

GDBlesson

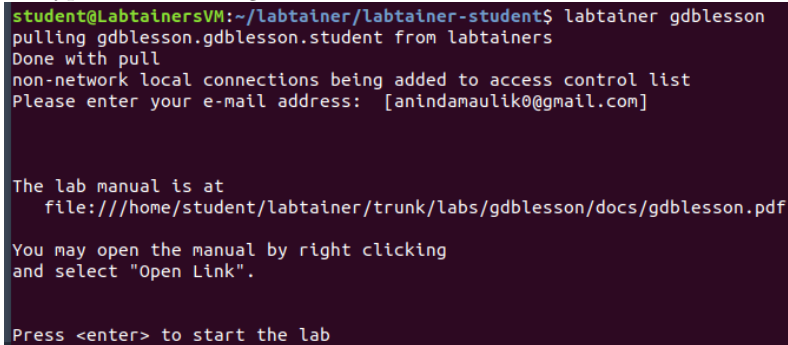
Aninda Maulik

September 2020

1 Introduction

The following is the basic directory student student@LabtainersVM: ~/labtainer/labtainer-student\$

we type in labtainer gdblesson



```
student@LabtainersVM:~/labtainer/labtainer-student$ labtainer gdblesson
pulling gdblesson.gdblesson.student from labtainers
Done with pull
non-network local connections being added to access control list
Please enter your e-mail address: [anindamaulik@gmail.com]

The lab manual is at
file:///home/student/labtainer/trunk/labs/gdblesson/docs/gdblesson.pdf

You may open the manual by right clicking
and select "Open Link".

Press <enter> to start the lab
```

then we press Enter, based on the instruction of the above image, and enter the lab

after entering the lab, we type in

less sampleMath.c

Following this, due to the above command, we get the below code

```
#include <stdio.h> /* Stdio.h is known to contain the input and output
operations like "printf" or "scanf" etc. stdio.h is the header file for
standard input and output. This is useful for getting the input from
the user(Keyboard) and output result text to the monitor(screen). */
void main() { /* the main function */
    int num; /* initialising the integer variables num,count,total */
    int count;
    int total;
    total = 0; /* assigning the values to the initialised variables */
    num = 6;
    count = 15;
    while(count > 0) { /* count=15 which is less than 0 */
```

```

        total = count / num; /*total=15/6=2.5 which is not an int */
        count--; /*count goes to 14 from 15 */
        num--; /* num goes to 5 instead of 6 */
        printf("Total is: %d\n", total); /* %d refers to an integer,
        but total is int, and hence expecting an error */
    }
}

```

- now compiling

```
gcc -g sampleMath.c -o sampleMath
```

- GNU Compiler Collections, GCC; -g stands for default debug information

```

ubuntu@gdblesson:~$ less sampleMath.c
ubuntu@gdblesson:~$ gcc -g sampleMath.c -o sampleMath
ubuntu@gdblesson:~$ ./sampleMath
Total is: 2
Total is: 2
Total is: 3
Total is: 4
Total is: 5
Total is: 10
/sbin/exec_wrap.sh: line 11: 444 Floating point exception(core dumped) ./sampleMath

```

- Now, we get into GDB, which will help us analyzing, what's throwing an exception. And for that we have to type "gdb sampleMath" into the command like

```

Total is: 5
Total is: 10
/sbin/exec_wrap.sh: line 11: 762 Floating point exception(core dumped) ./sampleMath
ubuntu@gdblesson:~$ gdb sampleMath
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 sampleMath...done.
(gdb) list

```

- we type in "r" to to run the program within GDB and notice an error at the end of the program within GDB

```

(gdb) r
Starting program: /home/ubuntu/sampleMath
Total is: 2
Total is: 2
Total is: 3
Total is: 4
Total is: 5
Total is: 10

Program received signal SIGFPE, Arithmetic exception.
0x0000000000400549 in main () at sampleMath.c:10
10         total = count / num;
(gdb)

```

- to better understand the source of the problem, we attempt to list the contents of the program by using "list"

```

(gdb) list
5      int total;
6      total = 0;
7      num = 6;
8      count = 15;
9      while(count > 0) { /* Modify this line only */
10         total = count / num;
11         count--;
12         num--;
13         printf("Total is: %d\n", total);
14     }
(gdb)

```

- The complete program shows up with line numbers now

```

(gdb) list
5      int total;
6      total = 0;
7      num = 6;
8      count = 15;
9      while(count > 0) { /* Modify this line only */
10         total = count / num;
11         count--;
12         num--;
13         printf("Total is: %d\n", total);
14     }
(gdb)
15     }
(gdb)
Line number 16 out of range; sampleMath.c has 15 lines.
(gdb)
Line number 16 out of range; sampleMath.c has 15 lines.
(gdb)

```

- A couple of taps to the "enter" key on the key board, takes us to the end of the program and later a couple of more strokes to "ENter" key shows us the end of the program.
- we try to now locate the initiation line of the while loop and subsequently type "break ", represents the line number and we have the following output based on snap

```

Line number 16 out of range; sampleMath.c has 15 lines.
(gdb) break 9
Breakpoint 1 at 0x400543: file sampleMath.c, line 9.
(gdb)

```

- if we hit the "run" command again based on the documentation then the message comes up as below

Use the "run" command again. Use the "p" or "print" command to display the value of any

```

Breakpoint 1 at 0x400543: file sampleMath.c, line 9.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) n
Program not restarted.
(gdb)

```

Use the “run” command again. Use the “p” or “print” command to display the value of any variable, just type its name after the “p” or “print” command. Type “s” or “step” to continue running one line through the program. Periodically print the value of the number until you see it reach zero. Once you realize what the issue with this code is type “q” to quit GDB. Then edit the source code

- so we type in something like p num and get a 0, somehow, really dont understand the reference. Tried to check some other values with p and not much is going to my head. Right now, try to push the command ”s” and it goes off.

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) n
Program not restarted.
(gdb) p num
$1 = 0
(gdb)
$2 = 0
(gdb) p total
$3 = 10
(gdb)
$4 = 10
(gdb) p count
$5 = 9
(gdb) p total
$6 = 10
(gdb) s

Program terminated with signal SIGFPE, Arithmetic exception.
The program no longer exists.
(gdb) p num
No symbol "num" in current context.
(gdb)
```

- re-initialised the program with the command ”r” and it takes us to the point where we initialised a change by introducing a break

```
(gdb) p total
$6 = 10
(gdb) s

Program terminated with signal SIGFPE, Arithmetic exception.
The program no longer exists.
(gdb) p num
No symbol "num" in current context.
(gdb) r
Starting program: /home/ubuntu/sampleMath

Breakpoint 1, main () at sampleMath.c:9
9          while(count > 0) { /* Modify this line only */
(gdb) █
```

```

Breakpoint 1, main () at sampleMath.c:9
9          while(count > 0) { /* Modify this line only */
(gdb) p num
$9 = 6
(gdb) break 9
Note: breakpoint 1 also set at pc 0x400543.
Breakpoint 2 at 0x400543: file sampleMath.c, line 9.
(gdb) p total
$10 = 0
(gdb) p count
$11 = 15
(gdb) s
10          total = count / num;
(gdb) p count
$12 = 15
(gdb) p num
$13 = 6
(gdb) p total
$14 = 0
(gdb) s
11          count--;
(gdb) p count
$15 = 15
(gdb) p total
$16 = 2
(gdb) p num
$17 = 6
(gdb) s
12          num--;
(gdb) p count
$18 = 14
(gdb) p total
$19 = 2
(gdb) p num
$20 = 6
(gdb) s
13          printf("Total is: %d\n", total);
(gdb) p num
$21 = 5
(gdb) p count
$22 = 14
(gdb) p total
$23 = 2
(gdb)

```

- let's look at the above screenshot for a moment
- we check the status of the variables at the beginning of the program

- and we get the values of total,count,num as 0,15,6 respectively
- now we hit "s" for "step" and get the situation total=count/num
- am guessing since the breakpoint is set to
`while(count > 0) {`
- which is also the line number 9, hence pushing the command "s" is taking us to the step after line 9
- at the time of, total=count/num, this operation has not been initiated so the situation continues and the values of total,count,num is still 0,15,6 respectively
- we hit "s" again and get to the point of
`count --`
- at this point, total=count/num, has been initiated and ended and hence we get a new value of total which 2 now, and other values remain unchanged.
- the value of count changes to 14 now at
`num --`
- the value of the num changed to 5 from 6 after overcoming situation of
`13 printf("Total is: %d\n", total);`

running one line through the program. Periodically print the value of the number until you see it reach zero. Once you realize what the issue with this code is type "q" to quit GDB. Then edit the source code

- So objective is to get a parameter to 0 based on the documentation
- variable total cannot go to 0 and shortest way is that num goes to 0
- and yes, num=0 now and the below snap shows how we got there
- just typing "s" takes us to the next line of code
- I just recollected that the initial code problem that came to my head was that we have mentioned int in the print statement, but actually that was right. The problem is that num goes to 0 and the num was supposed to be a divisor which going to 0 means that the quotient is infinity and that can never be an integer
- just something amazing thing that I learnt about the while loop is that though the condition taken into account for "while" has been count but still num was decremented everytime. Now I can visualise that this particular feature can be extended to other programming languages as well. Got to check something of the same in regards to forLoop.


```

Breakpoint 1, main () at sampleMath.c:9
9          while(count > 0) { /* Modify this line only */
(gdb) p num
$9 = 6
(gdb) break 9
Note: breakpoint 1 also set at pc 0x400543.
Breakpoint 2 at 0x400543: file sampleMath.c, line 9.
(gdb) p total
$10 = 0
(gdb) p count
$11 = 15
(gdb) s
10          total = count / num;
(gdb) p count
$12 = 15
(gdb) p num
$13 = 6
(gdb) p total
$14 = 0
(gdb) s
11          count--;
(gdb) p count
$15 = 15
(gdb) p total
$16 = 2
(gdb) p num
$17 = 6
(gdb) s
12          num--;
(gdb) p count
$18 = 14
(gdb) p total
$19 = 2
(gdb) p num
$20 = 6
(gdb) s
13          printf("Total is: %d\n", total);
(gdb) p num
$21 = 5
(gdb) p count
$22 = 14
(gdb) p total
$23 = 2
(gdb)

```

- Proceeding a bit further and we get an error, snap below, strangely total shows as 10; maybe that signifies some kind of high value. More can be understood later. Moreover, yes, the problem came up because num goes

to 0

```
Breakpoint 1, main () at sampleMath.c:9
9          while(count > 0) { /* Modify this line only */
(gdb) p num
$9 = 6
(gdb) break 9
Note: breakpoint 1 also set at pc 0x400543.
Breakpoint 2 at 0x400543: file sampleMath.c, line 9.
(gdb) p total
$10 = 0
(gdb) p count
$11 = 15
(gdb) s
10          total = count / num;
(gdb) p count
$12 = 15
(gdb) p num
$13 = 6
(gdb) p total
$14 = 0
(gdb) s
11          count--;
(gdb) p count
$15 = 15
(gdb) p total
$16 = 2
(gdb) p num
$17 = 6
(gdb) s
12          num--;
(gdb) p count
$18 = 14
(gdb) p total
$19 = 2
(gdb) p num
$20 = 6
(gdb) s
13          printf("Total is: %d\n", total);
(gdb) p num
$21 = 5
(gdb) p count
$22 = 14
(gdb) p total
$23 = 2
(gdb) 
```

- at the end we have num=0 total=10 which probably refers to infinity and count=9, which went 6 steps down

- there's nothing more from gdb that is required
- so as instructed in the document, the gdb session is closed
- now just did
 `nano sampleMath`
- and got the below, check snap
- probably that's the compiled version of c, which means that's just a machine readable code
- the below gives the power to read a file unlike "cd folderName" which just gives access to a folder
 `cat README`

```

Breakpoint 1, main () at sampleMath.c:9
9      while(count > 0) { /* Modify this line only */
(gdb) p num
$9 = 6
(gdb) break 9
Note: breakpoint 1 also set at pc 0x400543.
Breakpoint 2 at 0x400543: file sampleMath.c, line 9.
(gdb) p total
$10 = 0
(gdb) p count
$11 = 15
(gdb) s
10      total = count / num;
(gdb) p count
$12 = 15
(gdb) p num
$13 = 6
(gdb) p total
$14 = 0
(gdb) s
11      count--;
(gdb) p count
$15 = 15
(gdb) p total
$16 = 2
(gdb) p num
$17 = 6
(gdb) s
12      num--;
(gdb) p count
$18 = 14
(gdb) p total
$19 = 2
(gdb) p num
$20 = 6
(gdb) s
13      printf("Total is: %d\n", total);
(gdb) p num
$21 = 5
(gdb) p count
$22 = 14
(gdb) p total
$23 = 2
(gdb)

```

- below snap shows `disas main`, meaning disassemble main

```
(gdb) disas main
Dump of assembler code for function main:
0x000000000400526 <+0>:      push    %rbp
0x000000000400527 <+1>:      mov     %rsp,%rbp
0x00000000040052a <+4>:      sub     $0x10,%rsp
0x00000000040052e <+8>:      movl    $0x0,-0x4(%rbp)
0x000000000400535 <+15>:     movl    $0x6,-0xc(%rbp)
0x00000000040053c <+22>:     movl    $0xf,-0x8(%rbp)
0x000000000400543 <+29>:     jmp     0x40056b <main+69>
0x000000000400545 <+31>:     mov     -0x8(%rbp),%eax
0x000000000400548 <+34>:     cld
=> 0x000000000400549 <+35>:     idivl   -0xc(%rbp)
0x00000000040054c <+38>:     mov     %eax,-0x4(%rbp)
0x00000000040054f <+41>:     subl    $0x1,-0x8(%rbp)
0x000000000400553 <+45>:     subl    $0x1,-0xc(%rbp)
0x000000000400557 <+49>:     mov     -0x4(%rbp),%eax
0x00000000040055a <+52>:     mov     %eax,%esi
0x00000000040055c <+54>:     mov     $0x400604,%edi
0x000000000400561 <+59>:     mov     $0x0,%eax
0x000000000400566 <+64>:     callq   0x400400 <printf@plt>
0x00000000040056b <+69>:     cmpl    $0x0,-0x8(%rbp)
0x00000000040056f <+73>:     jg      0x400545 <main+31>
0x000000000400571 <+75>:     nop
0x000000000400572 <+76>:     leaveq
---Type <return> to continue, or q <return> to quit---
```

- quitting from the gdb session by "q"
- nano sampleMath.c
- we did the above because probably we wanna see the code and not the compiled stuff
- we attempted to see the compiled file by doing "nano sampleMath" and got the result which shows the machine readable code
- simple used the nano editor's cursor and changed the code to num=0

```
GNU nano 2.5.3                               File: sampleMath.c                               Modified

#include <stdio.h>
void main() {
    int num;
    int count;
    int total;
    total = 0;
    num = 6;
    count = 15;
    while(num > 0) { /* Modify this line only */
        total = count / num;
        count--;
        num--;
        printf("Total is: %d\n", total);
    }
}
```

- chose the option
^o which means ctrl+o for writing

- got the option to save the name differently and hence chose sampleMathedited.c
- would be checking if the file can be compiled successfully or not

```
ubuntu@gdblesson:~$ less sampleMathedited.c
ubuntu@gdblesson:~$ gcc -g sampleMathedited.c -o sampleMathedited
ubuntu@gdblesson:~$ ./sampleMathedited
Total is: 2
Total is: 2
Total is: 3
Total is: 4
Total is: 5
Total is: 10
ubuntu@gdblesson:~$
```

- opened the second program, sampleMath2.c and found the below code

```
#include <stdio.h>
#include <stdlib.h> /*h is the header of the general purpose standard
library of C programming language which includes functions involving
memory allocation , process control , conversions and others.
It is compatible with C++ and is known as cstdlib in C++.
The name "stdlib" stands for "standard_library". */
void main(int argc, char *argv[]) { /*taking up a integer variable argc and
the declaration char *argv[] is an array (of undetermined size)
of pointers to char , in other words an array of strings */
    int total; /*we initialise integer variables total,n,i */
    int n;
    int i;
    /* Your edit goes below */

    /* Add line above */
    if(argc > 1) { /* if the input gotten is more than 1*/
        i = atoi(argv[1]); /*Convert string to integer, atoi */
    }
    else {
        printf("You must provide one integer argument greater than 0.\n");
        i = -1;
    }
    for(n = 0; n <= i; n++) {
        if(n % 2 == 0){
            total += (n + n + 1) * n;
        }
        else{
            total -= (n + n + 1) * n;
        }
    }
    total = abs(total);
```

```

    printf("The value of 1 should be 3.\nThe value of 2 should be 7.\nThe va
}

```

- check the snapshot of the program below

```

#include <stdio.h>
#include <stdlib.h>
void main(int argc, char *argv[]) {
    int total;
    int n;
    int i;
    /* Your edit goes below */

    /* Add line above */
    if(argc > 1) {
        i = atoi(argv[1]);
    }
    else {
        printf("You must provide one integer argument greater than 0.\n");
        i = -1;
    }
    for(n = 0; n <= i; n++) {
        if(n % 2 == 0){
            total += (n + n + 1) * n;
        }
        else{
            total -= (n + n + 1) * n;
        }
    }
    total = abs(total);
    printf("The value of 1 should be 3.\nThe value of 2 should be 7.\nThe value of 3 should be 14.\n
The value of 4 should be 22.\nYour total is: %d\n", total);
}

```

- let's break up the program for sampleMath2.c
- in order to breakup the program let's create a file named sampleMath2breakup.c
- gedit sampleMath2breakup.c [1]
- the above is not working, if I remember it worked in a regular download folder
- breakup code wasn't working because am still not well-versed with C
- however if we compile the sampleMath2.c and then run by ./sampleMath2, we get the following

```

ubuntu@gdblesson:~$ ./sampleMath2
You must provide one integer argument greater than 0.
The value of 1 should be 3.
The value of 2 should be 7.
The value of 3 should be 14.
The value of 4 should be 22.
Your total is: 32766

```

- if we give an argument to the program like below

```
sampleMath2.c 1
```

```
ubuntu@gdblesson:~$ ./sampleMath2 1
The value of 1 should be 3.
The value of 2 should be 7.
The value of 3 should be 14.
The value of 4 should be 22.
Your total is: 32761
ubuntu@gdblesson:~$
```

4. I have compile and run the provided **sampleMath2.c** with one integer argument and saw the given and expected outputs are different. I run the **gdb** and printed the total value. At the beginning total is 32767. So I have understood the issue. At the beginning total should be zero. But in this program we have not defined total. It automatically takes the largest integer it can be saved.

```
ubuntu@gdblesson: ~
File Edit View Search Terminal Help
Your total is: 32774
[Inferior 1 (process 1553) exited with code 0207]
(gdb) b 4
Breakpoint 1 at 0x4005c5: file sampleMath2.c, line 4.
(gdb) r
Starting program: /hone/ubuntu/sampleMath2 2

Breakpoint 1, main (argc=2, argv=0x7fffffff678) at sampleMath2.c:10
10      if(argc > 1) {
(gdb) p total
$1 = 32767
(gdb)
```

5. After I debug and found out the issue I fixed the code by defining the value of total as zero. Again compiled and run the program it showed the expected output.

```
ubuntu@gdblesson: ~
File Edit View Search Terminal Help
GNU nano 2.5.3      File: sampleMath2.c      Modified

#include <stdio.h>
#include <stdlib.h>
void main(int argc, char *argv[]) {
    int total;
    int n;
    int i;
    /* Your edit goes below */
    total = 0;
    /* Add line above */
    if(argc > 1) {
        i = atoi(argv[1]);
    }
    else {
        printf("You must provide one integer argument greater than 0.\n");
    }
}
```

References

[1] <https://vitux.com/how-to-write-and-run-a-c-program-in-linux/>.