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

```
~$ mkdir gdb
~$ cd gdb
~/gdb$ vi gdb_sample_pointer_1_0.c
~/gdb$ gcc -o sample -g gdb_sample_pointer_1_0.c
~/gdb$ gdb sample
```

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

```
(gdb) r
Starting program: /home2/user46/gdb/sample
In main():
   x is 300 and is stored at 0x7fffffffe3fc.
   xptr points to 0x7fffffffe3fc which holds 300.
In display():
   z is 300 and is stored at 0x7fffffffe3dc.
   zptr points to 0x7fffffffe3fc which holds 100.
In display():
   z is 100 and is stored at 0x7fffffffe3dc.
   zptr points to 0x7fffffffe3fc which holds 100.
[Inferior 1 (process 4516) exited normally]
(gdb) r > output
Starting program: /home2/user46/gdb/sample > output
[Inferior 1 (process 4611) exited normally]
(gdb) q
```

View the Output file
cat output

```
In main():
   x is 300 and is stored at 0x7fffffffe3fc.
   xptr points to 0x7fffffffe3fc which holds 300.
In display():
   z is 300 and is stored at 0x7fffffffe3dc.
   zptr points to 0x7fffffffe3fc which holds 100.
In display():
   z is 100 and is stored at 0x7fffffffe3dc.
   zptr points to 0x7fffffffe3fc which holds 100.
user46@trainux01:~/gdb$ gdb sample
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
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 sample...done.
```

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

```
(gdb) list
10      Owner                   :       Sibu Cyriac
11
12      Date of Modification    :
13
14      Reason for modification :
15
16      Modifier                :
17
18      ********************************************************************
*******/
19
(gdb) set listsize 20
(gdb) list
20      #include<stdio.h>
21      static void display(int i, int *ptr);
22
23      int main(void)
24      {
25
26            int x = 300;
27            int *xptr = &x;
28
29            printf("In main():\n");
30            printf("   x is %d and is stored at %p.\n", x, &x);
31            printf("   xptr points to %p which holds %d.\n", xptr, *xptr);
32
33            display(x, xptr);
34            display(x,xptr);
35            return 0;
36      }
37
38      void display(int z, int *zptr)
39      {
```

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

```
(gdb) b main
Breakpoint 1 at 0x702: file gdb_sample_pointer_1_0.c, line 24.
(gdb) r
Starting program: /home2/user46/gdb/sample

Breakpoint 1, main () at gdb_sample_pointer_1_0.c:24
24      {
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555554702 in main
                                                   at gdb_sample_pointer_1_0.c:2
4
        breakpoint already hit 1 time
```

The first column shows the id of the breakpoint

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

(gdb) d id1 id2 …

```
(gdb) d 1
(gdb) d 2
```

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

```
(gdb) n
26          int x = 300;
(gdb) n
27          int *xptr = &x;
(gdb)
29          printf("In main():\n");
(gdb)
In main():
30          printf("   x is %d and is stored at %p.\n", x, &x);
(gdb)
   x is 300 and is stored at 0x7fffffffe3fc.
31          printf("   xptr points to %p which holds %d.\n", xptr, *xptr);
(gdb)
   xptr points to 0x7fffffffe3fc which holds 300.
33          display(x, xptr);
(gdb) s

Breakpoint 3, display (z=300, zptr=0x7fffffffe3fc)
    at gdb_sample_pointer_1_0.c:41
41          *zptr = 2500;
(gdb) c
Continuing.
In display():
   z is 300 and is stored at 0x7fffffffe3dc.
   zptr points to 0x7fffffffe3fc which holds 100.

Breakpoint 3, display (z=100, zptr=0x7fffffffe3fc)
    at gdb_sample_pointer_1_0.c:41
41          *zptr = 2500;
```

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

```
(gdb) p *zptr
$1 = 100
(gdb) p z
$2 = 100
(gdb) p z=10
$3 = 10
(gdb) p *zptr = 25
$4 = 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

```
(gdb) dis 3
(gdb) en 3
```

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?

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

```
(gdb) n
26              int x = 300;
(gdb)
27              int *xptr = &x;
(gdb)
29              printf("In main():\n");
(gdb) watch x
Hardware watchpoint 6: x
(gdb) c
Continuing.
In main():
   x is 300 and is stored at 0x7fffffffe3fc.
   xptr points to 0x7fffffffe3fc which holds 300.

Breakpoint 3, display (z=300, zptr=0x7fffffffe3fc)
    at gdb_sample_pointer_1_0.c:41
41              *zptr = 2500;
(gdb) help cmd
Undefined command: "cmd".  Try "help".
(gdb) until 50
No line 50 in the current file.
(gdb) until 38

Hardware watchpoint 6: x

Old value = 300
New value = 2500
display (z=300, zptr=0x7fffffffe3fc) at gdb_sample_pointer_1_0.c:43
43              printf("In display():\n");
```

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

```
user46@trainux01: ~/gdb
(gdb) shell cat gdb_sample_pointer_1_0.c

/*****************************************************************************
keywords                :       gdb

File Name               :       gdb_sample_llist_1_0.c

Date of Creation        :       29//2007

Owner                   :       Sibu Cyriac

Date of Modification    :

Reason for modification :

Modifier                :

*****************************************************************************/

#include<stdio.h>
static void display(int i, int *ptr);

int main(void)
{

      int x = 300;
      int *xptr = &x;

      printf("In main():\n");
      printf("   x is %d and is stored at %p.\n", x, &x);
      printf("   xptr points to %p which holds %d.\n", xptr, *xptr);

      display(x, xptr);
      display(x,xptr);
      return 0;
}

void display(int z, int *zptr)
{

      *zptr = 2500;
```

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

```
user46@trainux01: ~/gdb

(gdb) shell mv gdb_sample_pointer_1_0.c ../
(gdb) r
Starting program: /home2/user46/gdb/sample
In main():
   x is 300 and is stored at 0x7fffffffe41c.
   xptr points to 0x7fffffffe41c which holds 300.
In display():
   z is 300 and is stored at 0x7fffffffe3fc.
   zptr points to 0x7fffffffe41c which holds 100.
In display():
   z is 100 and is stored at 0x7fffffffe3fc.
   zptr points to 0x7fffffffe41c which holds 100.
[Inferior 1 (process 8905) exited normally]
(gdb) dir ../
Source directories searched: /home2/user46/gdb/..:$cdir:$cwd
(gdb) n
The program is not being run.
(gdb) r
Starting program: /home2/user46/gdb/sample
In main():
   x is 300 and is stored at 0x7fffffffe41c.
   xptr points to 0x7fffffffe41c which holds 300.
In display():
   z is 300 and is stored at 0x7fffffffe3fc.
   zptr points to 0x7fffffffe41c which holds 100.
In display():
   z is 100 and is stored at 0x7fffffffe3fc.
   zptr points to 0x7fffffffe41c which holds 100.
[Inferior 1 (process 9174) exited normally]
(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

```
(gdb) sou commands
Breakpoint 1 at 0x702: file gdb_sample_pointer_1_0.c, line 24.
Breakpoint 2 at 0x7b0: file gdb_sample_pointer_1_0.c, line 41.
(gdb)
```

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)

```
(gdb) set logging on
Copying output to gdb.txt.
(gdb) shell cat gdb.txt
```

Stack Frames

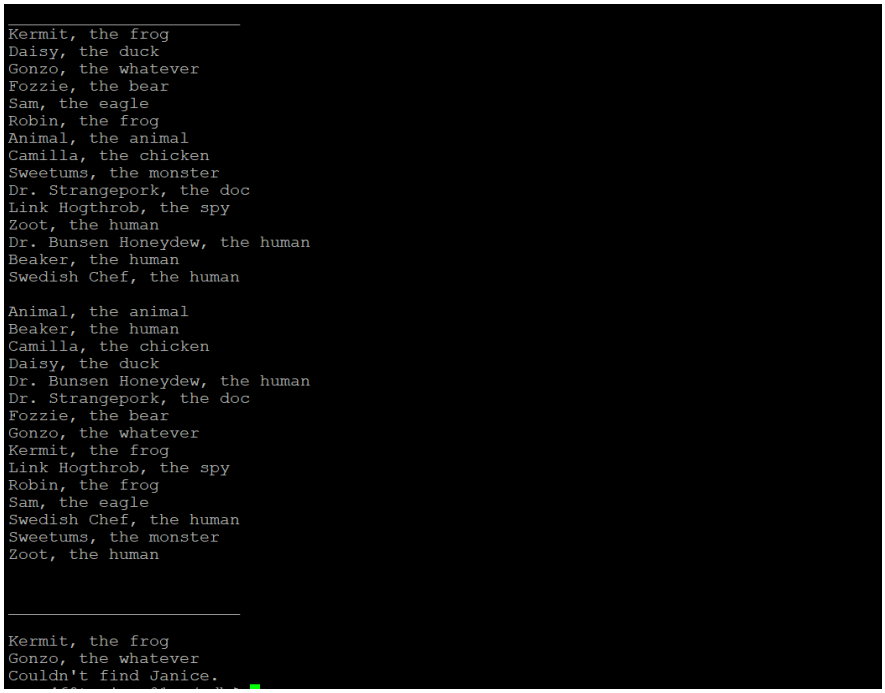
25. When your program has stopped, the first thing you need to know is where it stopped and how it got there.
View the stack frames of the process using the back trace command
(gdb) b display
(gdb) r
(gdb) bt
#0 display (z=5, zptr=0x7fbffff88c) at gdb_sample_pointer_1_0.c:27
#1 0x000000000040051d in main () at gdb_sample_pointer_1_0.c:21
The leftmost column gives the frame id

```
(gdb) bt
#0  display (z=300, zptr=0x7fffffffe41c) at gdb_sample_pointer_1_0.c:41
#1  0x0000555555554775 in main () at gdb_sample_pointer_1_0.c:33
```

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?

```
(gdb) p x
No symbol "x" in current context.
(gdb) fr 1
#1  0x0000555555554775 in main () at gdb_sample_pointer_1_0.c:33
33      in gdb_sample_pointer_1_0.c
(gdb) p x
$1 = 300
```

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 gdb.
A. Here by debugging found that the while loop is not getting executed as it is not initialized. So the bugs are rectified and the required output is displayed


Kermit, the frog
Daisy, the duck
Gonzo, the whatever
Fozzie, the bear
Sam, the eagle
Robin, the frog
Animal, the animal
Camilla, the chicken
Sweetums, the monster
Dr. Strangepork, the doc
Link Hogthrob, the spy
Zoot, the human
Dr. Bunsen Honeydew, the human
Beaker, the human
Swedish Chef, the human
Animal, the animal
Beaker, the human
Camilla, the chicken
Daisy, the duck
Dr. Bunsen Honeydew, the human
Dr. Strangepork, the doc
Fozzie, the bear
Gonzo, the whatever
Kermit, the frog
Link Hogthrob, the spy
Robin, the frog
Sam, the eagle
Swedish Chef, the human
Sweetums, the monster
Zoot, the human
Kermit, the frog
Gonzo, the whatever
Couldn't find Janice.