ATTACK DEFENSE

by PentesterAcademy

Name	Examining Source Files
URL	https://www.attackdefense.com/challengedetails?cid=2169
Type	Reverse Engineering : GDB Basics

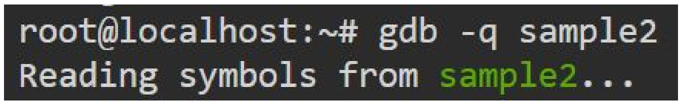
Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Objective: Learn how to examine the source files in GDB and check out different commands/options/methods.

Solution:

Step 1: Open sample2 binary using gdb.

Command: gdb -q sample2



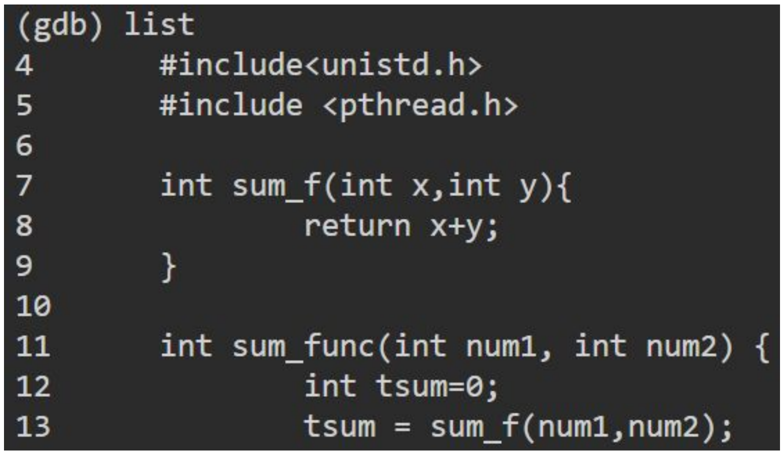
```
root@localhost:~# gdb -q sample2
Reading symbols from sample2...
```

<u>Listing source code lines</u>

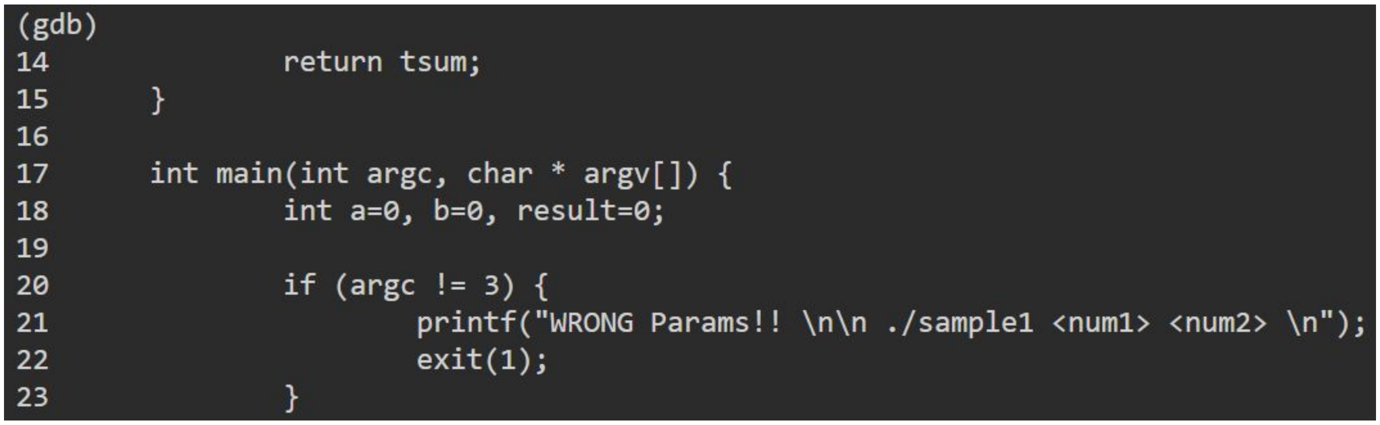
Step 2: Check the source code in steps.

Command: list

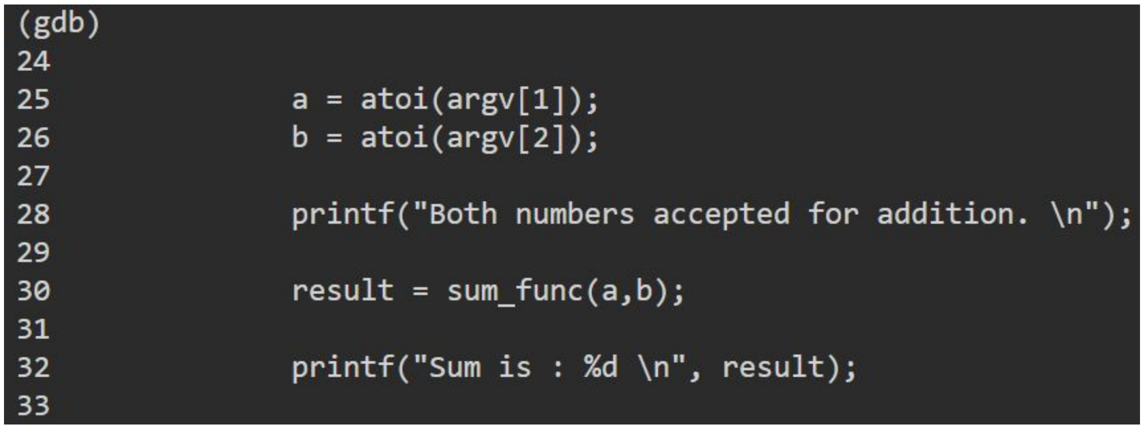
This will print x lines of the source code file. On each subsequent run of this command, the next x lines are printed.



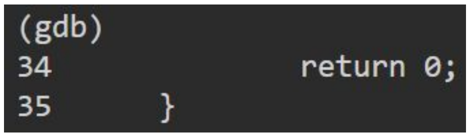
```
(gdb) list
4       #include<unistd.h>
5       #include <pthread.h>
6
7       int sum_f(int x,int y){
8               return x+y;
9       }
10
11      int sum_func(int num1, int num2) {
12              int tsum=0;
13              tsum = sum_f(num1,num2);
```



```
(gdb)
14              return tsum;
15      }
16
17      int main(int argc, char * argv[]) {
18              int a=0, b=0, result=0;
19
20              if (argc != 3) {
21                      printf("WRONG Params!! \n\n ./sample1 <num1> <num2> \n");
22                      exit(1);
23              }
```



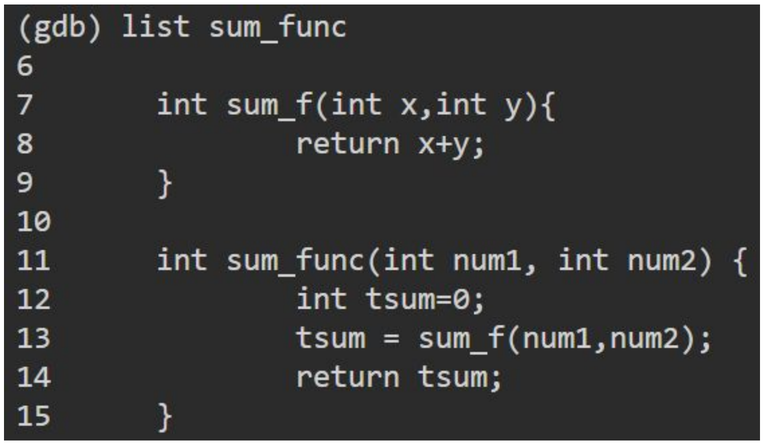
```
(gdb)
24
25              a = atoi(argv[1]);
26              b = atoi(argv[2]);
27
28              printf("Both numbers accepted for addition. \n");
29
30              result = sum_func(a,b);
31
32              printf("Sum is : %d \n", result);
33
```



```
(gdb)
34              return 0;
35      }
```

Step 3: One can print source code lines for a function. Please note that the list always tries to print a few lines before the requested section.

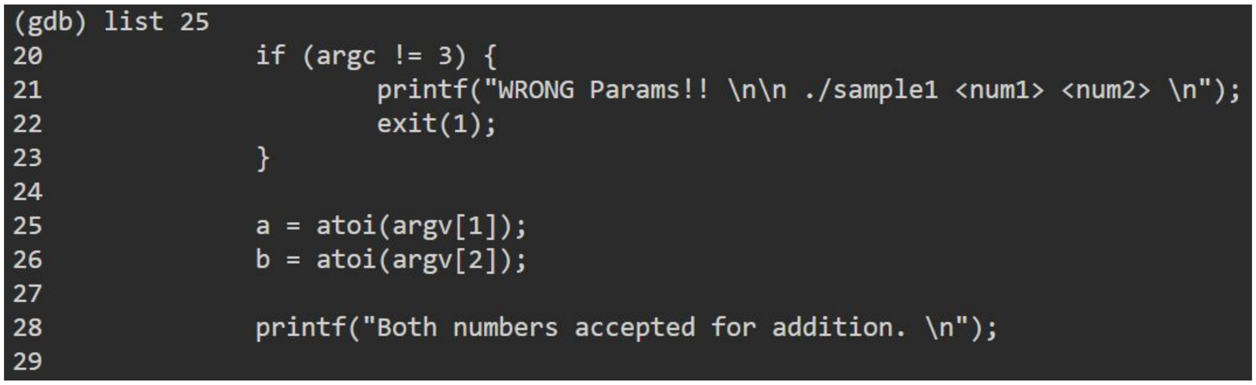
Command: list sum_func



```
(gdb) list sum_func
6
7       int sum_f(int x,int y){
8               return x+y;
9       }
10
11      int sum_func(int num1, int num2) {
12              int tsum=0;
13              tsum = sum_f(num1,num2);
14              return tsum;
15      }
```

Step 4: One can print source code lines around a specific line number. Please note that the list command will print it in such a way that the requested line comes in the middle of the output.

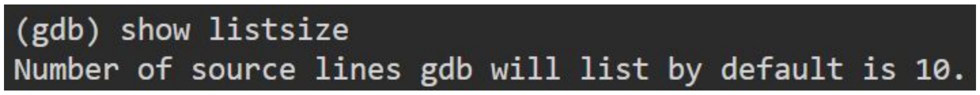
Command: list 25



```
(gdb) list 25
20              if (argc != 3) {
21                      printf("WRONG Params!! \n\n ./sample1 <num1> <num2> \n");
22                      exit(1);
23              }
24
25              a = atoi(argv[1]);
26              b = atoi(argv[2]);
27
28              printf("Both numbers accepted for addition. \n");
29
```

Step 5: The number of lines to be printed for one usage of list command, are defined by listsize variable. Check the current listsize.

Command: show listsize

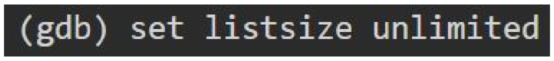


```
(gdb) show listsize
Number of source lines gdb will list by default is 10.
```

The current setting is to print 10 lines of code on each use of list command.

Step 6: Change the line limit from 10 to unlimited.

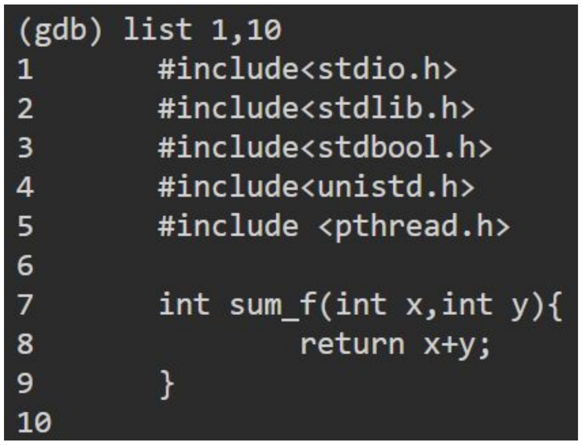
Command: set listsize unlimited



```
(gdb) set listsize unlimited
```

Step 7: Code lines can also be listed by providing line number range.

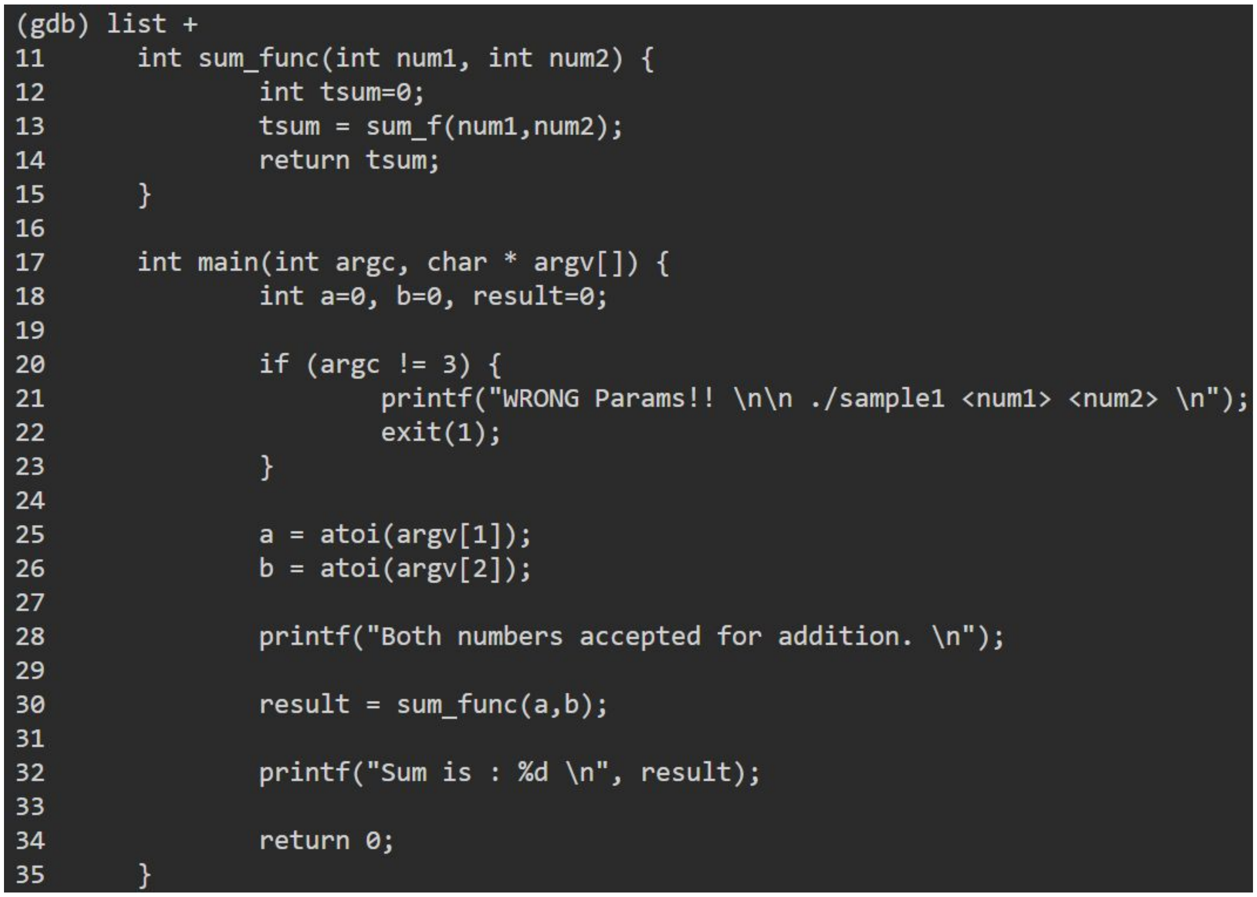
Command: list 1,10



```
(gdb) list 1,10
1       #include<stdio.h>
2       #include<stdlib.h>
3       #include<stdbool.h>
4       #include<unistd.h>
5       #include <pthread.h>
6
7       int sum_f(int x,int y){
8               return x+y;
9       }
10
```

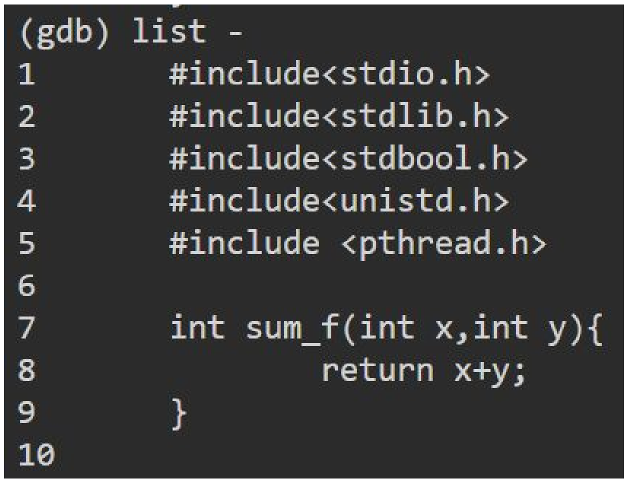
Step 8: List also takes + and - arguments. On passing + argument, it prints next lines till the end of program (because linesize is set to unlimited)

Command: list +


```
(gdb) list +
11      int sum_func(int num1, int num2) {
12              int tsum=0;
13              tsum = sum_f(num1,num2);
14              return tsum;
15      }
16
17      int main(int argc, char * argv[]) {
18              int a=0, b=0, result=0;
19
20              if (argc != 3) {
21                      printf("WRONG Params!! \n\n ./sample1 <num1> <num2> \n");
22                      exit(1);
23              }
24
25              a = atoi(argv[1]);
26              b = atoi(argv[2]);
27
28              printf("Both numbers accepted for addition. \n");
29
30              result = sum_func(a,b);
31
32              printf("Sum is : %d \n", result);
33
34              return 0;
35      }
```

Step 9: On passing - argument, it prints previous lines till the start of program (because linesize is set to unlimited)

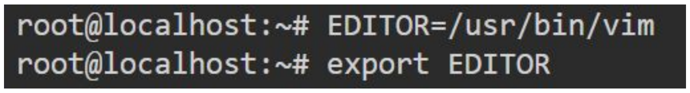
Command: list -


```
(gdb) list -
1       #include<stdio.h>
2       #include<stdlib.h>
3       #include<stdbool.h>
4       #include<unistd.h>
5       #include <pthread.h>
6
7       int sum_f(int x,int y){
8               return x+y;
9       }
10
```

Editing source code

GDB allows the user to edit the source file from inside the GDB by calling a text editor. By default, the binary /bin/ex which is non-existent. So, first the user has to map it to a valid text editor.

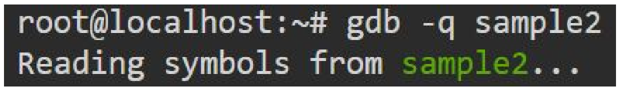
Commands:
EDITOR=/usr/bin/vim
Export EDITOR



```
root@localhost:~# EDITOR=/usr/bin/vim
root@localhost:~# export EDITOR
```

Step 10: Start the GDB with sample2 binary.

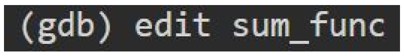
Command: gdb -q sample2



```
root@localhost:~# gdb -q sample2
Reading symbols from sample2...
```

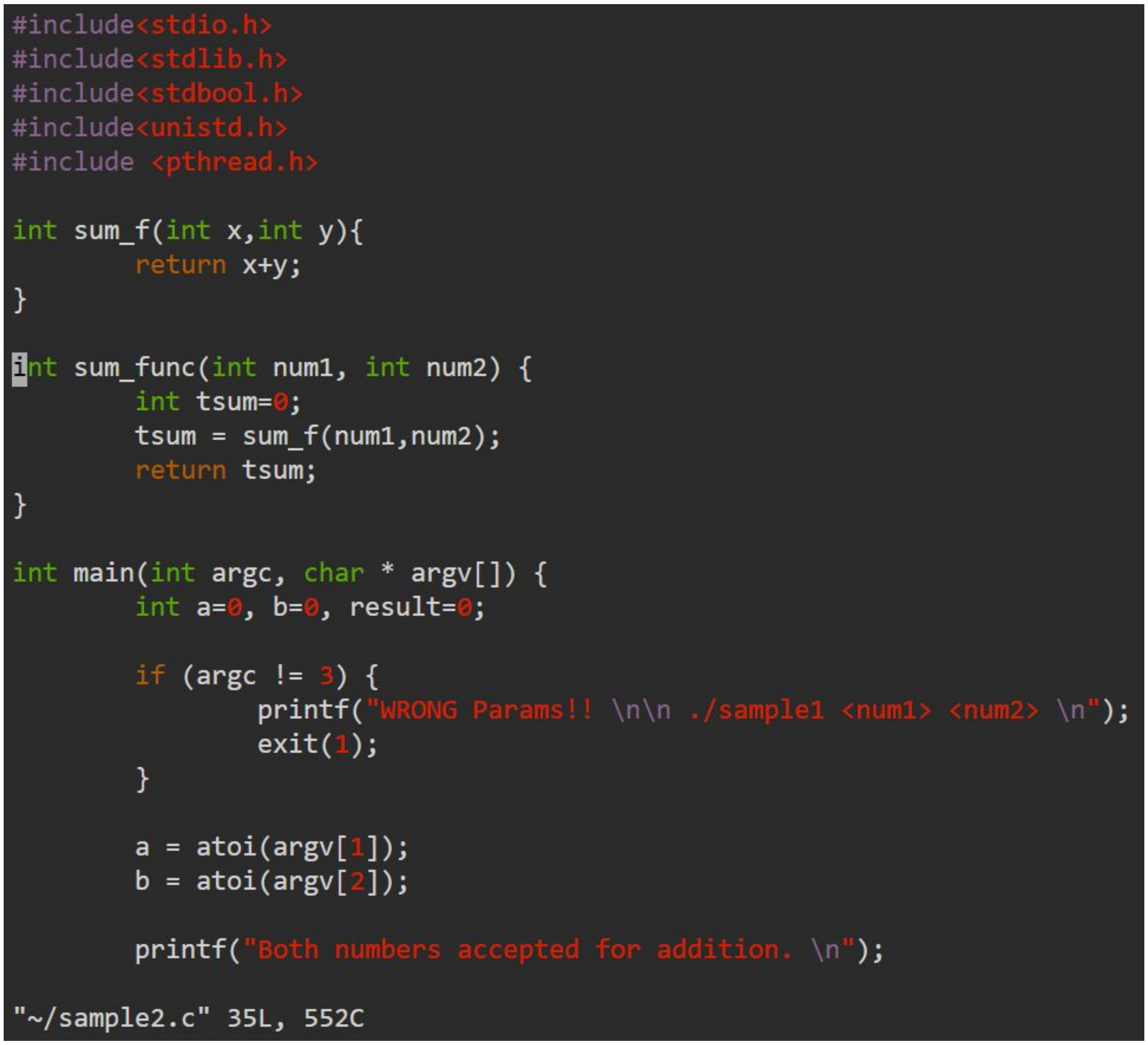
Step 11: Edit the source file at function sum_func

Command: edit sum_func



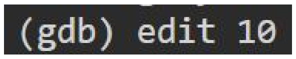
```
(gdb) edit sum_func
```

It will open the source file in the vim text editor with an edit pointer on the definition of function"sum_func".

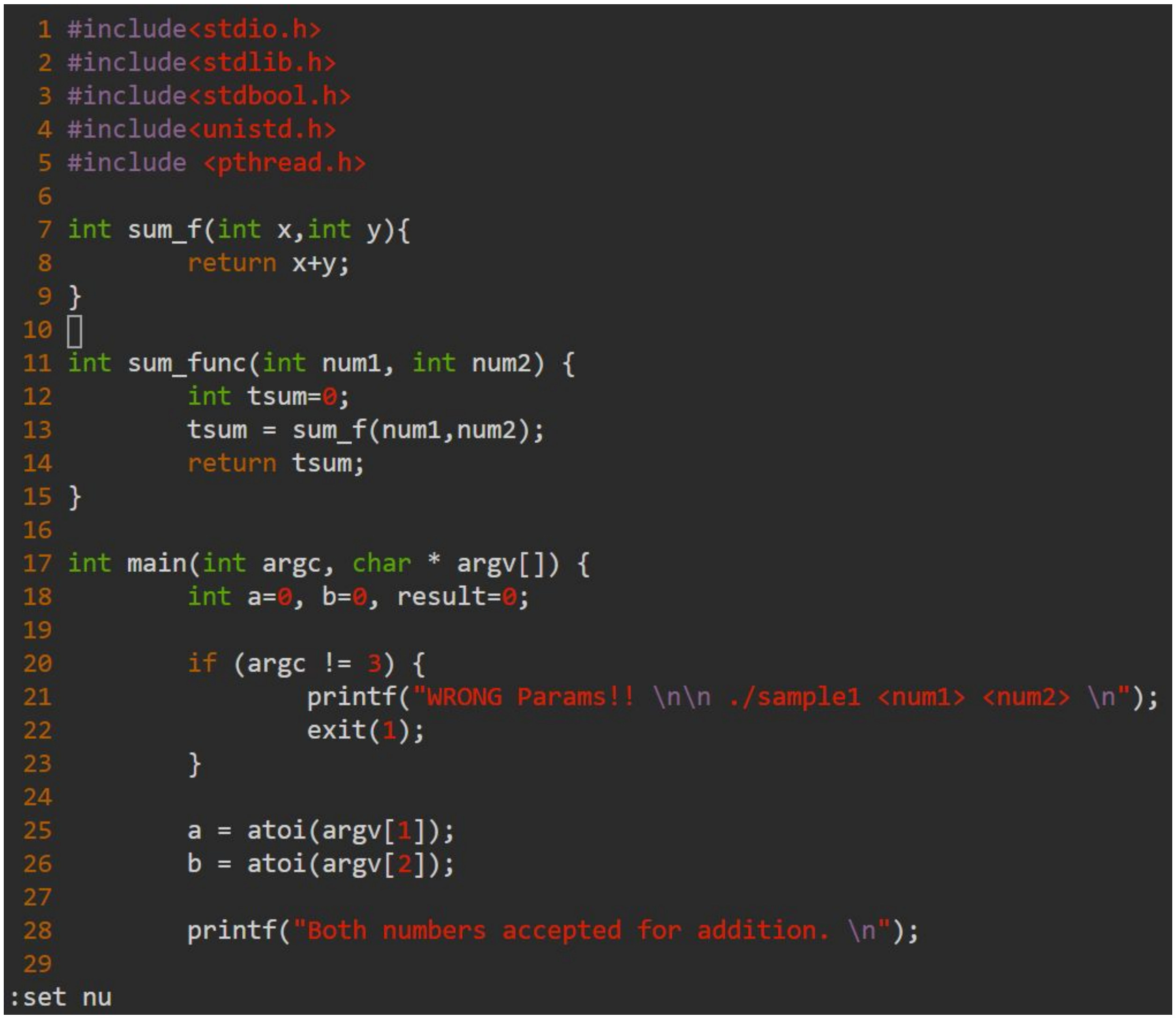

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<unistd.h>
#include <pthread.h>

int sum_f(int x,int y){
        return x+y;
}

int sum_func(int num1, int num2) {
        int tsum=0;
        tsum = sum_f(num1,num2);
        return tsum;
}

int main(int argc, char * argv[]) {
        int a=0, b=0, result=0;

        if (argc != 3) {
                printf("WRONG Params!! \n\n ./sample1 <num1> <num2> \n");
                exit(1);
        }

        a = atoi(argv[1]);
        b = atoi(argv[2]);

        printf("Both numbers accepted for addition. \n");

"~/sample2.c" 35L, 552C
```

Step 12: Edit the source file at line 10

Command: edit 10


```
(gdb) edit 10
```

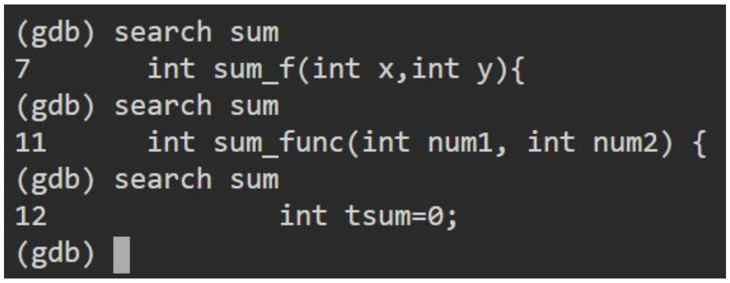
It will open the source file in the vim text editor with an edit pointer on line number 10.



```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<unistd.h>
#include <pthread.h>

int sum_f(int x,int y){
        return x+y;
}

int sum_func(int num1, int num2) {
        int tsum=0;
        tsum = sum_f(num1,num2);
        return tsum;
}

int main(int argc, char * argv[]) {
        int a=0, b=0, result=0;

        if (argc != 3) {
                printf("WRONG Params!! \n\n ./sample1 <num1> <num2> \n");
                exit(1);
        }

        a = atoi(argv[1]);
        b = atoi(argv[2]);

        printf("Both numbers accepted for addition. \n");

:set nu
```

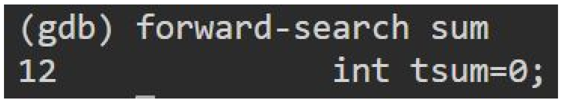
Step 12: The search command can be used to find the string/substring in the source code. On every run, the scope of search moves forward.

Command: search sum


```
(gdb) search sum
7       int sum_f(int x,int y){
(gdb) search sum
11      int sum_func(int num1, int num2) {
(gdb) search sum
12              int tsum=0;
(gdb) 
```

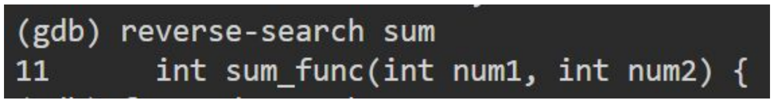
The same can be done using forward-search command.

Command: forward-search sum


```
(gdb) forward-search sum
12              int tsum=0;
```

Step 13: The reverse-search command can be used to find the string/substring in the source code in the reverse direction.

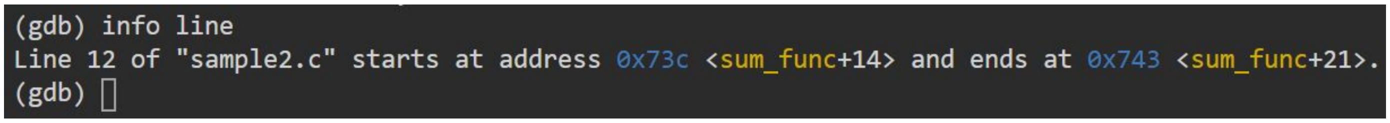
Command: reverse-search sum


```
(gdb) reverse-search sum
11      int sum_func(int num1, int num2) {
```

Source and Machine Code

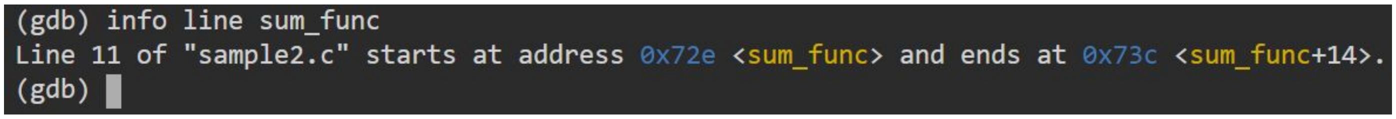
Step 13: The information (e.g. address) about the current line can be viewed.

Command: info line


```
(gdb) info line
Line 12 of "sample2.c" starts at address 0x73c <sum_func+14> and ends at 0x743 <sum_func+21>.
(gdb) 
```

Step 14: The information (e.g. address) about a line can also be viewed.by passing the name of the function to it.

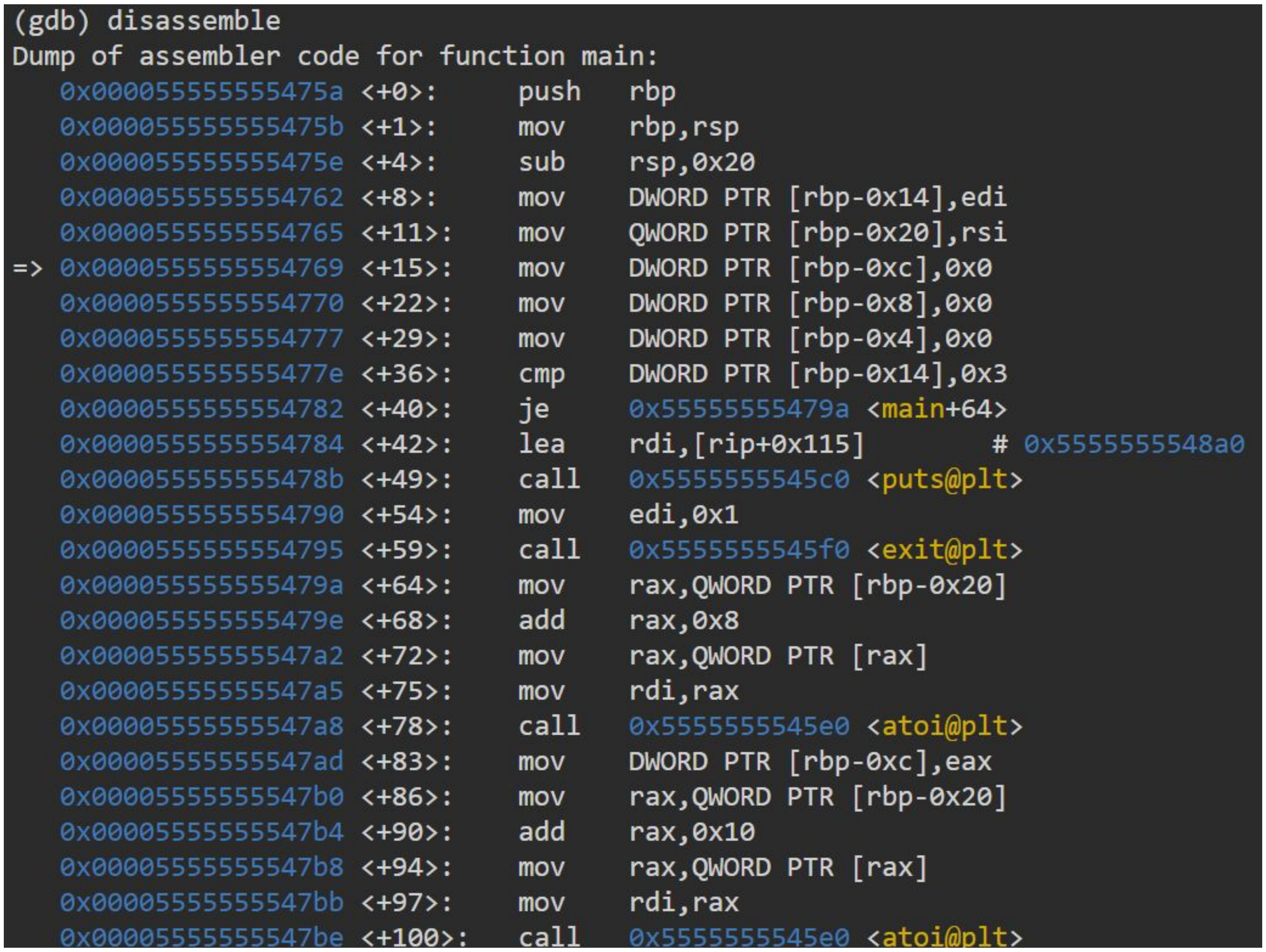
Command: info line sum_func



```
(gdb) info line sum_func
Line 11 of "sample2.c" starts at address 0x72e <sum_func> and ends at 0x73c <sum_func+14>.
(gdb) 
```

Step 13: GDB also allows the user to dump a range of memory as machine instructions for the program/binary.

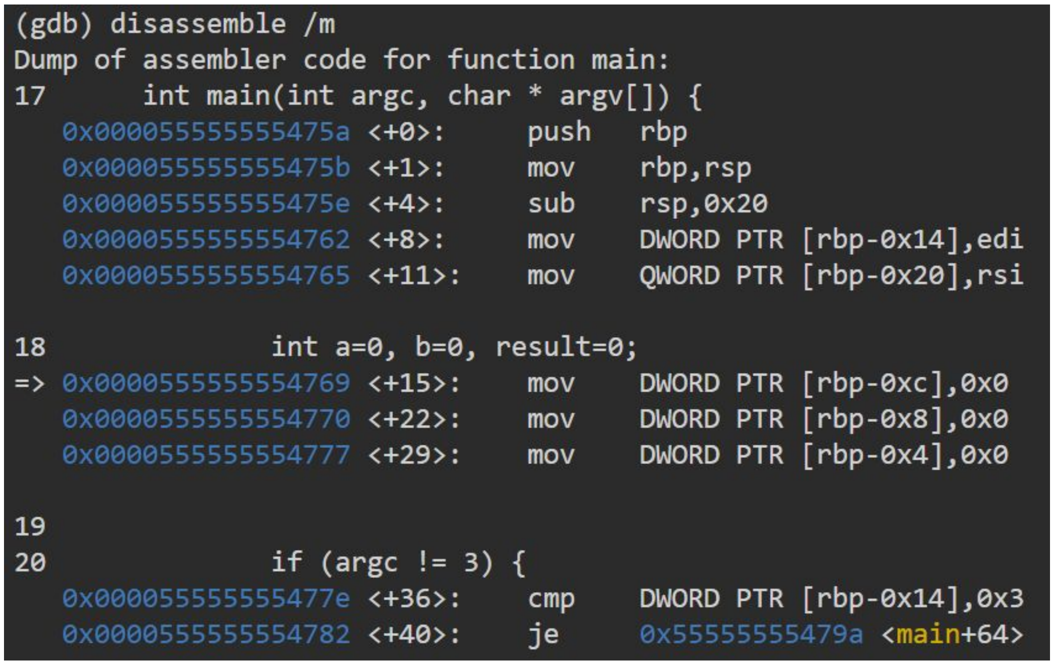
Command: disassemble or **Command:** disas



```
(gdb) disassemble
Dump of assembler code for function main:
   0x000055555555475a <+0>:     push   rbp
   0x000055555555475b <+1>:     mov    rbp,rsp
   0x000055555555475e <+4>:     sub    rsp,0x20
   0x0000555555554762 <+8>:     mov    DWORD PTR [rbp-0x14],edi
   0x0000555555554765 <+11>:    mov    QWORD PTR [rbp-0x20],rsi
=> 0x0000555555554769 <+15>:    mov    DWORD PTR [rbp-0xc],0x0
   0x0000555555554770 <+22>:    mov    DWORD PTR [rbp-0x8],0x0
   0x0000555555554777 <+29>:    mov    DWORD PTR [rbp-0x4],0x0
   0x000055555555477e <+36>:    cmp    DWORD PTR [rbp-0x14],0x3
   0x0000555555554782 <+40>:    je     0x55555555479a <main+64>
   0x0000555555554784 <+42>:    lea    rdi,[rip+0x115]        # 0x5555555548a0
   0x000055555555478b <+49>:    call   0x5555555545c0 <puts@plt>
   0x0000555555554790 <+54>:    mov    edi,0x1
   0x0000555555554795 <+59>:    call   0x5555555545f0 <exit@plt>
   0x000055555555479a <+64>:    mov    rax,QWORD PTR [rbp-0x20]
   0x000055555555479e <+68>:    add    rax,0x8
   0x00005555555547a2 <+72>:    mov    rax,QWORD PTR [rax]
   0x00005555555547a5 <+75>:    mov    rdi,rax
   0x00005555555547a8 <+78>:    call   0x5555555545e0 <atoi@plt>
   0x00005555555547ad <+83>:    mov    DWORD PTR [rbp-0xc],eax
   0x00005555555547b0 <+86>:    mov    rax,QWORD PTR [rbp-0x20]
   0x00005555555547b4 <+90>:    add    rax,0x10
   0x00005555555547b8 <+94>:    mov    rax,QWORD PTR [rax]
   0x00005555555547bb <+97>:    mov    rdi,rax
   0x00005555555547be <+100>:   call   0x5555555545e0 <atoi@plt>
```

Step 14: The assembly code and source code can be printed in mixed form by defining /s or /m

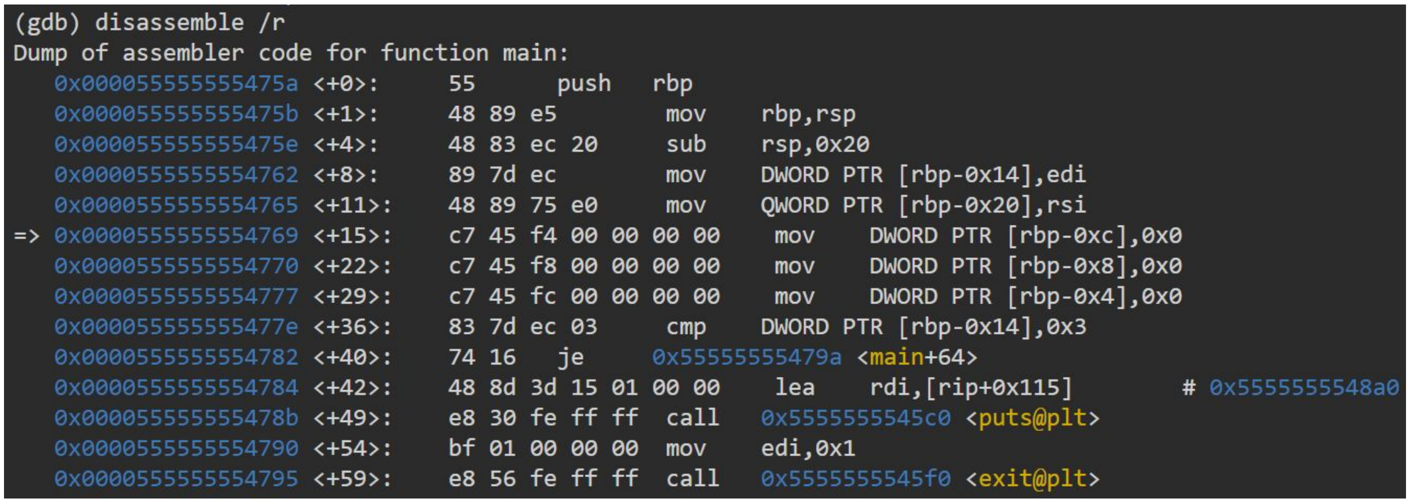
Command: disassemble /s or **Command:** disassemble /m



```
(gdb) disassemble /m
Dump of assembler code for function main:
17      int main(int argc, char * argv[]) {
   0x000055555555475a <+0>:     push   rbp
   0x000055555555475b <+1>:     mov    rbp,rsp
   0x000055555555475e <+4>:     sub    rsp,0x20
   0x0000555555554762 <+8>:     mov    DWORD PTR [rbp-0x14],edi
   0x0000555555554765 <+11>:    mov    QWORD PTR [rbp-0x20],rsi

18              int a=0, b=0, result=0;
=> 0x0000555555554769 <+15>:    mov    DWORD PTR [rbp-0xc],0x0
   0x0000555555554770 <+22>:    mov    DWORD PTR [rbp-0x8],0x0
   0x0000555555554777 <+29>:    mov    DWORD PTR [rbp-0x4],0x0

19
20              if (argc != 3) {
   0x000055555555477e <+36>:    cmp    DWORD PTR [rbp-0x14],0x3
   0x0000555555554782 <+40>:    je     0x55555555479a <main+64>
```

Step 15: Similarly, to print raw instructions in hex as well as in symbolic form, /r needs to be specified.

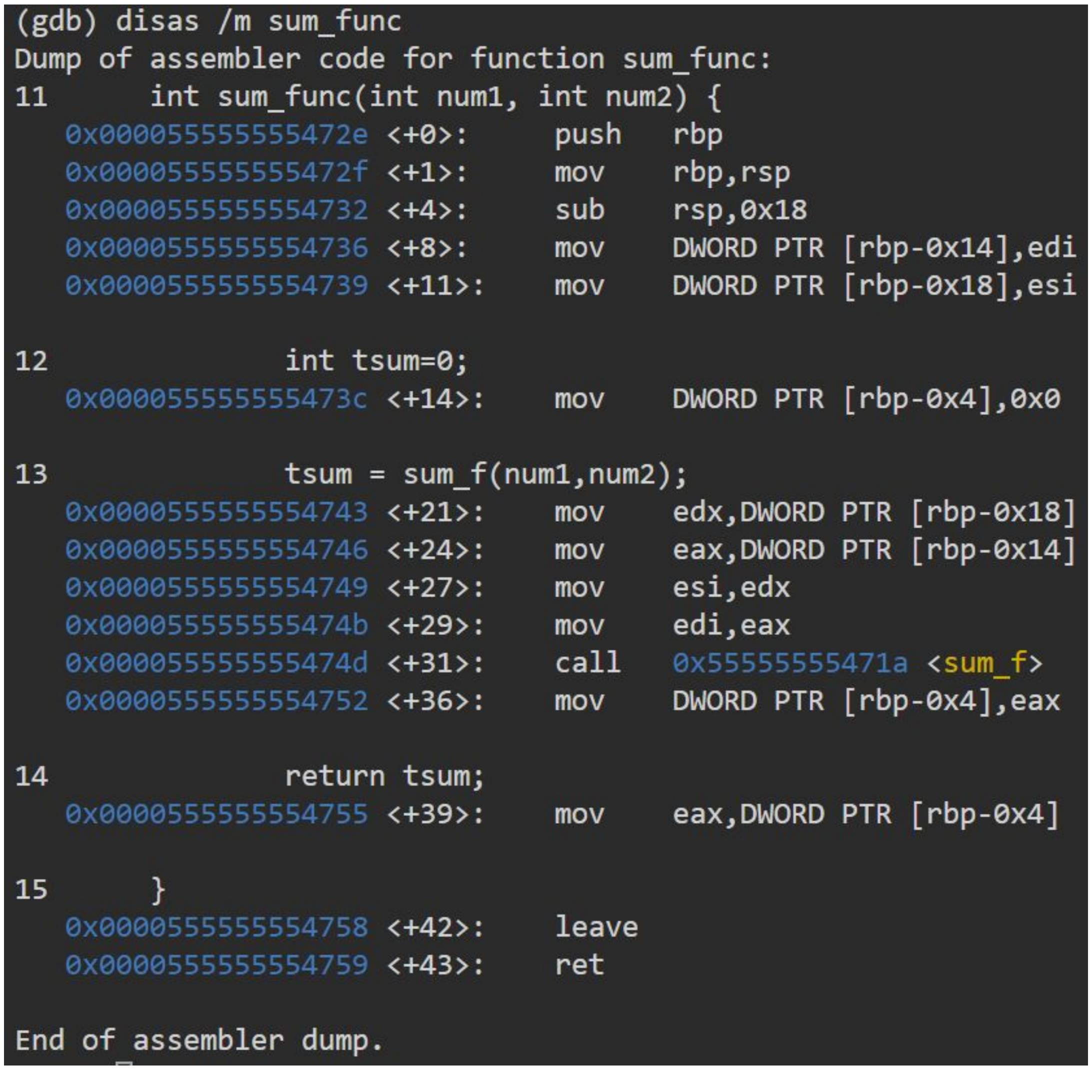
Command: disassemble /r



```
(gdb) disassemble /r
Dump of assembler code for function main:
   0x000055555555475a <+0>:     55      push   rbp
   0x000055555555475b <+1>:     48 89 e5        mov    rbp,rsp
   0x000055555555475e <+4>:     48 83 ec 20     sub    rsp,0x20
   0x0000555555554762 <+8>:     89 7d ec        mov    DWORD PTR [rbp-0x14],edi
   0x0000555555554765 <+11>:    48 89 75 e0     mov    QWORD PTR [rbp-0x20],rsi
=> 0x0000555555554769 <+15>:    c7 45 f4 00 00 00 00    mov    DWORD PTR [rbp-0xc],0x0
   0x0000555555554770 <+22>:    c7 45 f8 00 00 00 00    mov    DWORD PTR [rbp-0x8],0x0
   0x0000555555554777 <+29>:    c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0
   0x000055555555477e <+36>:    83 7d ec 03     cmp    DWORD PTR [rbp-0x14],0x3
   0x0000555555554782 <+40>:    74 16   je     0x55555555479a <main+64>
   0x0000555555554784 <+42>:    48 8d 3d 15 01 00 00    lea    rdi,[rip+0x115]        # 0x5555555548a0
   0x000055555555478b <+49>:    e8 30 fe ff ff  call   0x5555555545c0 <puts@plt>
   0x0000555555554790 <+54>:    bf 01 00 00 00  mov    edi,0x1
   0x0000555555554795 <+59>:    e8 56 fe ff ff  call   0x5555555545f0 <exit@plt>
```

Step 16: One can also see the assembly code for a specific function.

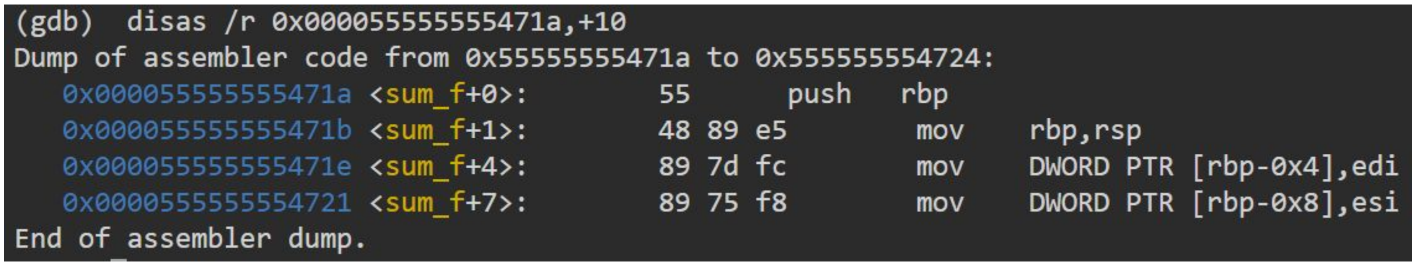
Command: disassemble /m sum_func



```
(gdb) disas /m sum_func
Dump of assembler code for function sum_func:
11      int sum_func(int num1, int num2) {
   0x000055555555472e <+0>:     push   rbp
   0x000055555555472f <+1>:     mov    rbp,rsp
   0x0000555555554732 <+4>:     sub    rsp,0x18
   0x0000555555554736 <+8>:     mov    DWORD PTR [rbp-0x14],edi
   0x0000555555554739 <+11>:    mov    DWORD PTR [rbp-0x18],esi

12              int tsum=0;
   0x000055555555473c <+14>:    mov    DWORD PTR [rbp-0x4],0x0

13              tsum = sum_f(num1,num2);
   0x0000555555554743 <+21>:    mov    edx,DWORD PTR [rbp-0x18]
   0x0000555555554746 <+24>:    mov    eax,DWORD PTR [rbp-0x14]
   0x0000555555554749 <+27>:    mov    esi,edx
   0x000055555555474b <+29>:    mov    edi,eax
   0x000055555555474d <+31>:    call   0x55555555471a <sum_f>
   0x0000555555554752 <+36>:    mov    DWORD PTR [rbp-0x4],eax

14              return tsum;
   0x0000555555554755 <+39>:    mov    eax,DWORD PTR [rbp-0x4]

15      }
   0x0000555555554758 <+42>:    leave
   0x0000555555554759 <+43>:    ret

End of assembler dump.
```

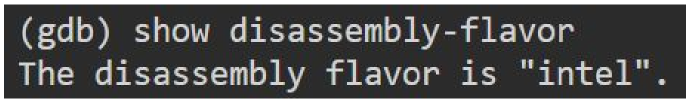
Step 17: Another way is to pass the address and offset.

Command: disas /r 0x000055555555471a,+10


```
(gdb)  disas /r 0x000055555555471a,+10
Dump of assembler code from 0x55555555471a to 0x555555554724:
   0x000055555555471a <sum_f+0>:        55      push   rbp
   0x000055555555471b <sum_f+1>:        48 89 e5        mov    rbp,rsp
   0x000055555555471e <sum_f+4>:        89 7d fc        mov    DWORD PTR [rbp-0x4],edi
   0x0000555555554721 <sum_f+7>:        89 75 f8        mov    DWORD PTR [rbp-0x8],esi
End of assembler dump.
```

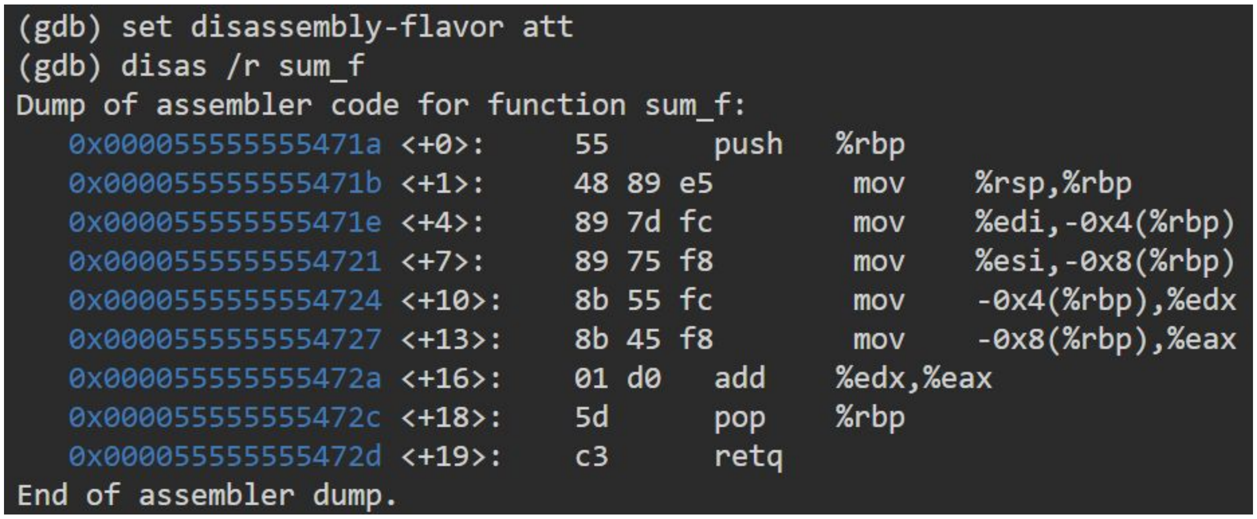
Step 18: Check the disassembly-flavor set for the GDB instance.

Command: show disassembly-flavour


```
(gdb) show disassembly-flavor
The disassembly flavor is "intel".
```

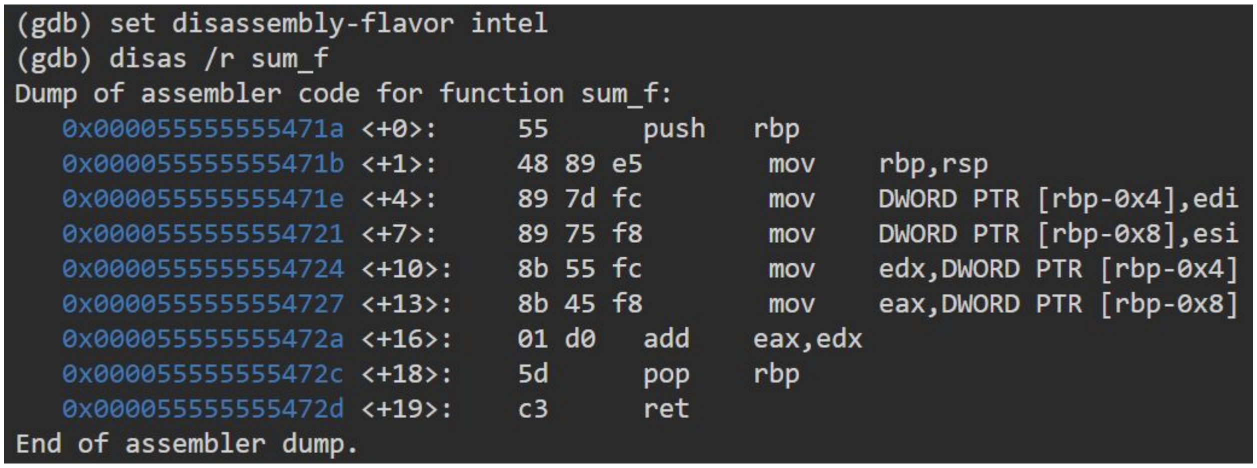
Step 19: Change disassembly-flavor to att and print assembly representation of sum_f() function.

Commands:
set disassembly-flavor att
disas /r sum_f


```
(gdb) set disassembly-flavor att
(gdb) disas /r sum_f
Dump of assembler code for function sum_f:
   0x000055555555471a <+0>:     55      push   %rbp
   0x000055555555471b <+1>:     48 89 e5        mov    %rsp,%rbp
   0x000055555555471e <+4>:     89 7d fc        mov    %edi,-0x4(%rbp)
   0x0000555555554721 <+7>:     89 75 f8        mov    %esi,-0x8(%rbp)
   0x0000555555554724 <+10>:    8b 55 fc        mov    -0x4(%rbp),%edx
   0x0000555555554727 <+13>:    8b 45 f8        mov    -0x8(%rbp),%eax
   0x000055555555472a <+16>:    01 d0   add    %edx,%eax
   0x000055555555472c <+18>:    5d      pop    %rbp
   0x000055555555472d <+19>:    c3      retq
End of assembler dump.
```

Step 20: Change disassembly-flavor to intel and print assembly representation of sum_f() function.

Commands:
set disassembly-flavor intel
disas /r sum_f



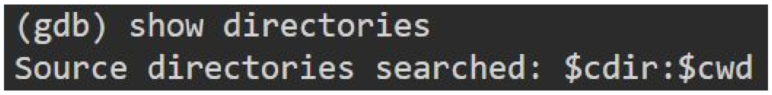
```
(gdb) set disassembly-flavor intel
(gdb) disas /r sum_f
Dump of assembler code for function sum_f:
   0x000055555555471a <+0>:     55      push   rbp
   0x000055555555471b <+1>:     48 89 e5        mov    rbp,rsp
   0x000055555555471e <+4>:     89 7d fc        mov    DWORD PTR [rbp-0x4],edi
   0x0000555555554721 <+7>:     89 75 f8        mov    DWORD PTR [rbp-0x8],esi
   0x0000555555554724 <+10>:    8b 55 fc        mov    edx,DWORD PTR [rbp-0x4]
   0x0000555555554727 <+13>:    8b 45 f8        mov    eax,DWORD PTR [rbp-0x8]
   0x000055555555472a <+16>:    01 d0   add    eax,edx
   0x000055555555472c <+18>:    5d      pop    rbp
   0x000055555555472d <+19>:    c3      ret
End of assembler dump.
```

Now both outputs can be compared to see the difference among two styles.

<u>Specifying Source Directories</u>

Step 21: Check the defined source directories (directories in which GDB should look for source files).

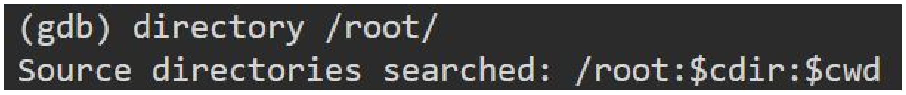
Command: show directories



```
(gdb) show directories
Source directories searched: $cdir:$cwd
```

Step 22: Add directory /root/ to the source directory.

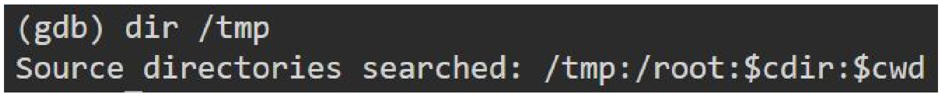
Command: directory /root/



```
(gdb) directory /root/
Source directories searched: /root:$cdir:$cwd
```

Alternatively, the short form of directory i.e. dir, can be used.

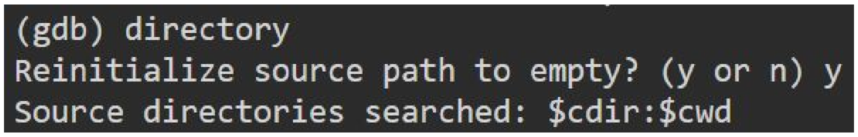
Command: dir /tmp/



```
(gdb) dir /tmp
Source directories searched: /tmp:/root:$cdir:$cwd
```

Step 23: Reset the source directories.

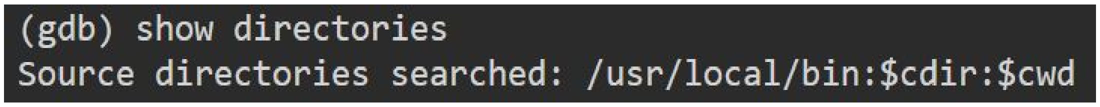
Command: directory



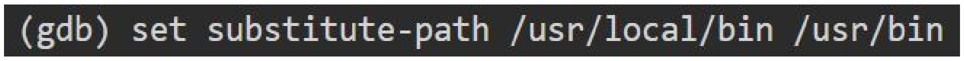
```
(gdb) directory
Reinitialize source path to empty? (y or n) y
Source directories searched: $cdir:$cwd
```

Step 24: Check the defined source directories and substitute the path /usr/local/bin with /usr/bin

Commands:
show directories
set substitute-path /usr/local/bin /usr/bin



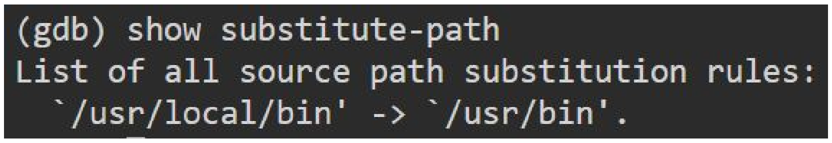
```
(gdb) show directories
Source directories searched: /usr/local/bin:$cdir:$cwd
```



```
(gdb) set substitute-path /usr/local/bin /usr/bin
```

Step 25: The substituted path can be viewed to confirm the change.

Command: show substitute-path



```
(gdb) show substitute-path
List of all source path substitution rules:
  `/usr/local/bin' -> `/usr/bin'.
```

References:

1. GDB Documentation (https://sourceware.org/gdb/current/onlinedocs/gdb)