**GDB Practice session**

Typographical conventions

We use the following conventions in this guide:

vi The name of a specific command or file

file You should replace file with a specific name

(gdb)help quit at the gdb prompt, type the command help quit, then press the <ENTER> key

Exit gdb Output that you see on the screen

Getting Started

1. Login into the Linux server with your login Ids. Open two terminals, one to work in gdb prompt and the other to work on the shell.

2. Create a new directory called gdb in your home directory <home>

mkdir gdb

3. Go inside the directory you have created in (2) /<home>/gdb

cd gdb

4. Copy the following files from the path as mentioned by the trainer:

a. gdb\_sample\_pointer\_1\_0.c

b. gdb\_sample\_core\_1\_0.c

c. bugs\_wall.c

d. first\_fit\_bugs.c

e. gdb\_practice\_array.c

5. Although all commands in gdb are full words, you can type any unambiguous prefix. For instance, p instead of print.

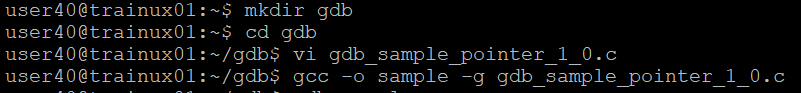
6. Use the help command in gdb for a list of topics. Then type help topic for information on a specific topic. help cmd gives help on the command cmd e.g. help print will give information about the print command. For comprehensive coverage of gdb, read the Info pages. At the shell prompt, type info. When info starts, type mgdb<ENTER>.

(Please note that the info command is not present on the server 10.203.161.9)

Compilation

7. Compile the file gdb\_sample\_pointer\_1\_0.c and put the output in the executable file called sample

gcc –o sample –g gdb\_sample\_pointer\_1\_0.c



Execution with gdb

8. Execute the file sample with gdb

gdb sample

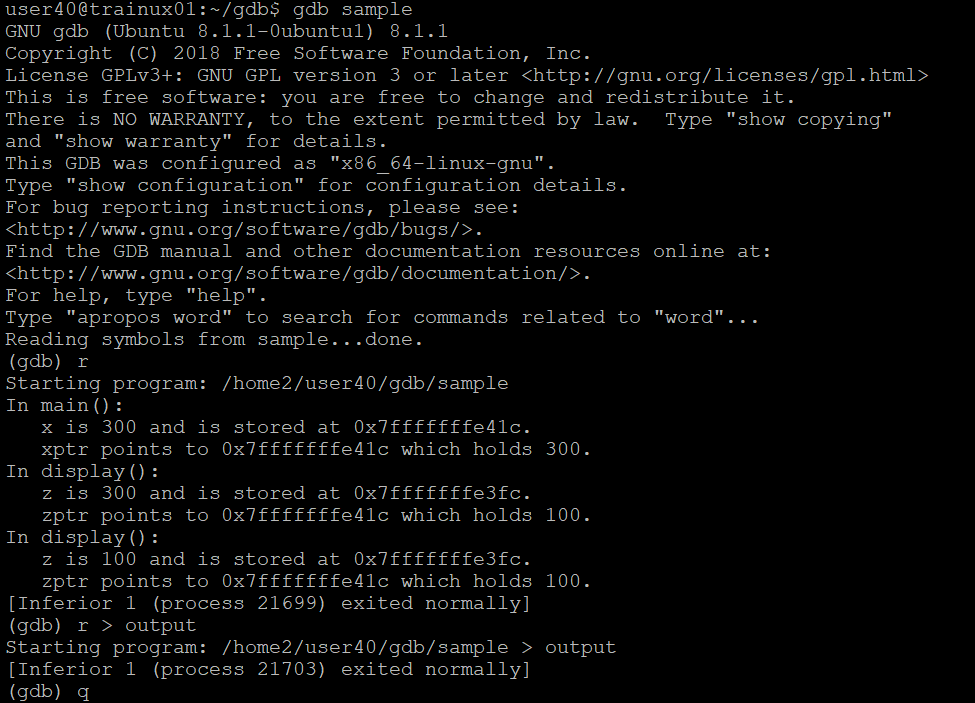
(gdb) r

9. Execute the file again, this time redirecting the output of the program to another file

(gdb) r > output

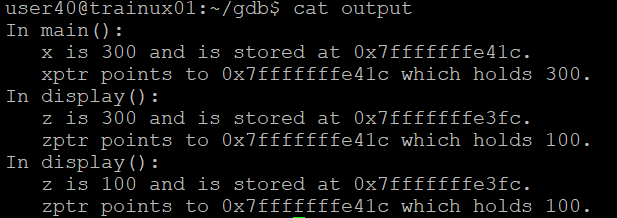
Quit gdb

(gdb) q



View the Output file

cat output

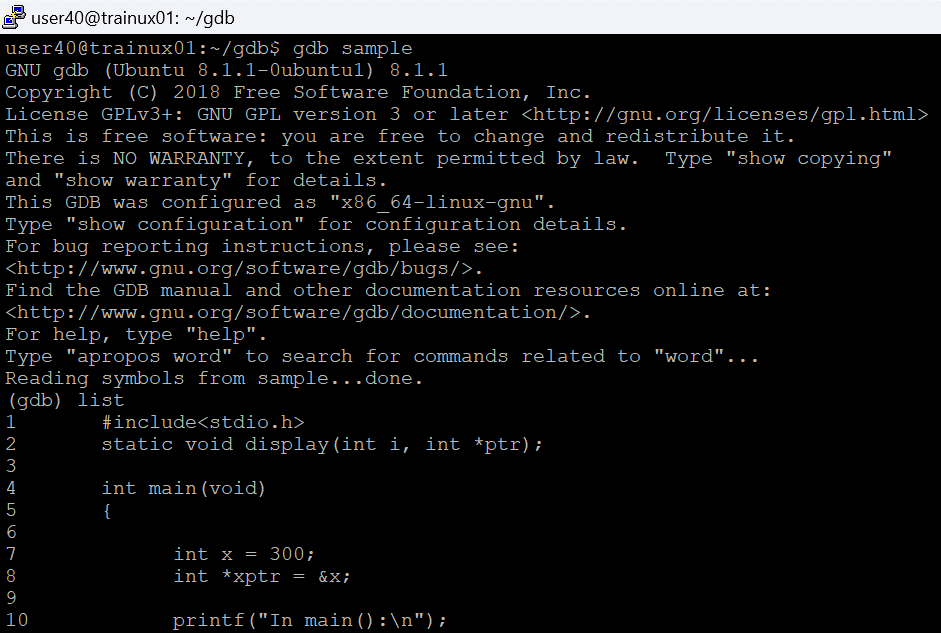


10. To have a look at the source code at any point in time

gdb sample

(gdb) list

By default the number of lines displayed is 10, to change the number of lines to be displayed



(gdb) set listsize 20

(gdb) list

A computer screen with white text

Description automatically generated

Breakpoints

11. Before executing the program this time, put a breakpoint

(gdb) b main

(gdb) r

Breakpoints can be put using either:

· the function name or

· the line number or

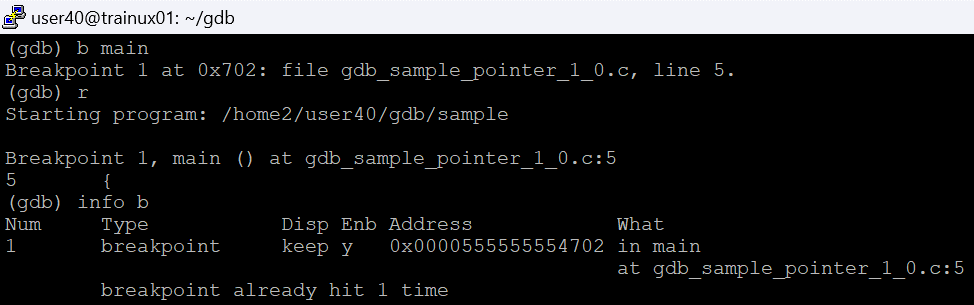
· source file name: line number

Try putting the breakpoint using all the above ways

12. Information of all the set breakpoints can be seen

(gdb) info b

The first column shows the id of the breakpoint



13. Delete all the breakpoints except the one on function main

(gdb) d id1 id2 …



14. Now run the program and observe that the execution stops at the first breakpoint that is encountered. Now the control of the program is with gdb and each statement can be executed one at a time using the command:

(gdb) n

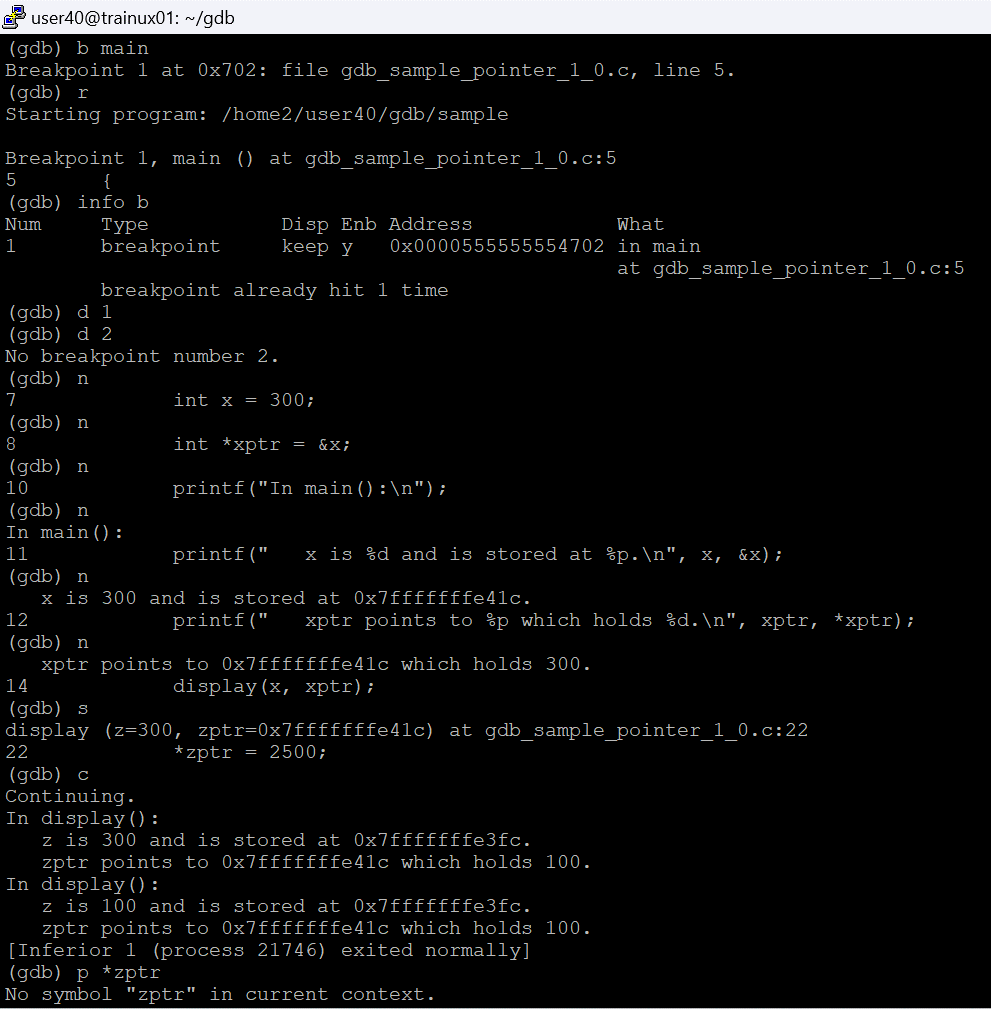
Run the n command till the statement before the call to function display is executed

15. Step inside the function display

(gdb) s

16. Let the program continue on its own

(gdb) c



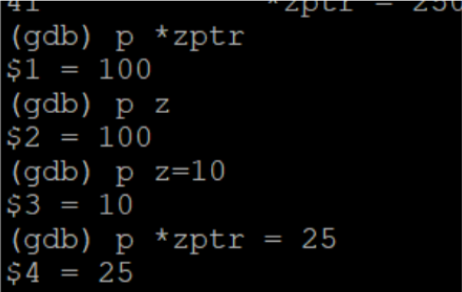
17. Lets now look into the contents of a variable and alter the value

(gdb) p \*zptr

(gdb) p z

(gdb) p z = 10

(gdb) p \*zptr = 25



p command can also be used to compute an expression or a function

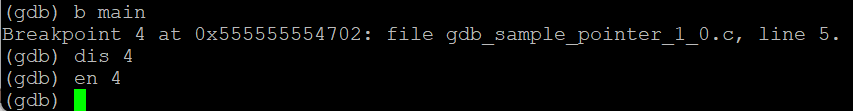
(gdb) p strlen(“abc”)

$4 = 3

18. 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

19. A watchpoint is a special breakpoint that stops your program when the value of an expression changes. Lets put a watchpoint on variable x in main

(gdb) b main

(gdb) r

(gdb) n

keep running the n command till the variable x is defined

(gdb) watch x

(gdb) c

What do you observe?

A screen shot of a computer

Description automatically generated

20. Commands to explore,

(gdb) help cmd



a. ignore command with breakpoints: ignore <id> <number of times to be ignored>

b. Conditional breakpoints: e.g. b main if x==6, b display if z==5

c. Until command: Continue running until a source line past the current line, in the current stack frame, is reached.

(gdb) b main

(gdb) r

(gdb) until 16

What code scenarios can you think where the ignore and until commands can be used?

21. Execute shell commands on the gdb prompt

(gdb) shell cat gdb\_sample\_pointer\_1\_0.c

A computer screen with white text

Description automatically generated

22. Try moving your source file gdb\_sample\_pointer\_1\_0.c to some other directory

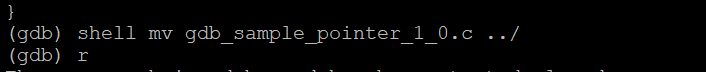
(gdb) shell mv gdb\_sample\_pointer\_1\_0.c ../

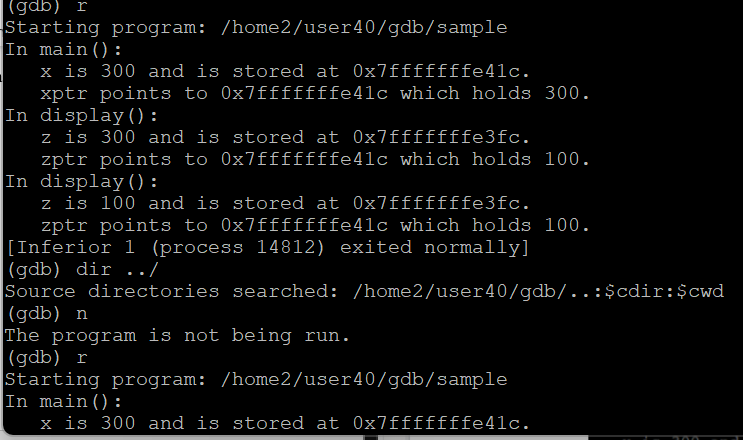
(gdb) r

gdb\_sample\_pointer\_1\_0.c: No such file or directory

(gdb) dir ../

(gdb) n





23. The same commands need not be typed again and again on each invocation of gdb but take the input from a file. Create a file called commands with the following lines :

b main

b display

Now run sample with gdb

gdb sample

(gdb) sou commands

A black background with white text

Description automatically generated

24. You may want to save the output of gdb commands to a file.

(gdb) set logging on

Give a few commands to gdb as discussed earlier

(gdb) shell cat gdb.txt

What do you observe?

(Please note that the set logging on command does not work with the gdb version installed on the 10.203.161.9 server)

Stack Frames

A black background with white text

Description automatically generated

25. When your program has stopped, the first thing you need to know is where it stopped and how it got there. View the stack frames of the process using the back trace command

(gdb) b display

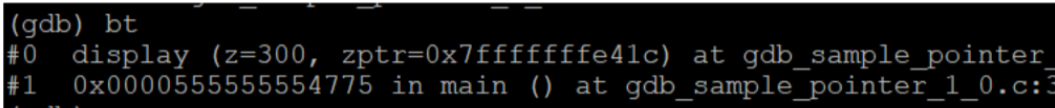
(gdb) r

(gdb) bt

#0 display (z=5, zptr=0x7fbffff88c) at gdb\_sample\_pointer\_1\_0.c:27

#1 0x000000000040051d in main () at gdb\_sample\_pointer\_1\_0.c:21

The leftmost column gives the frame id



Now, try to see the value of x (local variable of function main)

(gdb) p x

No symbol "x" in current context.

If you do want to see the value of x at this point in time, change the frame to 1 and proceed

(gdb) fr 1

(gdb) p x

What do you observe?

Analysis of core dumps

A black screen with white text

Description automatically generated

**Assignments**

**31. Assignment: gdb\_practice\_array.c contains a code with bugs. The actual output is supposed to generate a sorted list and then search for 3 strings in that as shown in actual\_output. Find out the bugs in practise.c using gd**

A screenshot of a computer

Description automatically generated

**Key Issues:**

1. **Infinite Loop in Main**: You're using i in the while loop in main() without resetting it after printing the original list of critters. This results in skipping the second loop that prints the sorted list.
2. **swap and med3 functions**: You have a mix of inline and macro versions of swap and med3, but these are not implemented correctly. Some parts of the code are redundant or not well-defined.
3. **Missing headers**: The stdio.h, string.h, and sys/types.h headers are included, but there is no need for sys/types.h. The standard headers should be sufficient.
4. **SWAPINIT and vecswap logic**: The SWAPINIT and vecswap macros are intended to handle platform-specific optimizations, but their use is incorrect and not needed for most common cases. They could be simplified for this task.
5. **Logic in qsort1**: The qsort1 function itself is too complex for this context, and the implementation could be simplified. The swap logic is also a bit problematic.