3. Now, compile the code. (here test.c).

**g flag** means you can see the proper names of variables and functions in your stack frames, get line numbers and see the source as you step around in the executable.

**-std=C99 flag** implies the use of standard C99 to compile the code.

**-o flag** writes the build output to an output file.

```
sree@ubuntu:~/debugging$ gcc -std=c99 -g -o test test.c
sree@ubuntu:~/debugging$ ls -lart
total 24
drwxr-xr-x 25 sree sree  4096 Oct 20 10:07 ..
-rw-r--r--  1 sree sree   197 Oct 20 11:13 test.c
drwxr-xr-x  2 sree sree  4096 Oct 20 11:14 .
-rwxr-xr-x  1 sree sree 10840 Oct 20 11:14 test
sree@ubuntu:~/debugging$
```

4. Run gdb with the generated executable.

```
sree@ubuntu:~/debugging$ gdb ./test
GNU gdb (Ubuntu 7.12.50.20170314-0ubuntu1) 7.12.50.20170314-git
Copyright (C) 2017 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 ./test...done.
(gdb) r
Starting program: /home/sree/debugging/test
-15808
[Inferior 1 (process 4989) exited normally]
(gdb) |
```

Here are a few useful commands to get started with gdb for the above example:-

run or r → executes the program from start to end.

break or b → sets a breakpoint on a particular line.

disable → disable a breakpoint.

enable → enable a disabled breakpoint.

next or n → executes the next line of code, but don't dive into functions.

step → go to the next instruction, diving into the function.

list or l → displays the code.

print or p → used to display the stored value.

quit or q → exits out of gdb.

clear → to clear all breakpoints.

continue → continue normal execution.

5. Now, type "l" at the gdb prompt to display the code.

```
(gdb) l
1      //sample program to show undefined behaviour
2      //Author : Sreeraj R
3      #include<stdio.h>
4
5      int main()
6      {
7          int x;
8          int a = x;
9          int b = x;
10         int c = a + b;
(gdb)
```

6. Let's introduce a break point, say line 5.

```
(gdb) b 5
Breakpoint 1 at 0x5555555546a8: file test.c, line 5.
(gdb) |
```

If you want to put a breakpoint at different lines, you can type "b *line\_number*". "By default, "list or l" displays only the first ten lines.

7. In order to see the breakpoints, type "info b."

```
(gdb) info b
Num      Type           Disp Enb Address                What
1        breakpoint     keep y   0x000000000000006a8 in main at test.c:5
(gdb) |
```

8. Having done the above, let's say you changed your mind and you want to revert.

Type "disable b."

```
(gdb) disable b
(gdb) info b
Num      Type           Disp Enb Address                What
1        breakpoint     keep n   0x000000000000006a8 in main at test.c:5
(gdb)
```

As marked in the blue circle, Enb becomes n for disabled.

9. To re-enable the recently disabled breakpoint. Type "enable b."

```
(gdb) enable b
(gdb) info b
Num      Type           Disp Enb Address                What
1        breakpoint     keep y   0x000000000000006a8 in main at test.c:5
(gdb)
```

10. Run the code by typing "run or r." If you haven't set any breakpoints, the run command will execute the entire program.

```
(gdb) r
Starting program: /home/sree/debugging/test

Breakpoint 1, main () at test.c:8
8      int a = x;
(gdb) |
```

11. To see the value of a variable, type “print *variable\_name* or p *variable\_name*. “

```
(gdb) p x
$1 = -7904
(gdb) |
```

The above shows the values stored at x at the time of execution.

12. To change the value of the variable in gdb and continue execution with the changed value, type “set *variable\_name*. “

13. Below screenshot shows the values of variables from which it's quite understandable the reason why we got a garbage value as output. At every execution of **.test**, we will receive different outputs.

**Exercise: Try using set x = 0 in gdb at the first run and see the output of c.**

```

(gdb) r
Starting program: /home/sree/debugging/test

Breakpoint 1, main () at test.c:8
8           int a = x;
(gdb) p x
$1 = -7904
(gdb) p a
$2 = 32767
(gdb) n
9           int b = x;
(gdb) p x
$3 = -7904
(gdb) p a
$4 = -7904
(gdb) p b
$5 = 0
(gdb) n
10          int c = a + b;
(gdb) p x
$6 = -7904
(gdb) p a
$7 = -7904
(gdb) p b
$8 = -7904
(gdb) p c
$9 = 0
(gdb) n
11          printf("%d\n", c);
(gdb) p x
$10 = -7904
(gdb) p a
$11 = -7904
(gdb) p b
$12 = -7904
(gdb) p c
$13 = -15808
(gdb) n
-15808

```

GDB offers many more ways to debug and understand your code, like examining stack, memory, threads, manipulating the program, etc. I hope the above example helps you get started with gdb.

### **[gdb command in Linux with examples](#)**

1. **run [args]** : This command runs the current executable file. In the below image, the program was executed twice, one with the command line argument 10 and another with

the command line argument 1, and their corresponding outputs were printed.

```
lokesh@lokesh-pc:~/sampleCodes/c++Files$ g++ -g -o gfg gfg.cpp
lokesh@lokesh-pc:~/sampleCodes/c++Files$ gdb gfg
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 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 gfg...done.
(gdb) run 10
Starting program: /home/lokesh/sampleCodes/c++Files/gfg 10
100
[Inferior 1 (process 7975) exited normally]
(gdb) run 1
Starting program: /home/lokesh/sampleCodes/c++Files/gfg 1
1
[Inferior 1 (process 7979) exited normally]
(gdb) □
```

2. **quit or q** : To quit the gdb console, either **quit** or **q** can be used.
3. **help** : It launches the manual of gdb along with all list of classes of individual commands.
4. **break** : The command **break [function name]** helps to pause the program during execution when it starts to execute the function. It helps to debug the program at that point. Multiple breakpoints can be inserted by executing the command wherever necessary. **b findSquare** command makes the gfg executable pause when the debugger starts to execute the findSquare function.

**b**

**break [function name]**

**break [file name]:[line number]**

**break [line number]**

**break \*[address]**

**break \*\*\*any of the above arguments\*\*\* if [condition]**

**b \*\*\*any of the above arguments\*\*\***

```

Lokesh@Lokesh-pc:~/sampleCodes/c++Files$ gdb gfg
GNU gdb (Ubuntu 8.1.0-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 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 gfg...done.
(gdb) b findSquare
Breakpoint 1 at 0x8e1: file gfg.cpp, line 7.
(gdb) run 10 100
Starting program: /home/lokesh/sampleCodes/c++Files/gfg 10 100

Breakpoint 1, findSquare (a=10) at gfg.cpp:7
7          return a*a;
(gdb) █

```


In the above example, the program that was being executed(**run 10 100**), paused when it encountered findSquare function call. The program pauses whenever the function is called. Once the command is successful, it prints the breakpoint number, information of the program counter, file name, and the line number. As it encounters any breakpoint during execution, it prints the breakpoint number, function name with the values of the arguments, file name, and line number. The breakpoint can be set either with the address of the instruction(in hexadecimal form preceded with \*0x) or the line number and it can be combined with if condition(if the condition fails, the breakpoint will not be set) For example, **break findSquare if a == 10**.

5. **continue** : This command helps to resume the current executable after it is paused by the breakpoint. It executes the program until it encounters any breakpoint or runs time error or the end of the program. If there is an integer in the argument(repeat count), it will consider it as the continue repeat count and will execute continue command "repeat count" number of times.

**continue [repeat count]**

**c [repeat count]**





```
lokes@lokes-pc:~/sampleCodes/c++Files$ gdb gfg
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 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 gfg...done.
(gdb) b findSquare
Breakpoint 1 at 0x8e1: file gfg.cpp, line 7.
(gdb) run 10 10 10
Starting program: /home/lokes/sampleCodes/c++Files/gfg 10 10 10

Breakpoint 1, findSquare (a=10) at gfg.cpp:7
7           return a*a;
(gdb) c
Continuing.
100

Breakpoint 1, findSquare (a=10) at gfg.cpp:7
7           return a*a;
(gdb) c
Continuing.
100

Breakpoint 1, findSquare (a=10) at gfg.cpp:7
7           return a*a;
(gdb) c
Continuing.
100
[Inferior 1 (process 8182) exited normally]
(gdb)
```

6. **next or n** : This command helps to execute the next instruction after it encounters the breakpoint.

```

(gdb) b findSquare
Breakpoint 1 at 0x8e1: file gfg.cpp, line 7.
(gdb) run 1 10 100
Starting program: /home/lokesk/sampleCodes/c++Files/gfg 1 10 100

Breakpoint 1, findSquare (a=1) at gfg.cpp:7
7           return a*a;
(gdb) n
8       }
(gdb) next
1
main (n=4, args=0x7fffffffde38) at gfg.cpp:11
11          for(int i=1;i<n;i++){
(gdb) n
12              int a=atoi(args[i]);
(gdb) n
13              cout<<findSquare(a)<<endl;
(gdb) next

Breakpoint 1, findSquare (a=10) at gfg.cpp:7
7           return a*a;
(gdb) next
8       }
(gdb) n
100
main (n=4, args=0x7fffffffde38) at gfg.cpp:11
11          for(int i=1;i<n;i++){
(gdb) n
12              int a=atoi(args[i]);
(gdb) n
13              cout<<findSquare(a)<<endl;
(gdb) n

```

Whenever it encounters the above command, it executes the next instruction of the executable by printing the line in execution.

7. **delete** : This command helps to delete the breakpoints and checkpoints. If the delete command is executed without any arguments, it deletes all the breakpoints without modifying any of the checkpoints. Similarly, if the checkpoint of the parent process is deleted, all the child checkpoints are automatically deleted.

**d**

**delete**

**delete [breakpoint number 1] [breakpoint number 2] ...**

**delete checkpoint [checkpoint number 1] [checkpoint number 2] ...**

```

Reading symbols from gfg...done.
(gdb) b main
Breakpoint 1 at 0x8f9: file gfg.cpp, line 11.
(gdb) b findSquare
Breakpoint 2 at 0x8e1: file gfg.cpp, line 7.
(gdb) d 2
(gdb) run 10
Starting program: /home/lokesesh/sampleCodes/c++Files/gfg 10

Breakpoint 1, main (n=2, args=0x7fffffffde48) at gfg.cpp:11
11         for(int i=1;i<n;i++){
(gdb) c
Continuing.
100
[Inferior 1 (process 11836) exited normally]
(gdb) 

```

In the above example, two breakpoints were defined, one at the main and the other at the findSquare. Using the above command findSquare breakpoint was deleted. If there is no argument after the command, the command deletes all the breakpoints.

8. **clear** : This command deletes the breakpoint which is at a particular function with the name FUNCTION\_NAME. If the argument is a number, then it deletes the breakpoint that lies in that particular line.

**clear [line number]**

**clear [FUNCTION\_NAME]**

```

(gdb) b findSquare
Breakpoint 1 at 0x8e1: file gfg.cpp, line 7.
(gdb) b main
Breakpoint 2 at 0x8f9: file gfg.cpp, line 11.
(gdb) clear main
Deleted breakpoint 2
(gdb) run 1
Starting program: /home/lokesesh/sampleCodes/c++Files/gfg 1

Breakpoint 1, findSquare (a=1) at gfg.cpp:7
7         return a*a;
(gdb) c
Continuing.
1
[Inferior 1 (process 20915) exited normally]
(gdb) 

```

In the above example, once the clear command is executed, the breakpoint is deleted after printing the breakpoint number.

9. **disable [breakpoint number 1] [breakpoint number 2] ....** : Instead of deleting or clearing the breakpoints, they can be disabled and can be enabled whenever they are necessary.
10. **enable [breakpoint number 1] [breakpoint number 2] ....** : To enable the disabled breakpoints, this command is used.
11. **info** : When the info breakpoints is invoked, the breakpoint number, type, display, status, address, the location will be displayed. If the breakpoint number is specified, only the information about that particular breakpoint will be displayed. Similarly, when the info checkpoints are invoked, the checkpoint number, the process id, program counter, file name, and line number are displayed.  
info breakpoints [breakpoint number 1] [breakpoint number 2] ...  
info checkpoints [checkpoint number 1] [checkpoint number 2] ...

```
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) break 1
Breakpoint 3 at 0x8e1: file gfg.cpp, line 1.
(gdb) break 2
Note: breakpoint 3 also set at pc 0x8e1.
Breakpoint 4 at 0x8e1: file gfg.cpp, line 2.
(gdb) break 3
Note: breakpoints 3 and 4 also set at pc 0x8e1.
Breakpoint 5 at 0x8e1: file gfg.cpp, line 3.
(gdb) info breakpoints
Num      Type           Disp Enb Address            What
3        breakpoint      keep y   0x000000000000008e1 in findSquare(int) at gfg.cpp:1
4        breakpoint      keep y   0x000000000000008e1 in findSquare(int) at gfg.cpp:2
5        breakpoint      keep y   0x000000000000008e1 in findSquare(int) at gfg.cpp:3
```

12. **checkpoint command and restart command** : These command creates a new process and keep that process in the suspended mode and prints the created process's process id.

```

(gdb) b findSquare
Breakpoint 1 at 0x8e1: file gfg.cpp, line 7.
(gdb) run 1 10 100
Starting program: /home/lokesk/sampleCodes/c++Files/gfg 1 10 100

Breakpoint 1, findSquare (a=1) at gfg.cpp:7
7         return a*a;
(gdb) checkpoint
checkpoint 1: fork returned pid 4272.
(gdb) continue
Continuing.
1

Breakpoint 1, findSquare (a=10) at gfg.cpp:7
7         return a*a;
(gdb) checkpoint
checkpoint 2: fork returned pid 4278.
(gdb) info checkpoints
* 0 process 4268 (main process) at 0x5555555548e1, file gfg.cpp, line 7
  1 process 4272 at 0x5555555548e1, file gfg.cpp, line 7
  2 process 4278 at 0x5555555548e1, file gfg.cpp, line 7
(gdb) restart 1
Switching to process 4272
#0 findSquare (a=1) at gfg.cpp:7
7         return a*a;
(gdb) info checkpoints
  0 process 4268 (main process) at 0x5555555548e1, file gfg.cpp, line 7
* 1 process 4272 at 0x5555555548e1, file gfg.cpp, line 7
  2 process 4278 at 0x5555555548e1, file gfg.cpp, line 7
(gdb) restart 0
Switching to process 4268
#0 findSquare (a=10) at gfg.cpp:7
7         return a*a;
(gdb) c
Continuing.
100

```

**For example**, in the above execution, the breakpoint is kept at function *findSquare* and the program was executed with the arguments “1 10 100”. When the function is called initially with  $a = 1$ , the breakpoint happens. Now we create a checkpoint and hence gdb returns a process id(4272), keeps it in the suspended mode and resumes the original thread once the continue command is invoked. Now the breakpoint happens with  $a = 10$  and another checkpoint(pid = 4278) is created. From the info checkpoint information, the asterisk mentions the process that will run if the gdb encounters a continue. To resume a specific process, **restart** command is used with the argument that specifies the serial

number of the process. If all the process are finished executing, the **info checkpoint** command returns nothing.

13. **set args [arg1] [arg2] ...** : This command creates the argument list and it passes the specified arguments as the command line arguments whenever the **run** command without any argument is invoked. If the **run** command is executed with arguments after **set args**, the arguments are updated. Whenever the **run** command is ran without the arguments, the arguments are set by default.

```
(gdb) set args 1 10
(gdb) run
Starting program: /home/lokesh/sampleCodes/c++Files/gfg 1 10
1
100
[Inferior 1 (process 4712) exited normally]
```

14. **show args** : The show args prints the default arguments that will passed if the **run** command is executed. If either **set args** or **run** command is executed with the arguments, the default arguments will get updated, and can be viewed using the above **show args** command.

```
(gdb) set args 1 10
(gdb) run
Starting program: /home/lokesh/sampleCodes/c++Files/gfg 1 10
1
100
[Inferior 1 (process 4712) exited normally]
(gdb) show args
Argument list to give program being debugged when it is started is "1 10".
(gdb) █
```

15. **display [/format specifier] [expression] and undisplay [display id1] [display id2] ...** : These command enables automatic displaying of expressions each time whenever the execution encounters a breakpoint or the **n** command. The **undisplay** command is used to remove display expressions. Valid format specifiers are as follows:

*o - octal*

*x - hexadecimal*

*d - decimal*

*u - unsigned decimal*

*t - binary*

*f - floating point*

*a - address*

*c - char*

*s - string*

*i - instruction*

```
(gdb) b 12
Breakpoint 2 at 0x908: file gfg.cpp, line 12.
(gdb) run 1 10 100
Starting program: /home/lokesh/sampleCodes/c++Files/gfg 1 10 100

Breakpoint 2, main (n=4, args=0x7fffffffde18) at gfg.cpp:12
12             int a=atoi(args[i]);
(gdb) display /x i
1: /x i = 0x1
(gdb) display /s args[i]
2: x/s args[i]  0x7fffffffde19a: "1"
(gdb) c
Continuing.
1

Breakpoint 2, main (n=4, args=0x7fffffffde18) at gfg.cpp:12
12             int a=atoi(args[i]);
1: /x i = 0x2
2: x/s args[i]  0x7fffffffde19c: "10"
(gdb) undisplay 1
(gdb) c
Continuing.
100

Breakpoint 2, main (n=4, args=0x7fffffffde18) at gfg.cpp:12
12             int a=atoi(args[i]);
2: x/s args[i]  0x7fffffffde19f: "100"
(gdb) n
13             cout<<findSquare(a)<<endl;
2: x/s args[i]  0x7fffffffde19f: "100"
(gdb) n
10000
11             for(int i=1;i<n;i++){
2: x/s args[i]  0x7fffffffde19f: "100"
(gdb) c
Continuing.
[Inferior 1 (process 5868) exited normally]
(gdb) □
```

In the above example, the breakpoint is set at line 12 and ran with the arguments 1 10 100. Once the breakpoint is encountered, display command is executed to print the value of **i** in hexadecimal form and value of **args[i]** in the string form. After then, whenever the command **n** or a breakpoint is encountered, the values are displayed again until they are disabled using **undisplay** command.

16. **print** : This command prints the value of a given expression. The display command prints all the previously displayed values whenever it encounters a breakpoint or the next command, whereas the print command saves all the previously displayed values and prints whenever it is called.

**print [Expression]**

**print \$[Previous value number]**



**`print {[Type]}[Address]`**  
**`print [First element]@[Element count]`**  
**`print /[Format] [Expression]`**

```
(gdb) set args 1 10 100
(gdb) start
Temporary breakpoint 1 at 0x8f9: file gfg.cpp, line 11.
Starting program: /home/lokesk/sampleCodes/c++Files/gfg 1 10 100

Temporary breakpoint 1, main (n=4, args=0x7fffffffde18) at gfg.cpp:11
11      for(int i=1;i<n;i++){
(gdb) print /x i
$1 = 0x0
(gdb) n
12      int a=atoi(args[i]);
(gdb) print /x i
$2 = 0x1
(gdb) print /s args
$3 = (char **) 0x7fffffffde18
(gdb) print /s *args
$4 = 0x7fffffff174 "/home/lokesk/sampleCodes/c++Files/gfg"
(gdb) print /s *args@n
$5 = {0x7fffffff174 "/home/lokesk/sampleCodes/c++Files/gfg", 0x7fffffff19a "1", 0x7fffffff19c "10", 0x7fffffff19f "100"}
(gdb) print $4
$6 = 0x7fffffff174 "/home/lokesk/sampleCodes/c++Files/gfg"
(gdb) continue
Continuing.
1
100
10000
[Inferior 1 (process 6142) exited normally]
(gdb) □
```

17. **file** : gdb console can be opened using the command **`gdb`** command. To debug the executables from the console, **`file [executable filename]`** command is used.

```
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 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".
(gdb) file gfg
Reading symbols from gfg...done.
(gdb) run
Starting program: /home/lokesk/sampleCodes/c++Files/gfg
[Inferior 1 (process 6249) exited normally]
(gdb) run 1 10 100
Starting program: /home/lokesk/sampleCodes/c++Files/gfg 1 10 100
1
100
10000
[Inferior 1 (process 6253) exited normally]
(gdb) □
```