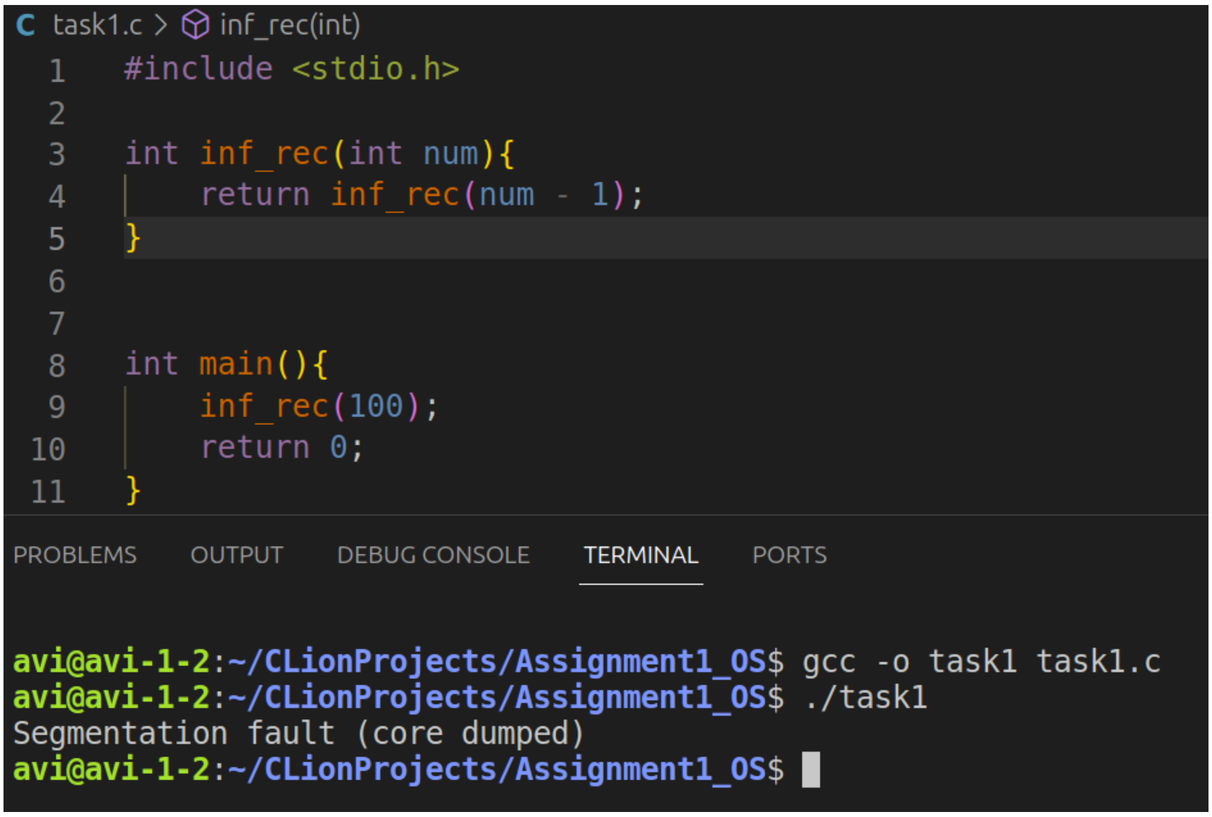
Assignment 1 - OS

task 1:

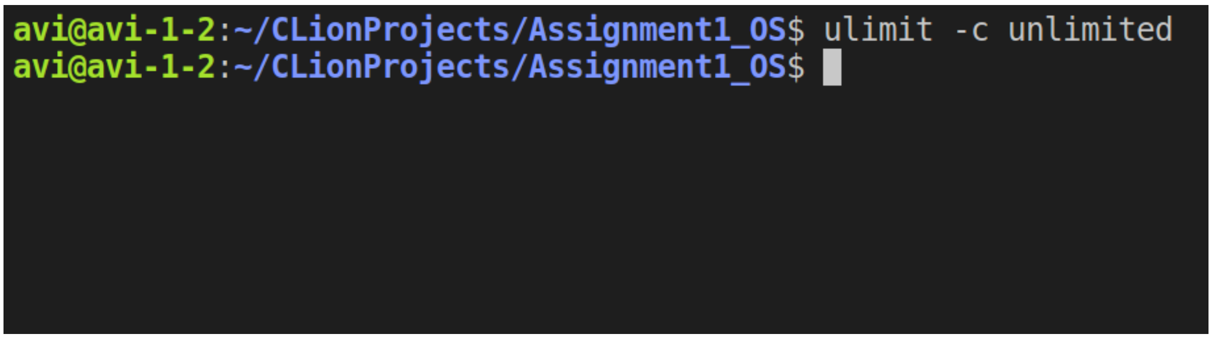
compile the task without debugging leads to this result:



```c
#include <stdio.h>

int inf_rec(int num){
    return inf_rec(num - 1);
}


int main(){
    inf_rec(100);
    return 0;
}
```

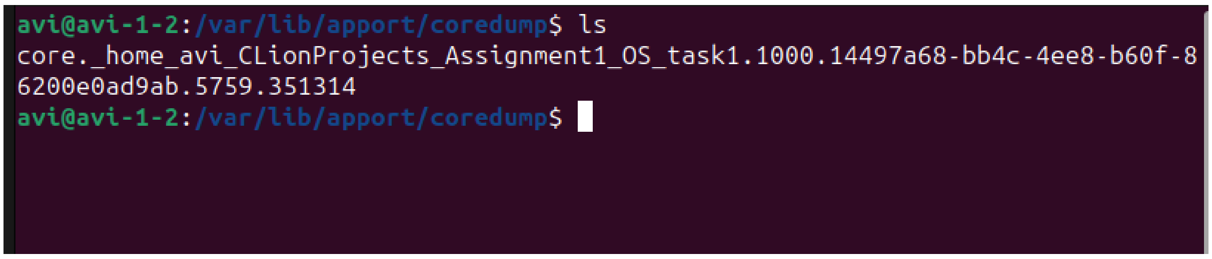
```
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ gcc -o task1 task1.c
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ ./task1
Segmentation fault (core dumped)
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ 
```

we got a segmentation fault, now we are going to debug it and see the results. First, to enable the “core dump” which is a snapshot of the environment at the moment of the crash we need to run this command:



```
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ ulimit -c unlimited
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ 
```

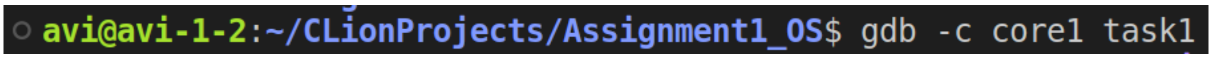
now the “core dump” is enabled and is it stored in this path:



```
avi@avi-1-2:/var/lib/apport/coredump$ ls
core._home_avi_CLionProjects_Assignment1_OS_task1.1000.14497a68-bb4c-4ee8-b60f-8
6200e0ad9ab.5759.351314
avi@avi-1-2:/var/lib/apport/coredump$ 
```

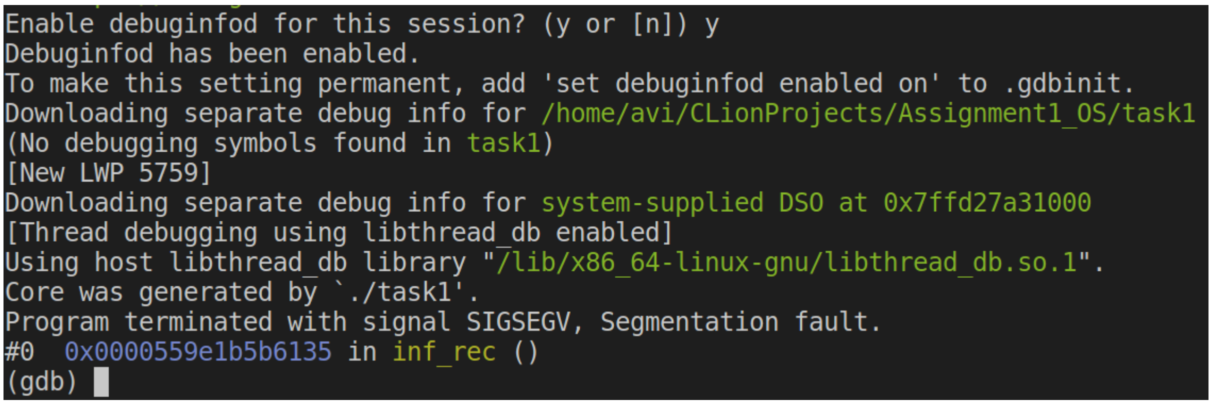
we are going to copy this file to our tasks folder to use it with the executable file.

By running this:



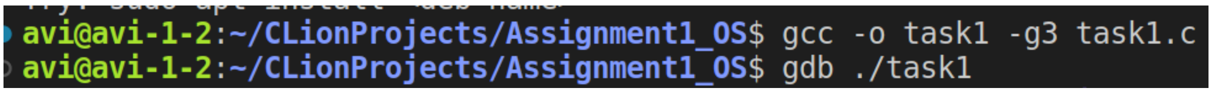
```
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ gdb -c core1 task1
```

we get this result, this is without debugging symbols:



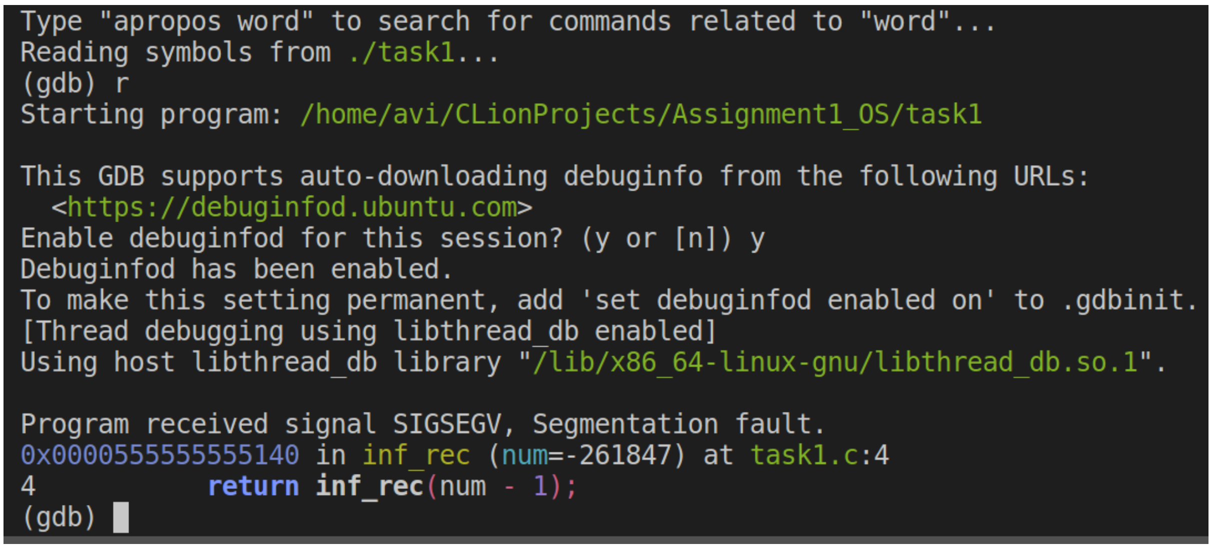
```
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for /home/avi/CLionProjects/Assignment1_OS/task1
(No debugging symbols found in task1)
[New LWP 5759]
Downloading separate debug info for system-supplied DSO at 0x7ffd27a31000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Core was generated by `./task1'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000559e1b5b6135 in inf_rec ()
(gdb) 
```

the SIGSEGV doesn't tell us much about the error, we learned that it says that we are trying to access the wrong memory, but if we want to see it in more detail we can run this with debug symbols like this:



```
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ gcc -o task1 -g3 task1.c
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ gdb ./task1
```

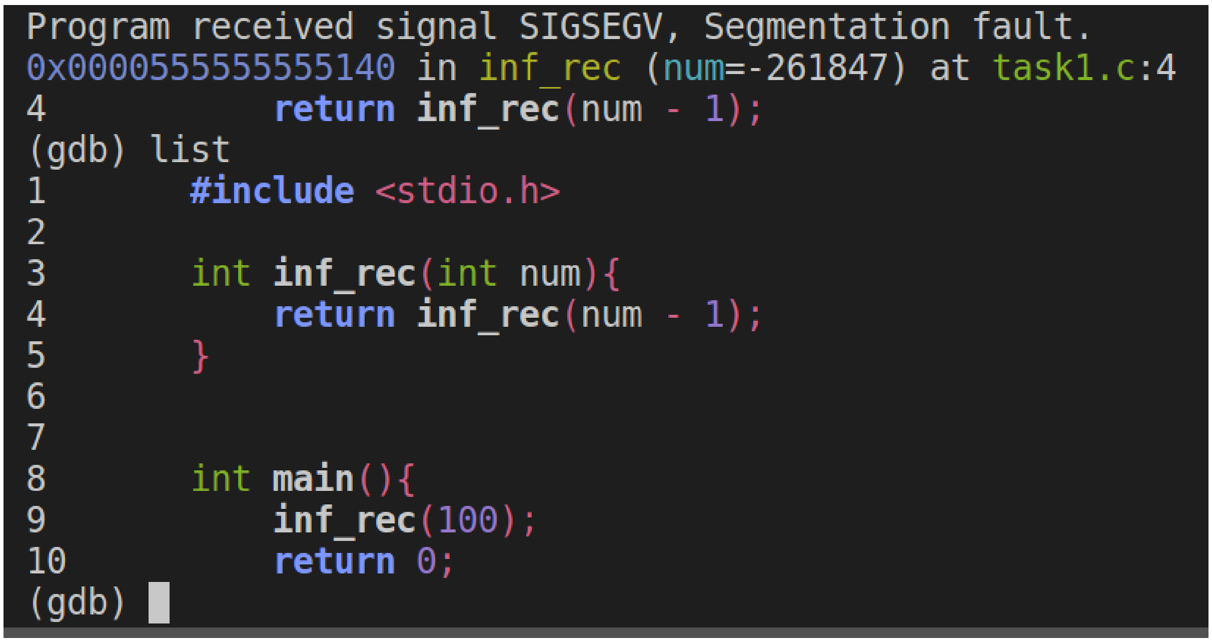
and then we get this(we wrote r to run the program):



```
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./task1...
(gdb) r
Starting program: /home/avi/CLionProjects/Assignment1_OS/task1

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555140 in inf_rec (num=-261847) at task1.c:4
4           return inf_rec(num - 1);
(gdb) 
```

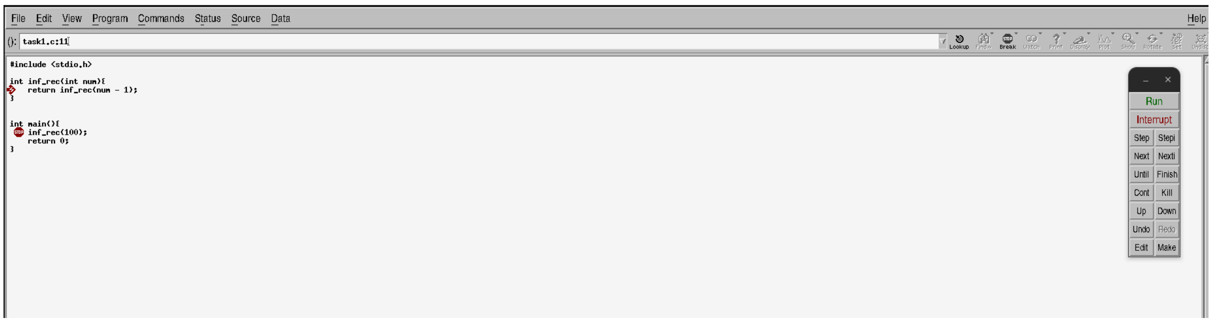
Now we can understand a bit more than before, the value of num was -261847 while the program crashed which is not displayed in the previous core run(without debugging).

The number 4 at the beginning says that the problem was in line 4. To be sure, we can type the list to see the whole code with the lines:



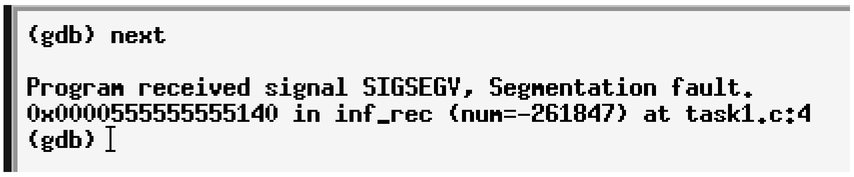
```
Program received signal SIGSEGV, Segmentation fault.
0x0000555555555140 in inf_rec (num=-261847) at task1.c:4
4           return inf_rec(num - 1);
(gdb) list
1       #include <stdio.h>
2
3       int inf_rec(int num){
4           return inf_rec(num - 1);
5       }
6
7
8       int main(){
9           inf_rec(100);
10          return 0;
(gdb) 
```

Now, we get to the part of debugging with graphic debugger, we used ddd graphic debugger.
After running ddd ./task1, this window opened



```
(): task1.c:11

#include <stdio.h>

int inf_rec(int num){
    return inf_rec(num - 1);
}


int main(){
    inf_rec(100);
    return 0;
}
```

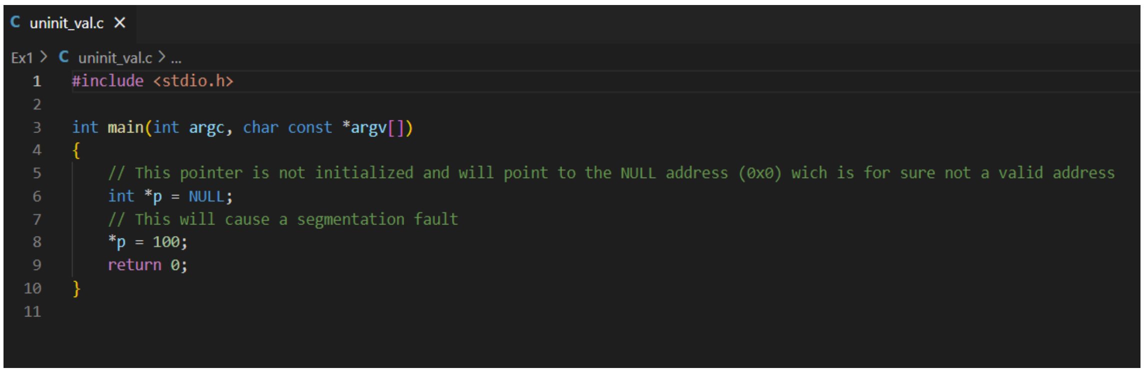
then, we set a breakpoint in the line where we call the recursive function then, press run to run the program, the debugger stopped in the line of the breakpoint and after pressing NEXT we got this:



```
(gdb) next

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555140 in inf_rec (num=-261847) at task1.c:4
(gdb) 
```

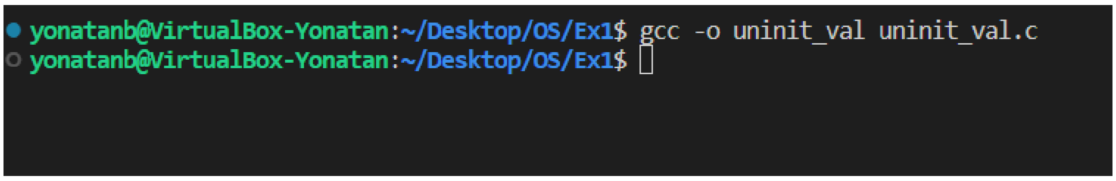
which tell us that we got a segmentation fault.

task 1 bug “B”:
Below is a program that will crash due to an invalid memory access following a call to the memory of a pointer to NULL.



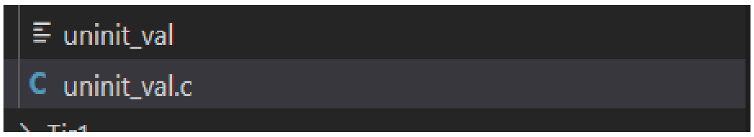
```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    // This pointer is not initialized and will point to the NULL address (0x0) wich is for sure not a valid address
    int *p = NULL;
    // This will cause a segmentation fault
    *p = 100;
    return 0;
}

```

The program should compile successfully using gcc:



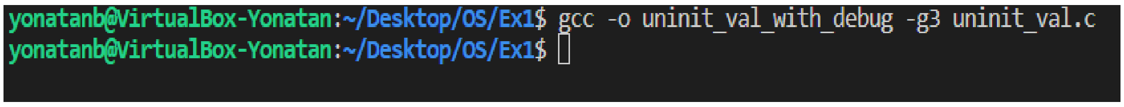
```
yonatanb@VirtualBox-Yonatan:~/Desktop/OS/Ex1$ gcc -o uninit_val uninit_val.c
yonatanb@VirtualBox-Yonatan:~/Desktop/OS/Ex1$
```

And we get the program uninit_val:



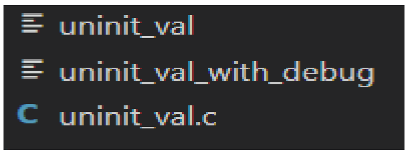
```
uninit_val
uninit_val.c
```

We will also compile the program with a debug flag:



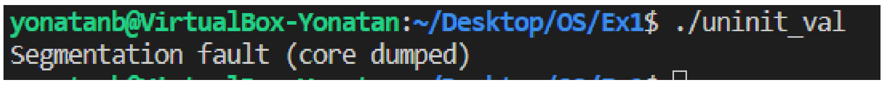
```
yonatanb@VirtualBox-Yonatan:~/Desktop/OS/Ex1$ gcc -o uninit_val_with_debug -g3 uninit_val.c
yonatanb@VirtualBox-Yonatan:~/Desktop/OS/Ex1$
```

And we get the program uninit_val_with_debug:



```
uninit_val
uninit_val_with_debug
uninit_val.c
```

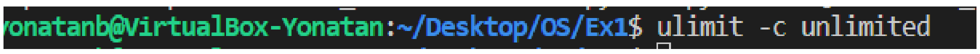
Now we will run the program:



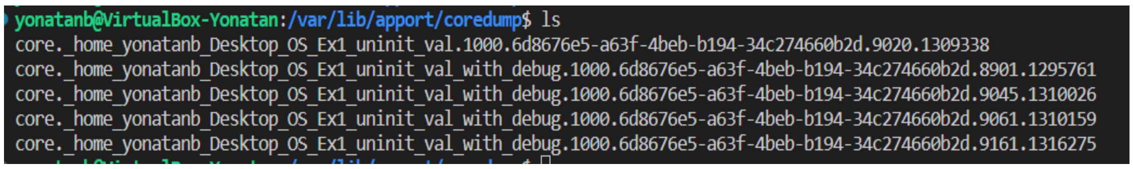
```
yonatanb@VirtualBox-Yonatan:~/Desktop/OS/Ex1$ ./uninit_val
Segmentation fault (core dumped)
```

We will receive a segmentation fault (illegal access to memory) and the core dump will return to us, which is a kind of image of the state of the memory at the moment of the crash.

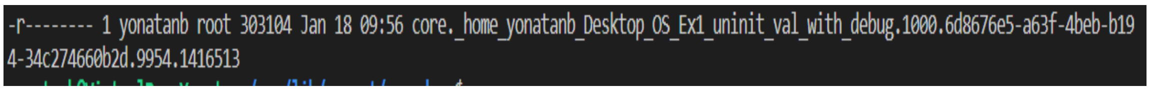
We will run the following command:


```
onatanb@VirtualBox-Yonatan:~/Desktop/OS/Ex1$ ulimit -c unlimited
```

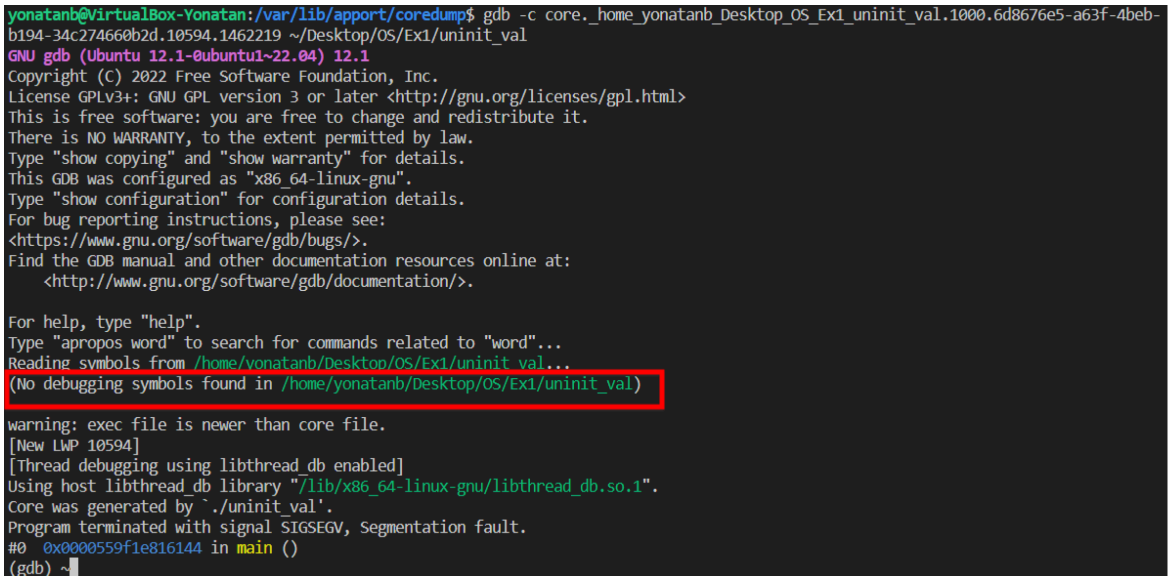
In /var/lib/apport/coredump all the core files that the errors created are saved:


```
yonatanb@VirtualBox-Yonatan:/var/lib/apport/coredump$ ls
core._home_yonatanb_Desktop_OS_Ex1_uninit_val.1000.6d8676e5-a63f-4beb-b194-34c274660b2d.9020.1309338
core._home_yonatanb_Desktop_OS_Ex1_uninit_val_with_debug.1000.6d8676e5-a63f-4beb-b194-34c274660b2d.8901.1295761
core._home_yonatanb_Desktop_OS_Ex1_uninit_val_with_debug.1000.6d8676e5-a63f-4beb-b194-34c274660b2d.9045.1310026
core._home_yonatanb_Desktop_OS_Ex1_uninit_val_with_debug.1000.6d8676e5-a63f-4beb-b194-34c274660b2d.9061.1310159
core._home_yonatanb_Desktop_OS_Ex1_uninit_val_with_debug.1000.6d8676e5-a63f-4beb-b194-34c274660b2d.9161.1316275
```

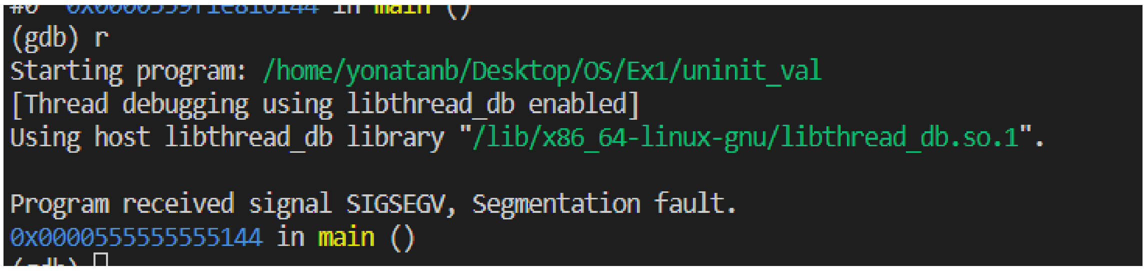
Now we will run the program one more time and find the core created:


```
-r-------- 1 yonatanb root 303104 Jan 18 09:56 core._home_yonatanb_Desktop_OS_Ex1_uninit_val_with_debug.1000.6d8676e5-a63f-4beb-b19
4-34c274660b2d.9954.1416513
```

Now we open the core file created for the program without the debug flag:

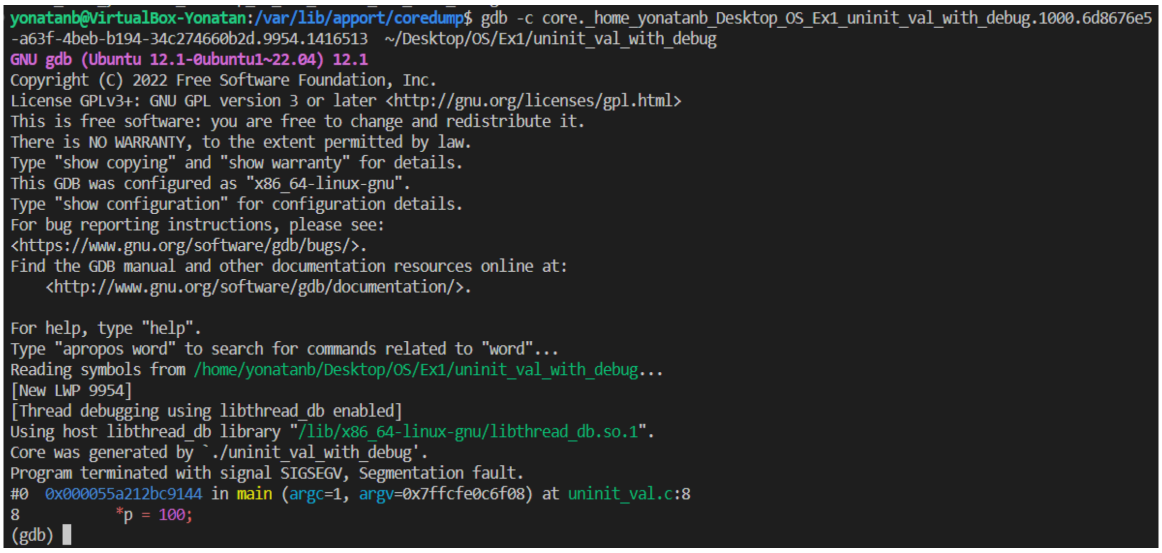

```
yonatanb@VirtualBox-Yonatan:/var/lib/apport/coredump$ gdb -c core._home_yonatanb_Desktop_OS_Ex1_uninit_val.1000.6d8676e5-a63f-4beb-
b194-34c274660b2d.10594.1462219 ~/Desktop/OS/Ex1/uninit_val
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: 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:
<https://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 /home/yonatanb/Desktop/OS/Ex1/uninit_val...
(No debugging symbols found in /home/yonatanb/Desktop/OS/Ex1/uninit_val)

warning: exec file is newer than core file.
[New LWP 10594]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Core was generated by `./uninit_val'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000559f1e816144 in main ()
(gdb)
```

Let's try to run to find the problematic line in the code:


```
(gdb) r
Starting program: /home/yonatanb/Desktop/OS/Ex1/uninit_val
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555144 in main ()
```

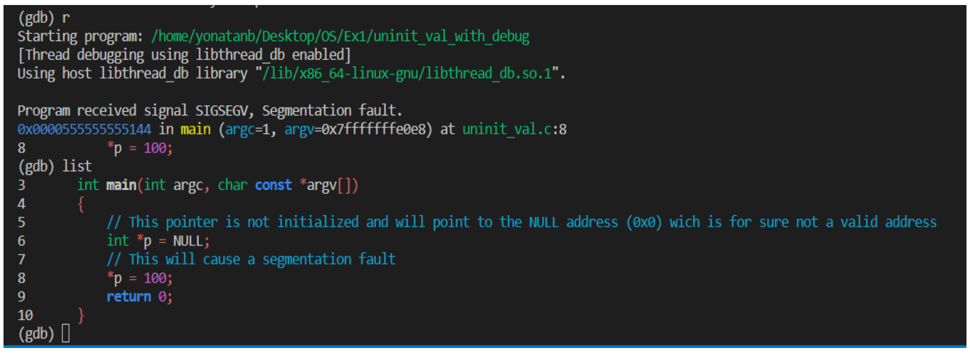
It is impossible to conclude what the problematic line is, only to understand that the problem is in main and that a signal is sent to a program of type SIGSEGV.

Open the core file created for the program with the debug flag:



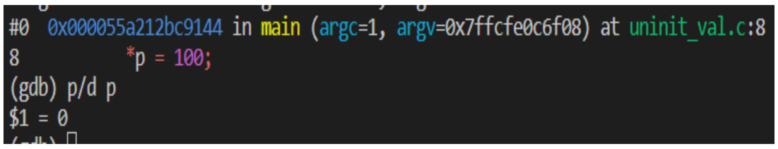
```
yonatanb@VirtualBox-Yonatan:/var/lib/apport/coredump$ gdb -c core._home_yonatanb_Desktop_OS_Ex1_uninit_val_with_debug.1000.6d8676e5
-a63f-4beb-b194-34c274660b2d.9954.1416513  ~/Desktop/OS/Ex1/uninit_val_with_debug
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, 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:
<https://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 /home/yonatanb/Desktop/OS/Ex1/uninit_val_with_debug...
[New LWP 9954]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Core was generated by `./uninit_val_with_debug'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000055a212bc9144 in main (argc=1, argv=0x7ffcfe0c6f08) at uninit_val.c:8
8           *p = 100;
(gdb) 
```

We can see that the problem is in the uninit_val.c file, in the main function and in line 8, including the contents of the line. You can also use list for more lines:



```
(gdb) r
Starting program: /home/yonatanb/Desktop/OS/Ex1/uninit_val_with_debug
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555144 in main (argc=1, argv=0x7fffffffe0e8) at uninit_val.c:8
8           *p = 100;
(gdb) list
3       int main(int argc, char const *argv[])
4       {
5           // This pointer is not initialized and will point to the NULL address (0x0) wich is for sure not a valid address
6           int *p = NULL;
7           // This will cause a segmentation fault
8           *p = 100;
9           return 0;
10      }
(gdb) 
```

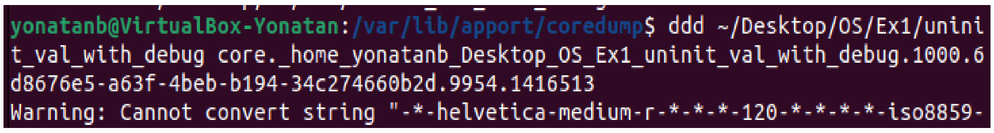
We can see that the value of p was 0:



```
#0  0x000055a212bc9144 in main (argc=1, argv=0x7ffcfe0c6f08) at uninit_val.c:8
8           *p = 100;
(gdb) p/d p
$1 = 0
```

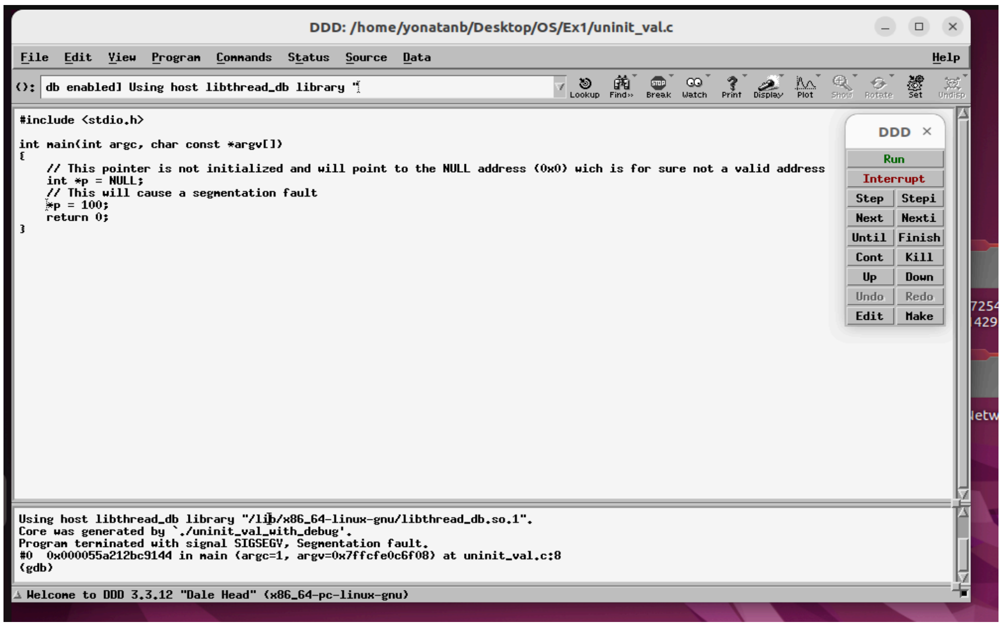
Now we open the core using a graphical debugger (ddd) using the command:

ddd program-file core.



```
yonatanb@VirtualBox-Yonatan:/var/lib/apport/coredump$ ddd ~/Desktop/OS/Ex1/unini
t_val_with_debug core._home_yonatanb_Desktop_OS_Ex1_uninit_val_with_debug.1000.6
d8676e5-a63f-4beb-b194-34c274660b2d.9954.1416513
Warning: Cannot convert string "-*-helvetica-medium-r-*-*-*-120-*-*-*-*-iso8859-
```

Now we can look at the code:

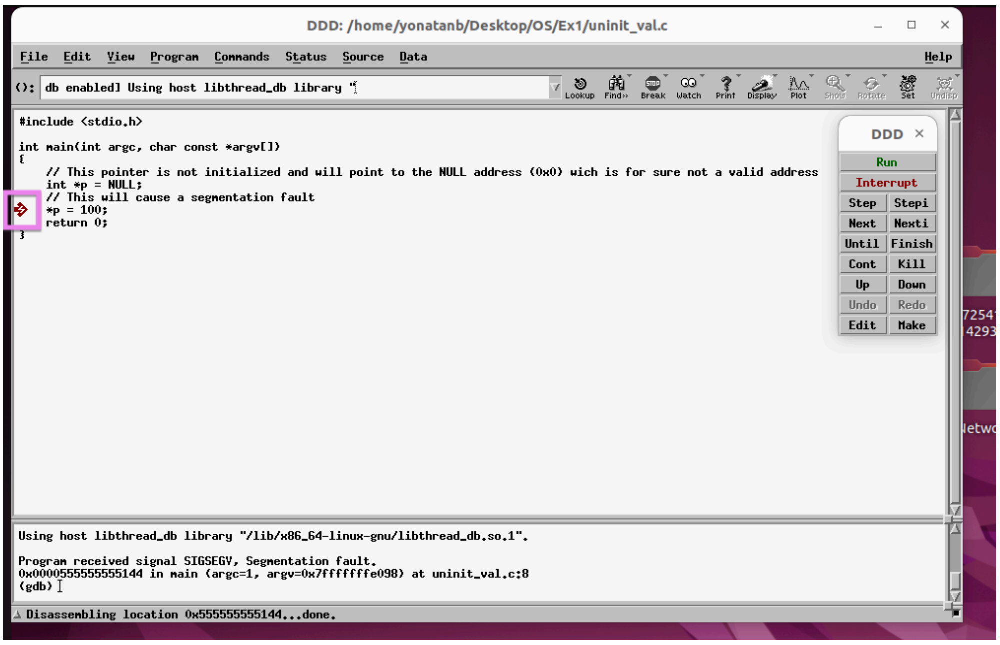

DDD: /home/yonatanb/Desktop/OS/Ex1/uninit_val.c
File Edit View Program Commands Status Source Data Help
(): db enabled] Using host libthread_db library "
Lookup Find Break Watch Print Display Plot Show Rotate Set Undisp
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make

```
#include <stdio.h>

int main(int argc, char const *argv[])
{
    // This pointer is not initialized and will point to the NULL address (0x0) wich is for sure not a valid address
    int *p = NULL;
    // This will cause a segmentation fault
    *p = 100;
    return 0;
}
```

```
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Core was generated by `./uninit_val_with_debug'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000055a212bc9144 in main (argc=1, argv=0x7ffcfe0c6f08) at uninit_val.c:8
(gdb)
```

```
Welcome to DDD 3.3.12 "Dale Head" (x86_64-pc-linux-gnu)
```

We will activate the run and we can visually see where the problem happened:


DDD: /home/yonatanb/Desktop/OS/Ex1/uninit_val.c
File Edit View Program Commands Status Source Data Help
(): db enabled] Using host libthread_db library "
Lookup Find Break Watch Print Display Plot Show Rotate Set Undisp
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make

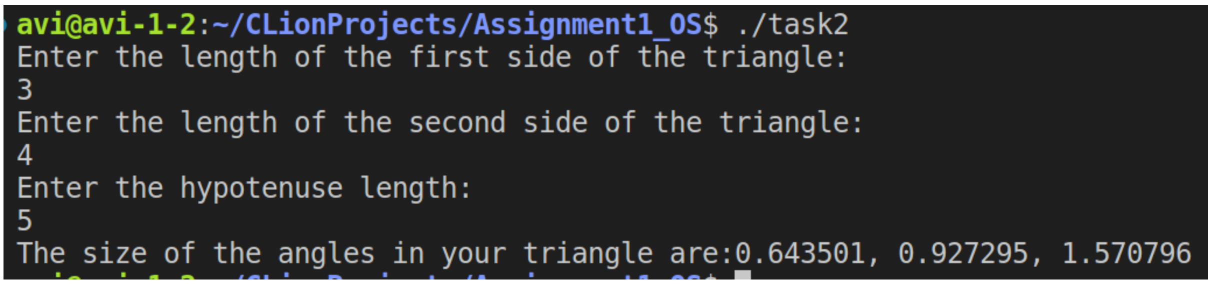
```
#include <stdio.h>

int main(int argc, char const *argv[])
{
    // This pointer is not initialized and will point to the NULL address (0x0) wich is for sure not a valid address
    int *p = NULL;
    // This will cause a segmentation fault
    *p = 100;
    return 0;
}
```

```
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555144 in main (argc=1, argv=0x7fffffffe098) at uninit_val.c:8
(gdb)
```

```
Disassembling location 0x555555555144...done.
```

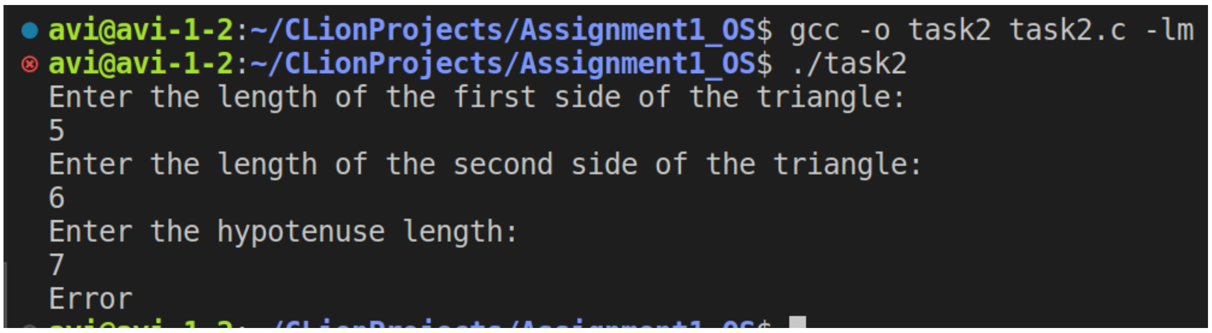
task 2:

Example of correct pythagorean triple:



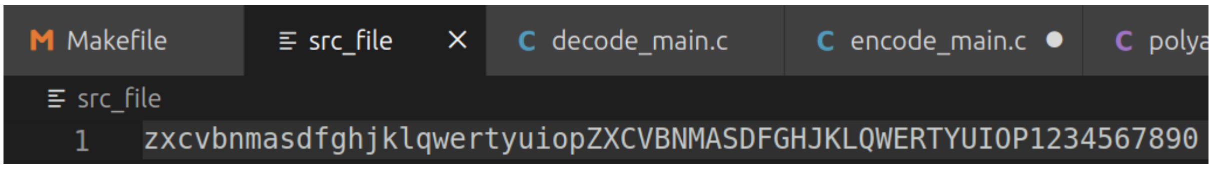
```
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ ./task2
Enter the length of the first side of the triangle:
3
Enter the length of the second side of the triangle:
4
Enter the hypotenuse length:
5
The size of the angles in your triangle are:0.643501, 0.927295, 1.570796
```

Example of bad sides of the triangle:



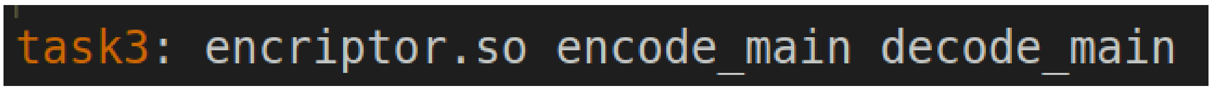
```
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ gcc -o task2 task2.c -lm
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ ./task2
Enter the length of the first side of the triangle:
5
Enter the length of the second side of the triangle:
6
Enter the hypotenuse length:
7
Error
```

task 3:
First we need to store a 62 characters string into file named src_file like this:



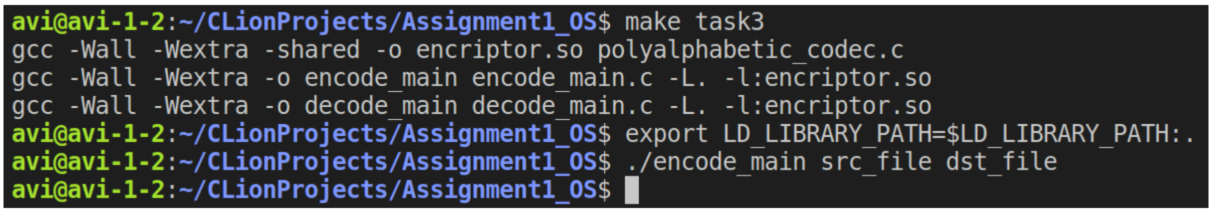
```
zxcvbnmasdfghjklqwertyuiopZXCVBNMASDFGHJKLQWERTYUIOP1234567890
```

then, we need to run make task3 to run the necessary files and libs for this task, task3 compile this files:



```
task3: encriptor.so encode_main decode_main
```

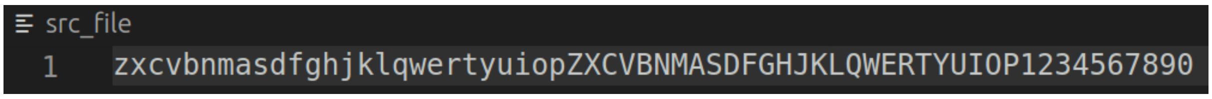
the encriptor.so is the dynamic library we created and the encode_main, decode_main are the main for encoding and decoding the string we store in the src_file.
This is how you should run the programs:



```
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ make task3
gcc -Wall -Wextra -shared -o encriptor.so polyalphabetic_codec.c
gcc -Wall -Wextra -o encode_main encode_main.c -L. -l:encriptor.so
gcc -Wall -Wextra -o decode_main decode_main.c -L. -l:encriptor.so
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
avi@avi-1-2:~/CLionProjects/Assignment1_OS$ ./encode_main src_file dst_file
avi@avi-1-2:~/CLionProjects/Assignment1_OS$
```

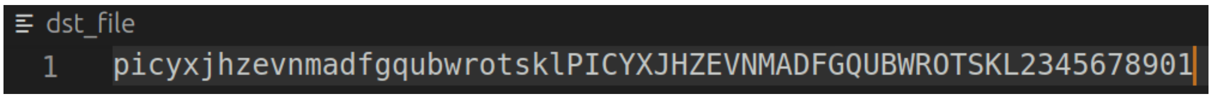
- make task3 as we explained.
- export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
  for saying to the runtime linker where to look for the library encriptor.so.
- ./encode_main src_file dst_file for running the encode_main program that takes the src_file and read the string stored in it and encoding it to dst_file.

for example, if we had this string in src_file:



```
zxcvbnmasdfghjklqwertyuiopZXCVBNMASDFGHJKLQWERTYUIOP1234567890
```

so, in the dst_file we will see:



```
picyxjhzevnmadfgqubwrotsklPICYXJHZEVNMADFGQUBWROTSKL2345678901
```

the same running for decode_main, just replace the encode_main by decode_main and this will take the string from src_file and decode it into dst_file.

task 4:

1. Building the Program

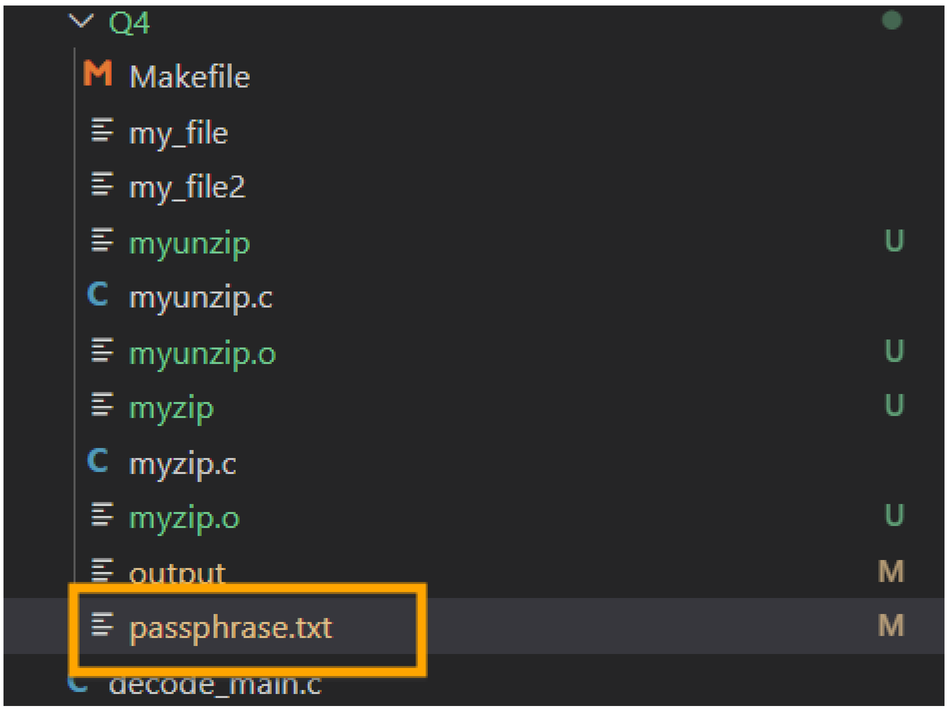
- Navigate to the project directory.
- Build the program by executing make all or make task4.

2. Creating a GPG Key

- Generate a new GPG key using gpg --gen-key.
- Complete the key creation by following the on-screen prompts.

3. Storing the Passphrase

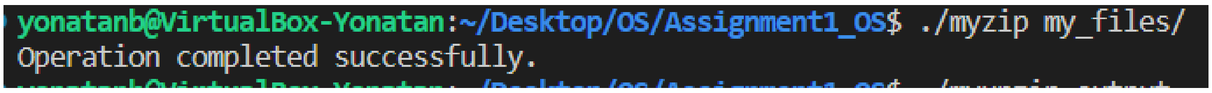
- Save the passphrase used for the GPG key in passphrase.txt.



4. Compressing and Encrypting Files

- Compress and encrypt files using ./myzip <directory/file>.

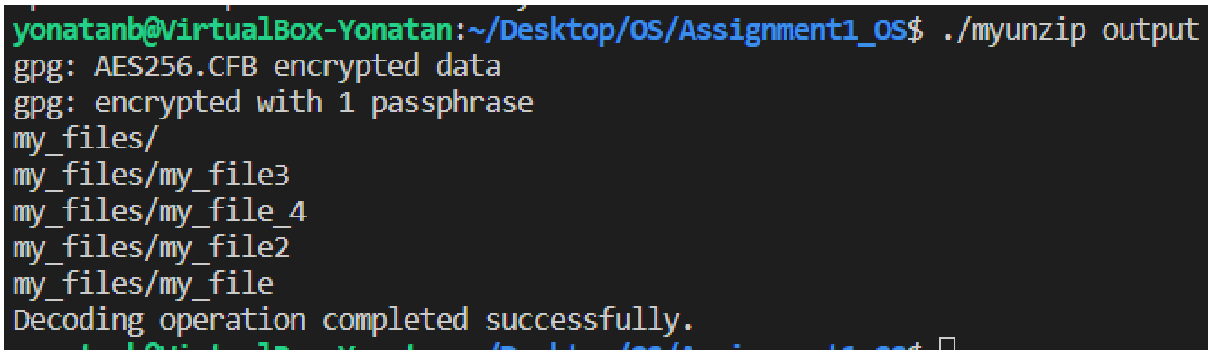
The resulting output file will be named output.



```
yonatanb@VirtualBox-Yonatan:~/Desktop/OS/Assignment1_OS$ ./myzip my_files/
Operation completed successfully.
```

5. Decompressing and Decrypting File

Decompress and decrypt the file by running ./myunzip <compressed file>.



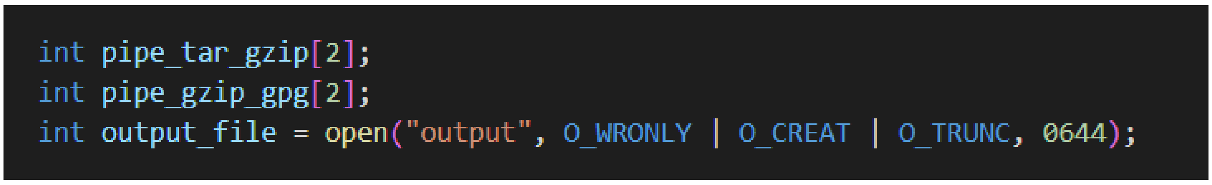
```
yonatanb@VirtualBox-Yonatan:~/Desktop/OS/Assignment1_OS$ ./myunzip output
gpg: AES256.CFB encrypted data
gpg: encrypted with 1 passphrase
my_files/
my_files/my_file3
my_files/my_file_4
my_files/my_file2
my_files/my_file
Decoding operation completed successfully.
```

Implementation Notes

- Pipes and Child Processes:

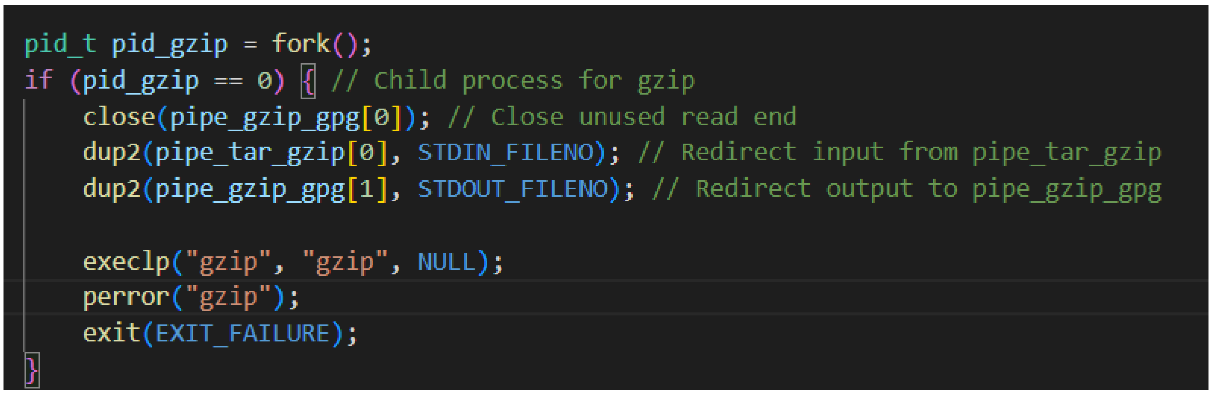
The implementation utilizes pipes for a seamless pipeline between the gpg, tar, and gzip commands.

- Three child processes are created for tar, gzip, and gpg.



```c
int pipe_tar_gzip[2];
int pipe_gzip_gpg[2];
int output_file = open("output", O_WRONLY | O_CREAT | O_TRUNC, 0644);
```

Each child process duplicates (dup2) the file descriptor for the pipe's read-end to stdin, and stdout to the pipe's write-end.



```c
pid_t pid_gzip = fork();
if (pid_gzip == 0) { // Child process for gzip
    close(pipe_gzip_gpg[0]); // Close unused read end
    dup2(pipe_tar_gzip[0], STDIN_FILENO); // Redirect input from pipe_tar_gzip
    dup2(pipe_gzip_gpg[1], STDOUT_FILENO); // Redirect output to pipe_gzip_gpg

    execlp("gzip", "gzip", NULL);
    perror("gzip");
    exit(EXIT_FAILURE);
}
```