UE18CS257C: SECURE PROGRAMMING WITH C

Lab 1: Intro to Debugging C Programs with GDB

What Is gdb?

The most widely used tool for debugging C programs is the GDB. GDB stands for GNU Debugger, is a powerful text debugger that will let you do many things

gdb is a special program that lets you inspect other programs as they run. For example, you can use gdb to spy on the values of variables in a C program you're working on to see if there are any problems with your logic.

Errors in programs are often called *bugs*. If a program unexpectedly stops executing because it has a bug, we say the program has *crashed*. Not all bugs cause programs to crash: bugs can cause all sorts of unexpected and often damaging behaviour. Examples include printing incorrect information, refusing to accept input from users, corrupting databases, and (in rare cases) even erasing entire disks.

Command Summary

Command	Shortcut	Explanation
run	r	Run your program
list 12	l 12	Display the source code at line 12
list myfun	l myfun	List your source code starting at the definition of function myfun
print var	p var	Print the current value of var. As well as variables, you can also print most C expressions, e.g. myList[i] + 10, student.name. Note that #defined values don't work
backtrace	bt	Print a backtrace of the functions that have been called to reach the current execution point in your program
break 10	b 10	Set a breakpoint at line 10 of your code.
break myfun	b myfun	Set a breakpoint at the start of the function myfun in your code.
delete 3	d 3	Delete breakpoint number 3.
continue	c	Continue running your program after a breakpoint has paused it.

next	n	If your program is paused, execute the next line of code. Execute entire function calls rather than stepping into them
step	s	If your program is paused, execute the next line of code, stepping into any function calls rather than executing them completely.
quit	q	Quit gdb.

Prerequisite:

Install gdb package on Ubuntu 16.04 (Xenial Xerus)

```
$ sudo apt-get update
$ sudo apt-get install gdb
```

Exercise 1: To inspect the values of the variables and trace the given C program execution

```
#include <stdio.h>
int sum=0, val , nu m =0;
double avg;
int main()
{       while (scanf("%d",&val )!= E O F)
        {       sum += val;num++;}
        if (num > 0)
        {       avg = sum/num;
                printf("Average is %f\n", avg);}
}
```

Step 1 : Compile pgm2.c with -g option and start gdb with the executable loaded

Before using gdb to debug your code, you should compile your code with the -ggdb or -g option. For example instead of typing:

$ gcc -g pgm2.c

$ gdb a.out

gdb will print some information about itself and then prompt you to type gdb commands. You'll see something like this:

```
ns@pes$ gcc -g pgm2.c
ns@pes$ gdb a.out
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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...done.
```

The -ggdb option tells the compiler to insert debugging information into the compiled code. That information enables gdb to do its job.

If you try to use gdb on code that hasn't been compiled with the debugging information, you'll typically see mysterious errors like these:

No symbol "myVariable" in current context.
Function "myFunction" not defined.
No line 34 in file "../sysdeps/i386/elf/start.S".

Recompile your code with -ggdb option and the errors should disappear.

Step 2: List the program to know the line number

(gdb) l

```
(gdb) l
1       #include<stdio.h>
2       int sum=0,val,num=0;
3       double avg;
4       int main()
5       {
6               while(scanf("%d",&val)!=EOF)
7               {
8                       sum+=val;
9                       num++;
10              }
(gdb)
11              if(num > 0)
12              {
13                      avg=sum/num;
14                      printf("Average is %f\n", avg);
15              }
16              return 0;
17      }
18
```

Step 3: Set breakpoints at specific line

(gdb) b 6

```
(gdb) b 6
Breakpoint 1 at 0x40059e: file pgm2.c, line 6.
(gdb) b 13
Breakpoint 2 at 0x4005e6: file pgm2.c, line 13.
```

Step 4: Start the execution and display the contents of the variable

(gdb) r

(gdb) display num

```
(gdb) r
Starting program: /home/osboxes/a.out

Breakpoint 1, main () at pgm2.c:6
6               while(scanf("%d",&val)!=EOF)
(gdb) display num
1: num = 0
(gdb) display val
2: val = 0
(gdb) display sum
3: sum = 0
```

Step 5: Continue the execution after breakpoint 1 and read input

(gdb) c

```
(gdb) c
Continuing.
1
2
3

Breakpoint 2, main () at pgm2.c:13
13                      avg=sum/num;
1: num = 3
2: val = 3
3: sum = 6
```

Step 6: Step up every line of execution

(gdb) step

```
(gdb) step
14                      printf("Average is %f\n", avg);
1: num = 3
2: val = 3
3: sum = 6
(gdb)
__printf (format=0x4006b7 "Average is %f\n") at printf.c:28
28      printf.c: No such file or directory.
1: num = 3
2: val = 3
3: sum = 6
(gdb)
32      in printf.c
1: num = 3
2: val = 3
3: sum = 6
(gdb)
33      in printf.c
1: num = 3
2: val = 3
3: sum = 6
(gdb) step
32      in printf.c
1: num = 3
2: val = 3
3: sum = 6
```

Step 7: Continue the execution to fetch the result and stop debugging

(gdb) c

(gdb) q

```
(gdb) c
Continuing.
Average is 2.000000
[Inferior 1 (process 2308) exited normally]
(gdb) q
```

Exercise 1A: Answer the following questions with screenshots for each

Note:

- *Modify the given program according to the constraints defined in the question if necessary*

1A.1 How do you pass command line arguments to a program when using gdb?

1A.2 How do you set a breakpoint which only occurs when a set of conditions is true (eg when certain variables are a certain value)?

1A.3 If the next line is a function call, you'll execute the call in one step. How do you execute the C code, line by line, inside the function call?

1A.4 How do you configure gdb so it prints the value of a variable after every step?

1A.5 How do you print a list of all variables and their values in the current function?

Exercise 2: Where Did the Program Crash?

Often, if you run a program with a bug from the shell, the program just stops running and the shell prints a message like "Segmentation fault" or "Illegal instruction". That's not very helpful. If you run your program from inside gdb instead, gdb will tell you what line of your program was executing when the program stopped.

Consider the following code and answer the following questions with screenshots for each

```
1 : #include <stdio.h>
2 : #include <stdlib.h>

3 : int main(int argc, char **argv)
4 : {
5 :   char *buf;
6 :
7 :   buf = malloc(1<<31);
8 :
9 :   fgets(buf, 1024, stdin);
10:   printf("%s\n", buf);
11:
12:   return 1;
13: }
```

2.1 Backtrace and identify where exactly the program crashed?

2.2 Set breakpoint at line 8 and Illustrate step by step execution of the program and print the buffer contents

2.3 What section of code has to be modified so that the segmentation fault doesn't exist?

2.4 After modification, again debug the program to ensure the segmentation fault doesn't exist.

Submission:

Prepare a document answering Exercise 1 and Exercise 2 questions accompanied with screenshots and upload the same within the lab hours. Plagiarism strictly to be avoided.