GDB Demonstration Document

*Note : 46255542\_Smit Bhamwala*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This Code uses two functions to find the sum and product.

Sum finds the sum of two numbers.

Product finds the product of two numbers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h> int sum(int, int);

int product(int, int);

int main() {

int x,y; x=5;

y=20;

printf("The Sum is %d\n",sum(x,y));

printf("The Product is %d\n",product(x,y));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Sum finds the sum of two integers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int sum(int a , int b)

{

int s; s=a+b; return(s);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Product finds the product of two integers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int product(int a, int b)

{

int p; p=a\*b; return(p);

}

**Step 1**: Write down the above code using vi editor.

**Step 2**: Save and Quit.

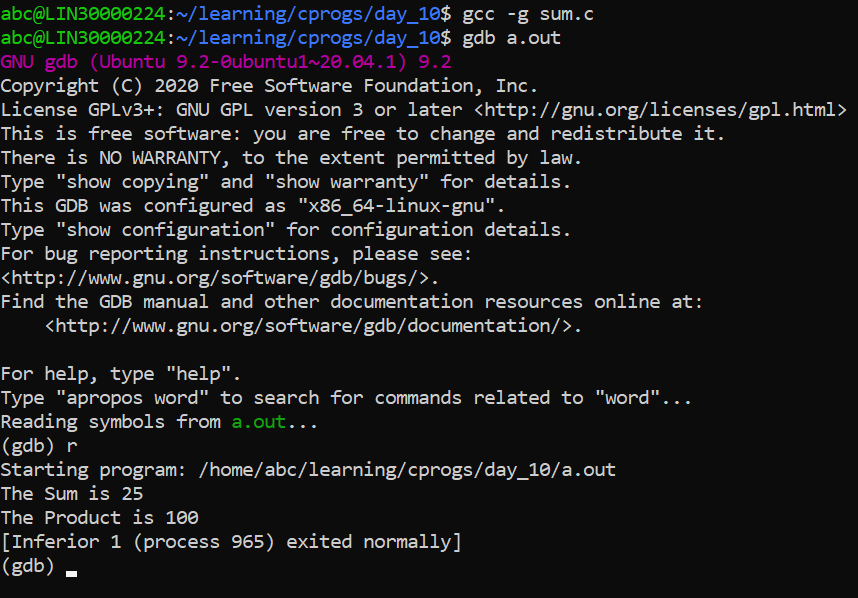
**Step 3**: Compile the code using gcc with the option –g

*gcc -g filename.c*

( The –g option with gcc, collects the symbol table information which shall be used by gdb)

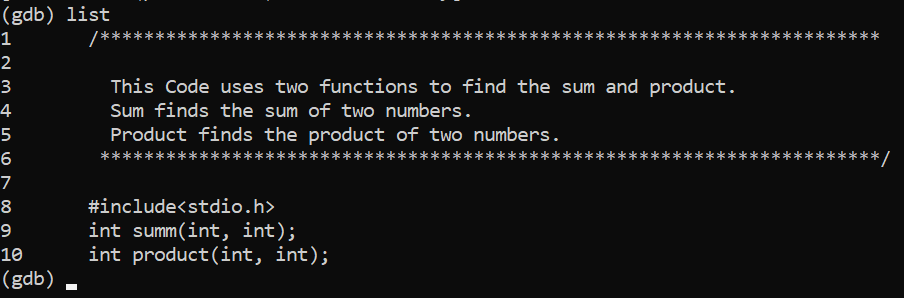
**Step 4**: Execute the executable with gdb

*gdb a.out*



**Step 5**: List the code

(gdb) *list*



What do you observe ?

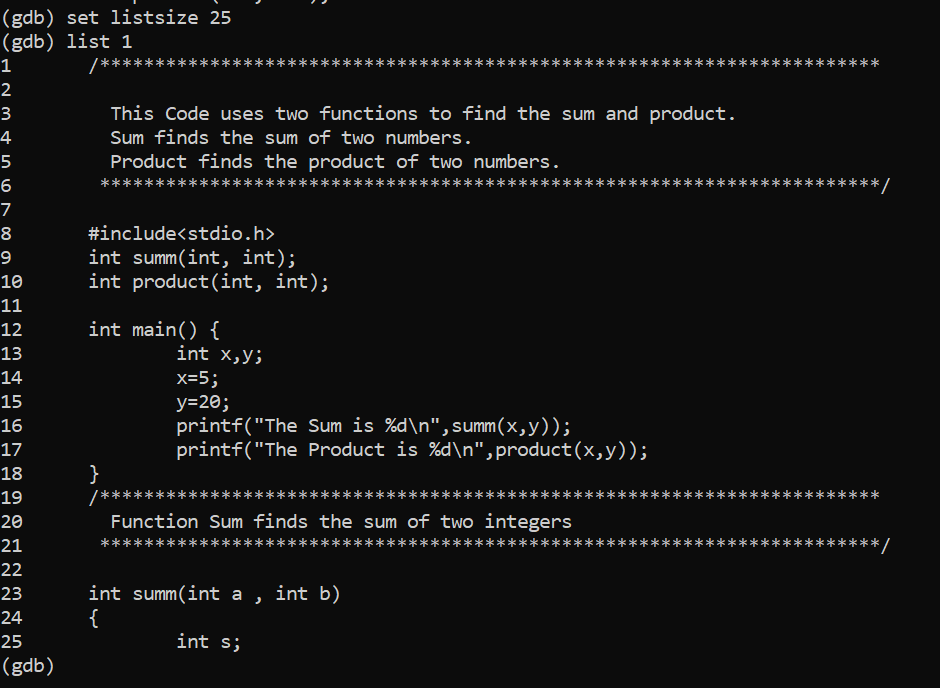
* It shows 10 lines from code in text format.

**Step 6**: If you want to list more than 10 lines, set the list size.

(gdb) *set listsize 25*

(gdb) *list*

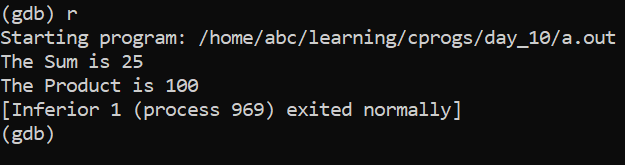
What do you observe? ( Type *list 1* : Observe the output)



* **Now its showing entire code from 1 to line 25.**

**Step 7**: Run the code.

(gdb) *r*



What do you observe ?

* Runs the program and display the output.

Gdb says that the “*program exited with code 023*”

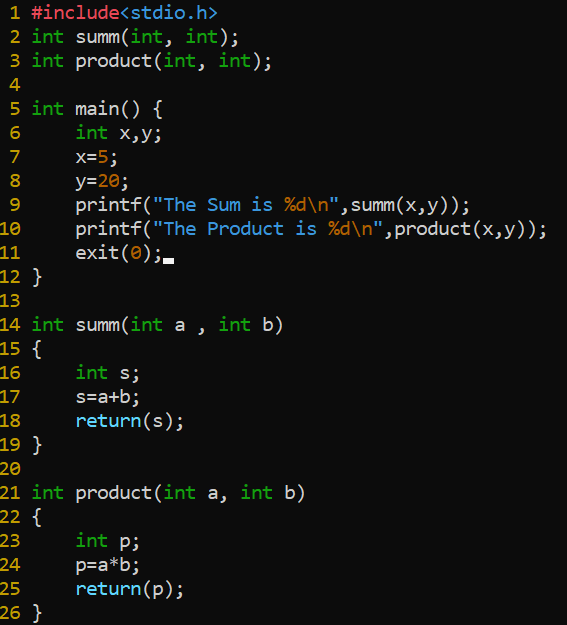
What does this mean ?

* For me, its showing process excited normally means it doesn’t find any error and compiled successfully.

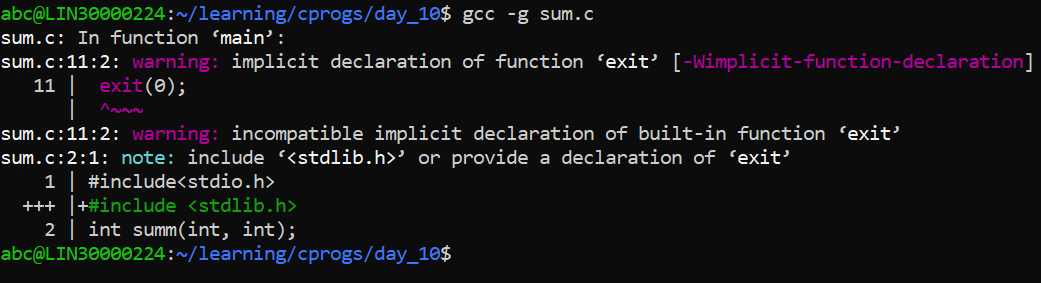
**Step 8**: Quit gdb

(gdb) *q*

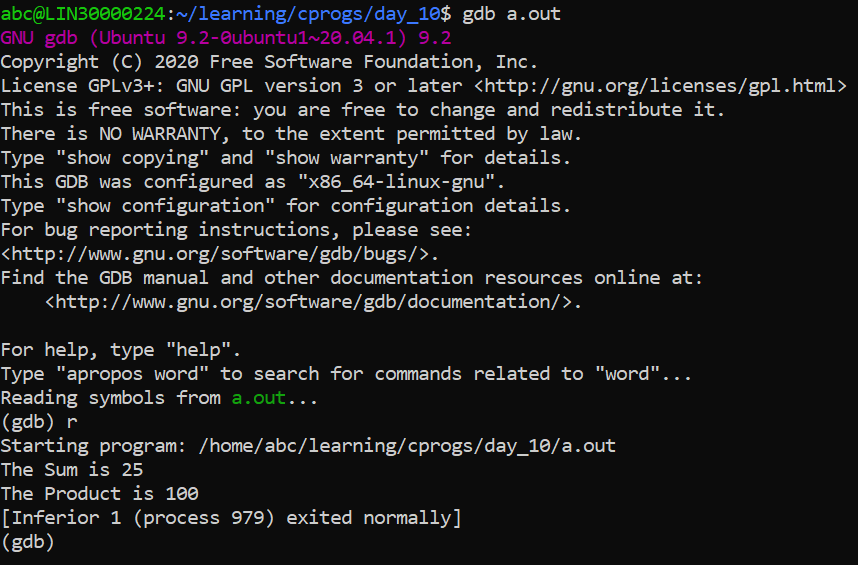
**Step 9**: Edit the code. In the main function, type *‘exit(0)*’ before the last *‘}’.*

* 

**Step 10**: Save and exit. Compile the code again with –g option. Invoke the output with gdb.



**Step 11**: Again run the program.



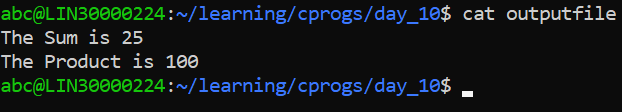
Now what do you observe ?

* **No change**

**Step 12**: Redirect the output to a file.

(gdb) *r > outputfile*

**Step 13**: quit gdb. List the contents of outputfile *cat outputfile*



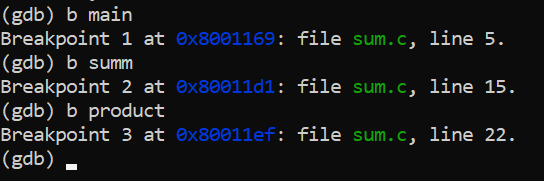
**Step 14**: Again invoke the output with gdb. List 40 lines of the code.

*set listsize 40*

**Step 15**: Set breakpoint. At main, function sum and function product.

(gdb) *b main*  (gdb) *b 23*

(gdb) *b product.*

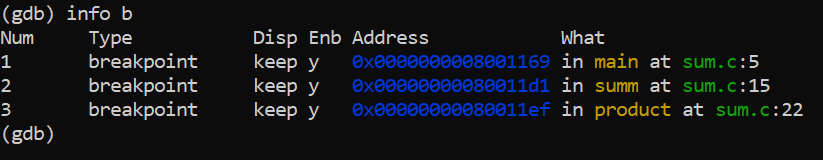


What do you observe ?

* **Breakpoints can be assigned using function name as well as line numbers**.

**Step 16**: List information about breakpoint.

(gdb) *info b*



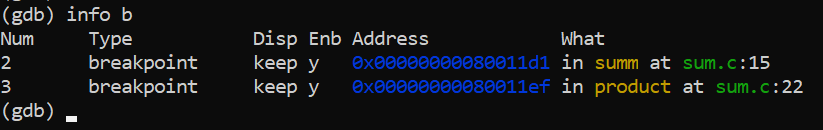
What do you observe?

* We got all information about all breakpoints.

**Step 17**: Delete breakpoint with id = 1

(gdb) *d 1*

**Step 18**: Again list information on break points.



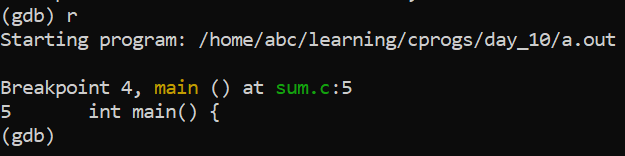
What do you observe?

* Breakpoint 1 , which was at main, got deleted.

**Step 19**: Again put a break point on main.

(gdb) *b main*

**Step 20**: Run the program

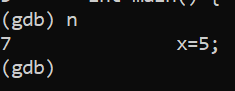


What do you observe?

* Gdb stops at the first breakpoint (main).

**Step 21**: To execute this line and go to next line, type the command n (next) and press enter.

(gdb*) n*

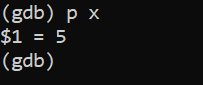


**Step 22**: Again type n



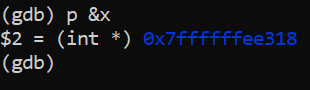
**Step 23**: Fnd out the value of a variable.

(gdb) *p x*

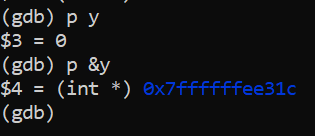


**Step 23** : Find the address of the variable.

(gdb*) p &x*



**Step 24**: Similarly find the value of variable y and address of y



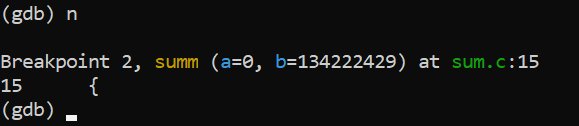
What do you observe? Can you explain the output? Can you explain the address of variable x and variable y.

* We can see that it prints the value and address of the variables ‘x’ and ‘y’.

The address shows that both variables are integer pointers and the memory locations of each is provided.

**Step 25**: Again execute the next line.

What do you observe ?

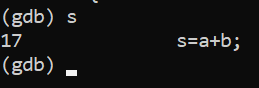


* Gdb has encountered the second breakpoint.

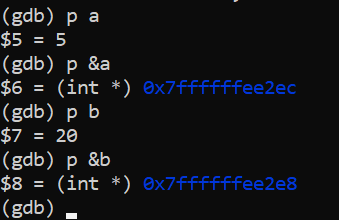
**Step 26:** You may enter the function and execute each line of function one by one.

(gdb) *s*

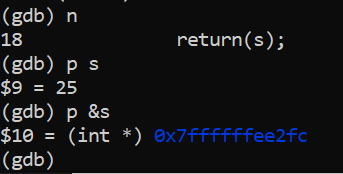
(Note the difference between n (next) and s (step). To go inside a function we use the command s.)



**Step 27**: Find out the value and address of variable a and variable b. What do you observe?

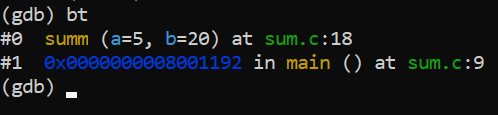


**Step 30**: Execute the next line by typing ‘n’. Find out the value and address of variable s.



**Step 31**: List out the number of active stack frames.

(gdb) *bt*

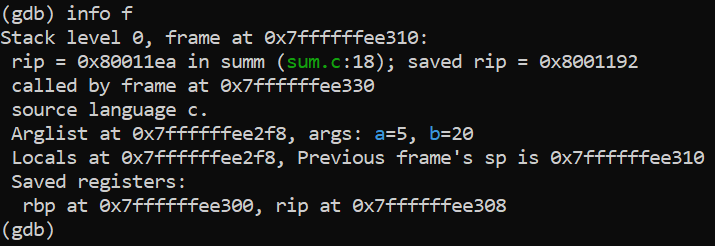


What do you observe?

* Displays all the current variable that is stored in stack.

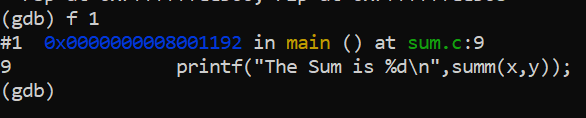
**Step 32** : Get info about current frame.

(gdb) *info f*

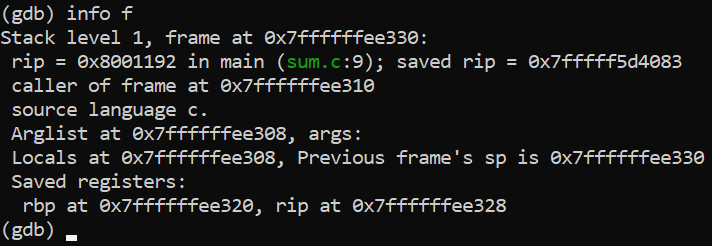


**Step 33**: Move to previous frame i.e frame number 1

(gdb*) f 1*



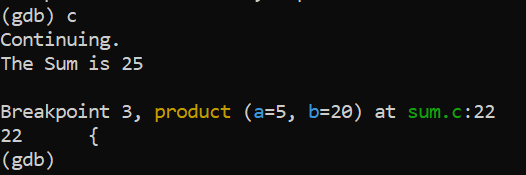
**Step 34**: List info about the current stack frame. What do you observe?



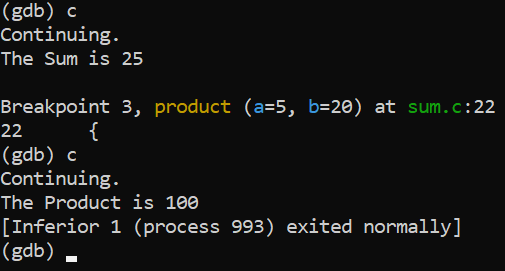
**Step 35**: Continue executing the code until you reach the next breakpoint.

(gdb) *c*

(Command c stands for continue.)



**Step 36**: Again use the command c and press enter



What do you observe?

* Function of product executed successfully

***TYPE THE FOLLOWING CODE USING VI EDITOR***.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This program finds the sum of n natural numbers , where n is passed as a command line argument..

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

int main(int argc, char\* argv[])

{

int input ,sum, count;

if (argc < 2)

{

printf("Enter the number as a command line arg\n"); exit(1);

}

//We need to convert the argument string to number

//Call Library Function atoi, which converts the string to number

input = atoi(argv[1]); sum = 0;

for(count=1 ; count <=input ; count++)

{

sum = sum + count;

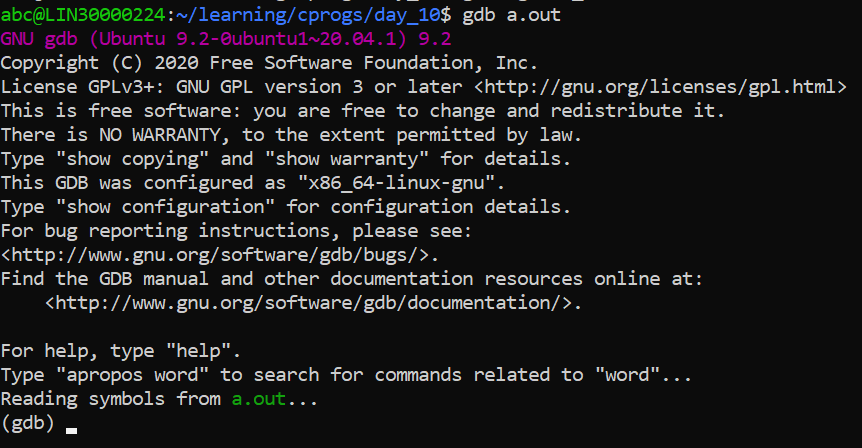
}

printf("The sum is %d\n",sum);

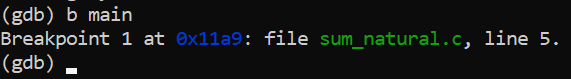
return 0;

}

**Step 1**: Compile the code using gcc with option –g and invoke gdb (**gdb a.out**)



**Step 2**: Assign a breakpoint at main. (**b main**)

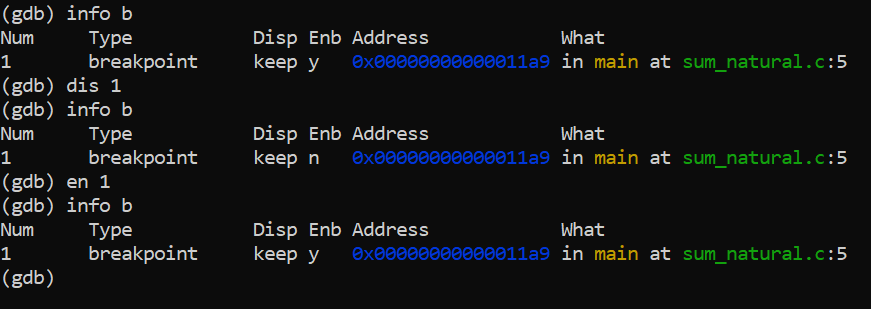


**Step 3**: Breakpoints can be disabled and enabled using the following commands:

(gdb) *dis <id>*

(gdb*) en <id>*

Check the status of the breakpoint after disabling and enabling the breakpoint



**Step 4**: A watchpoint is a special breakpoint that stops your program when the

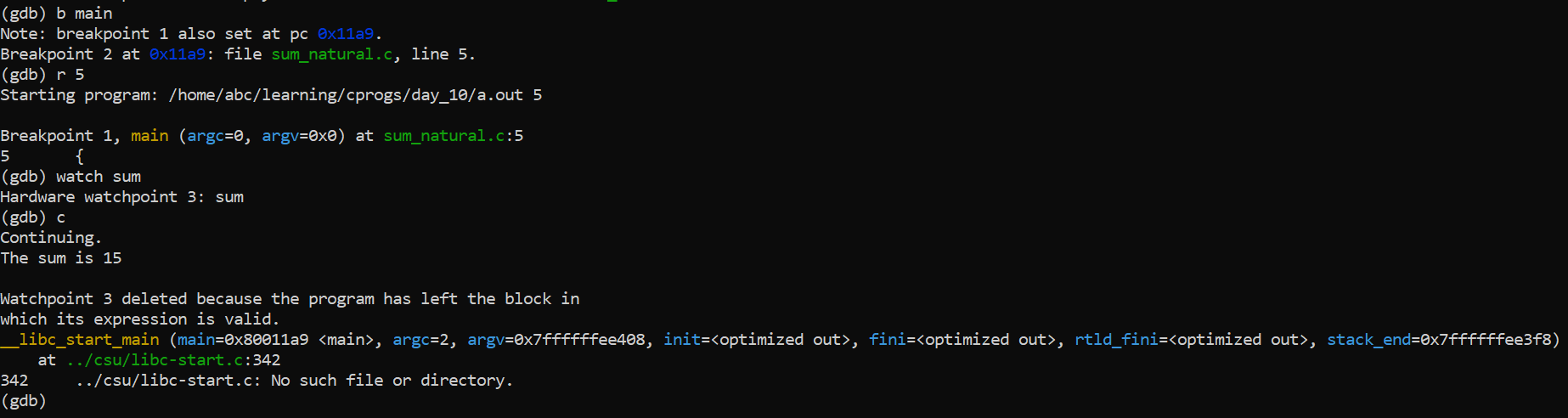
Value of an expression changes. Let us put a watchpoint on variable sum in main.

(gdb) *b main*

(gdb*) r 5 ( Here execute the code with command Line Argument -5)* (gdb) *n*

(gdb) *watch sum*

(gdb) *c*



What do you observe?

* **After you continue using command ‘c’, the program stops when the value of the variable sum changes.**

*Note: When you run the program using command r, you can provide the command line argument(Which is 5 in this example).*

***Some more gdb useful Commands***

To execute a shell command

gdb> shell <cmd to execute>

To view the contents of a memory location

# /15c <address> //display 15 characters

To set a break with condition

Consider the code below in myfile.c, where to break when i = 4

1. for (i = 0; i < 10; i++)
2. { 22 ret +=i;

....

25 }

gdb>break myfile.c:21 if (i == 4)

To view the assembly code with source

# gdb> disassem /m