Terminal GDB Debugging - CSEL Guide

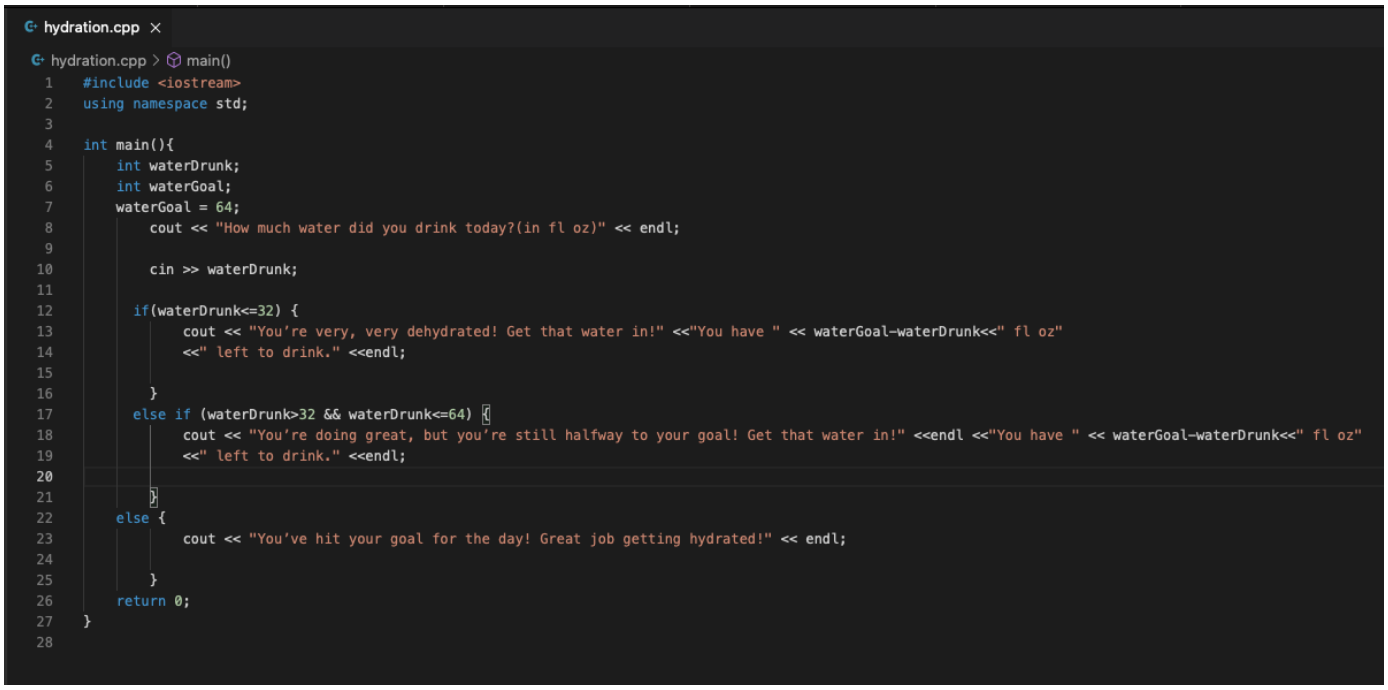
GNU debugger or GDB is an application for finding out how your C++ program runs or for analyzing the moment the program crashes. You can perform many useful tasks with GDB: run the program, stop the program under specific conditions, analyze the situation, make modifications, and test new changes.

Installation:

GDB is pre-installed on linux systems. Since the cloud environment, cs1300.csel.io (AKA coding.csel.io) uses a linux system, no separate installation is required.

Debugging with gdb in VS Code Terminal.

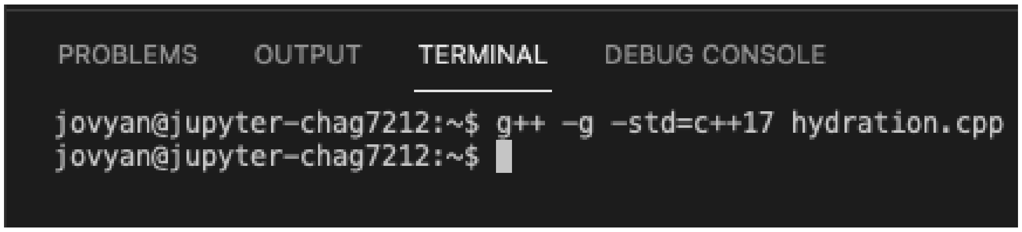
1. Let us debug the code for the **Hydration App** question from recitation 3.



```cpp
#include <iostream>
using namespace std;

int main(){
    int waterDrunk;
    int waterGoal;
    waterGoal = 64;
        cout << "How much water did you drink today?(in fl oz)" << endl;

        cin >> waterDrunk;

      if(waterDrunk<=32) {
            cout << "You're very, very dehydrated! Get that water in!" <<"You have " << waterGoal-waterDrunk<<" fl oz"
            <<" left to drink." <<endl;

        }
      else if (waterDrunk>32 && waterDrunk<=64) {
            cout << "You're doing great, but you're still halfway to your goal! Get that water in!" <<endl <<"You have " << waterGoal-waterDrunk<<" fl oz"
            <<" left to drink." <<endl;

        }
    else {
            cout << "You've hit your goal for the day! Great job getting hydrated!" << endl;

        }
    return 0;
}
```

2. Complete the code for the question and open a terminal to start debugging.
3. The first is to compile the C++ code with the -g flag:

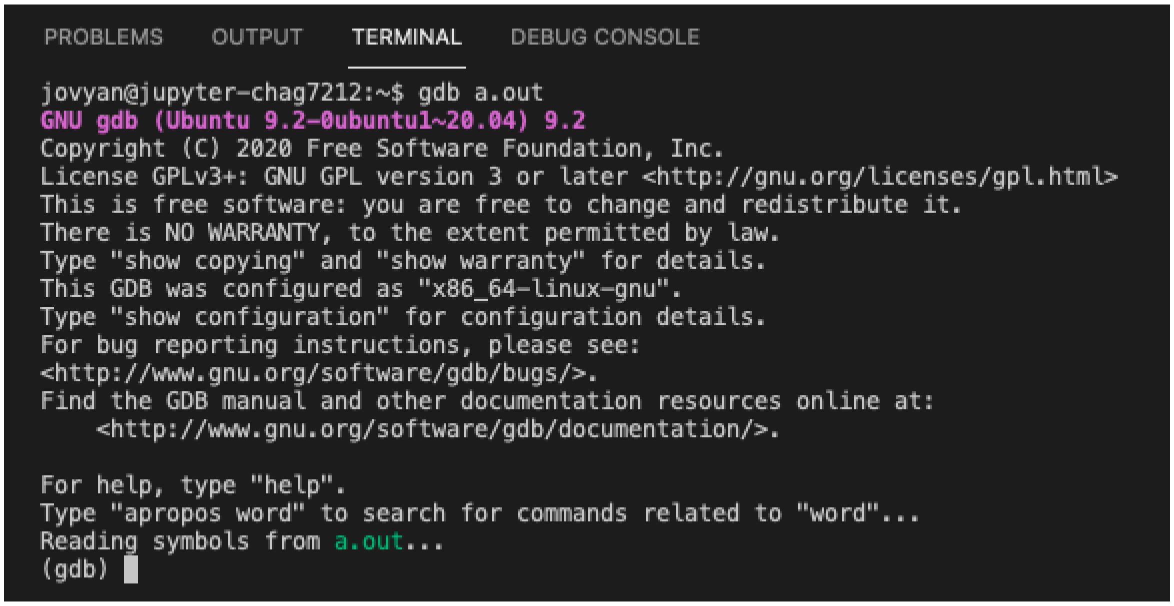
```
g++ -g -std=c++17 filename.cpp
```



```
jovyan@jupyter-chag7212:~$ g++ -g -std=c++17 hydration.cpp
jovyan@jupyter-chag7212:~$
```

4. The next step is calling the GDB to start the debugging process for the program:

   Type `gdb a.out` and press enter once to get to the gdb prompt.

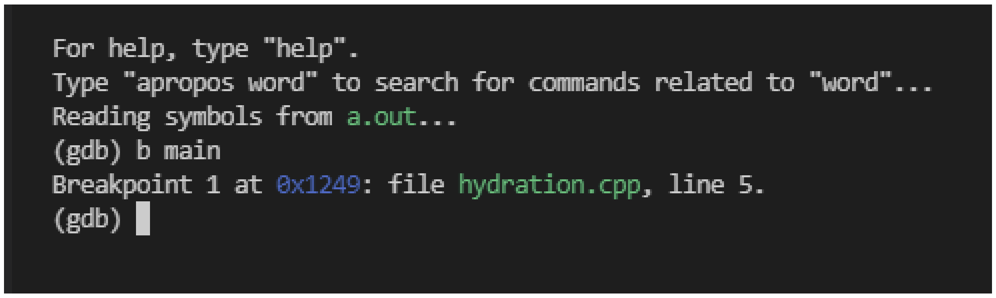


```
jovyan@jupyter-chag7212:~$ gdb a.out
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to 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...
(gdb)
```

Let us set breakpoints to stop the execution of the program.

5. It is possible to set breakpoints in two ways.
6. The following example sets a breakpoint at the start of the main function:

```
$(gdb)b main
```



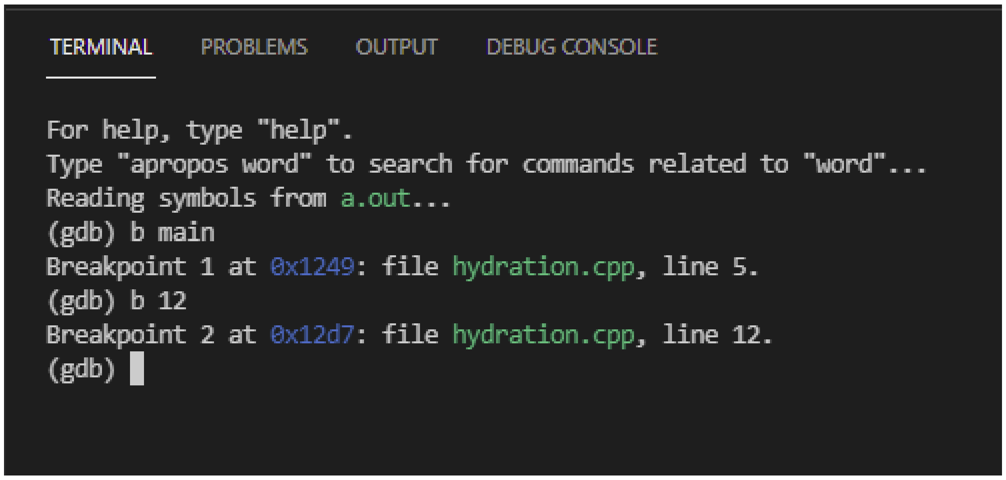
```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) b main
Breakpoint 1 at 0x1249: file hydration.cpp, line 5.
(gdb)
```

Breakpoints for other user defined function can be set in a similar way:

```
$ (gdb)b function_name
```

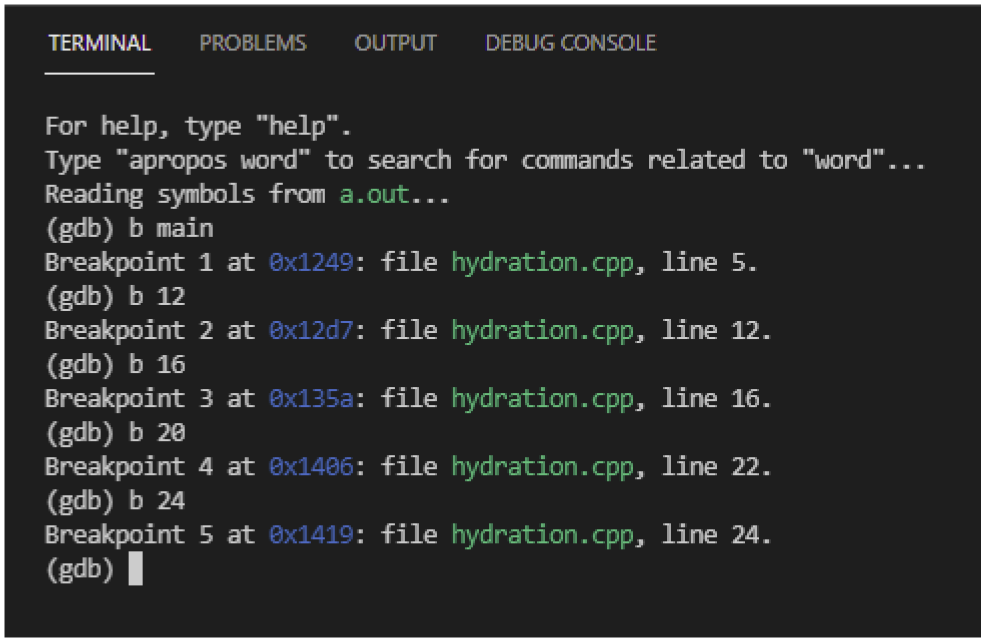
7. This example sets a breakpoint at a specific line (12):

```
$(gdb)b 10
```



```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) b main
Breakpoint 1 at 0x1249: file hydration.cpp, line 5.
(gdb) b 12
Breakpoint 2 at 0x12d7: file hydration.cpp, line 12.
(gdb)
```

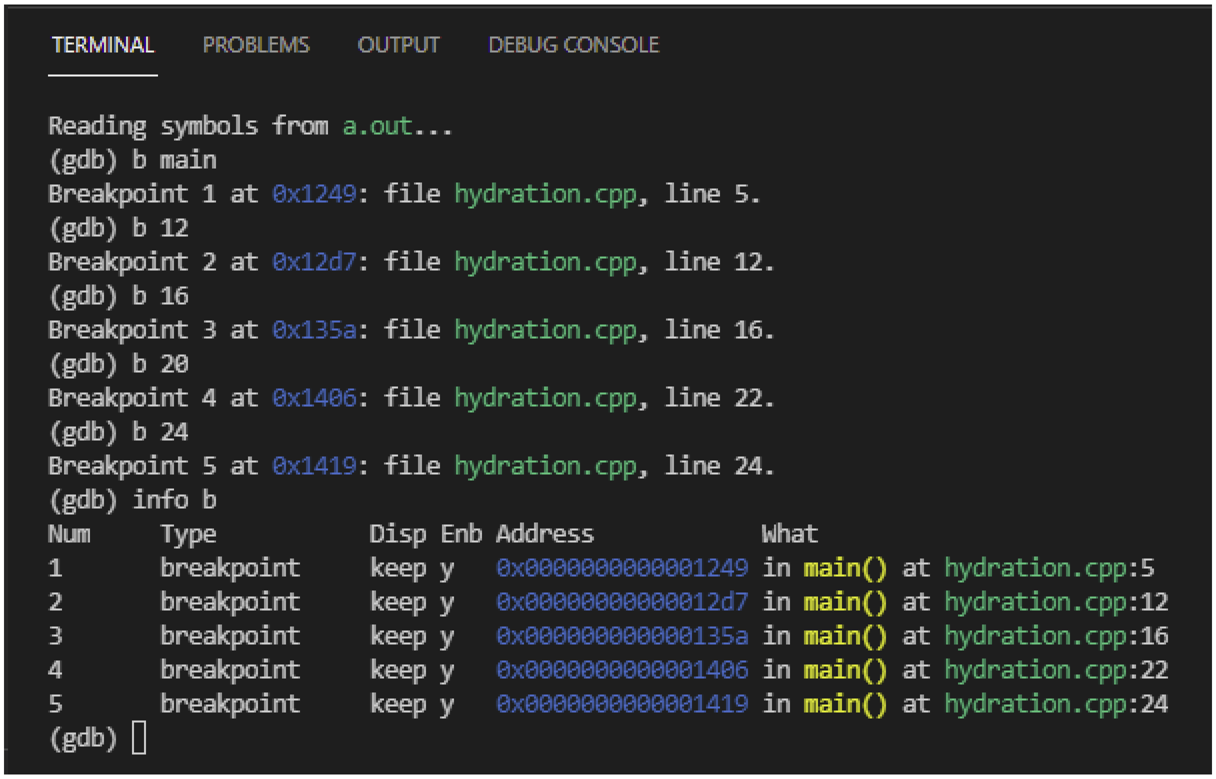
8. Let us set a few more breakpoints at lines 16, 20 and 24.



```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) b main
Breakpoint 1 at 0x1249: file hydration.cpp, line 5.
(gdb) b 12
Breakpoint 2 at 0x12d7: file hydration.cpp, line 12.
(gdb) b 16
Breakpoint 3 at 0x135a: file hydration.cpp, line 16.
(gdb) b 20
Breakpoint 4 at 0x1406: file hydration.cpp, line 22.
(gdb) b 24
Breakpoint 5 at 0x1419: file hydration.cpp, line 24.
(gdb)
```

9. Now we can list all the breakpoints using the following example:

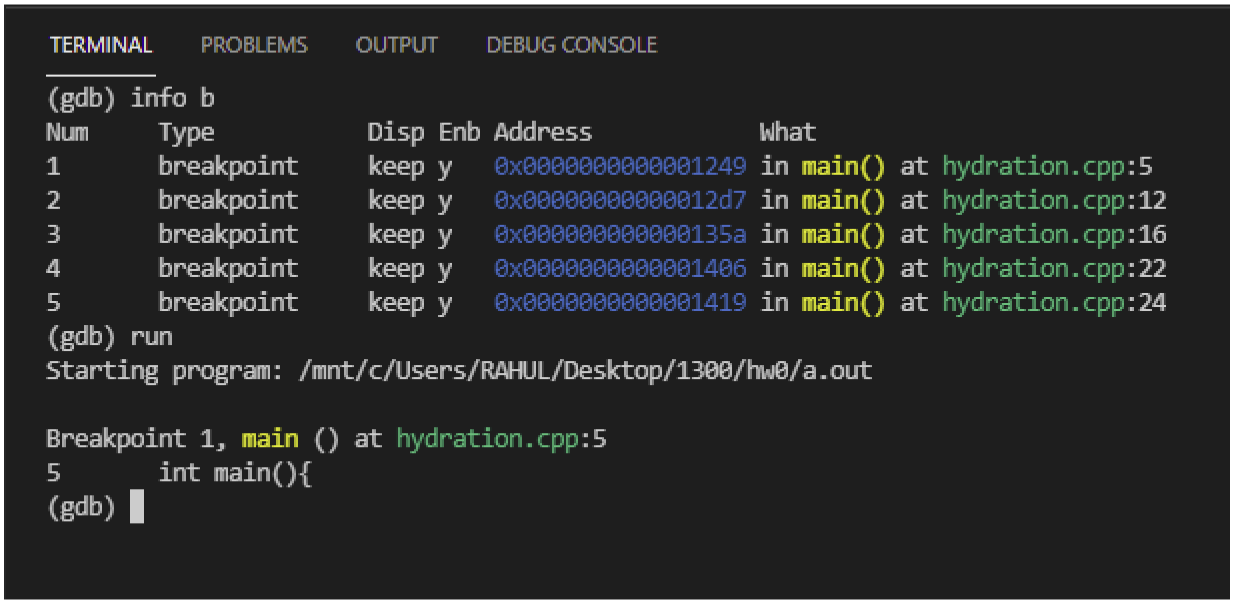
```
$(gdb)info b
```



```
Reading symbols from a.out...
(gdb) b main
Breakpoint 1 at 0x1249: file hydration.cpp, line 5.
(gdb) b 12
Breakpoint 2 at 0x12d7: file hydration.cpp, line 12.
(gdb) b 16
Breakpoint 3 at 0x135a: file hydration.cpp, line 16.
(gdb) b 20
Breakpoint 4 at 0x1406: file hydration.cpp, line 22.
(gdb) b 24
Breakpoint 5 at 0x1419: file hydration.cpp, line 24.
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001249 in main() at hydration.cpp:5
2       breakpoint     keep y   0x00000000000012d7 in main() at hydration.cpp:12
3       breakpoint     keep y   0x000000000000135a in main() at hydration.cpp:16
4       breakpoint     keep y   0x0000000000001406 in main() at hydration.cpp:22
5       breakpoint     keep y   0x0000000000001419 in main() at hydration.cpp:24
(gdb) 
```

10. Let us start running the program to debug.

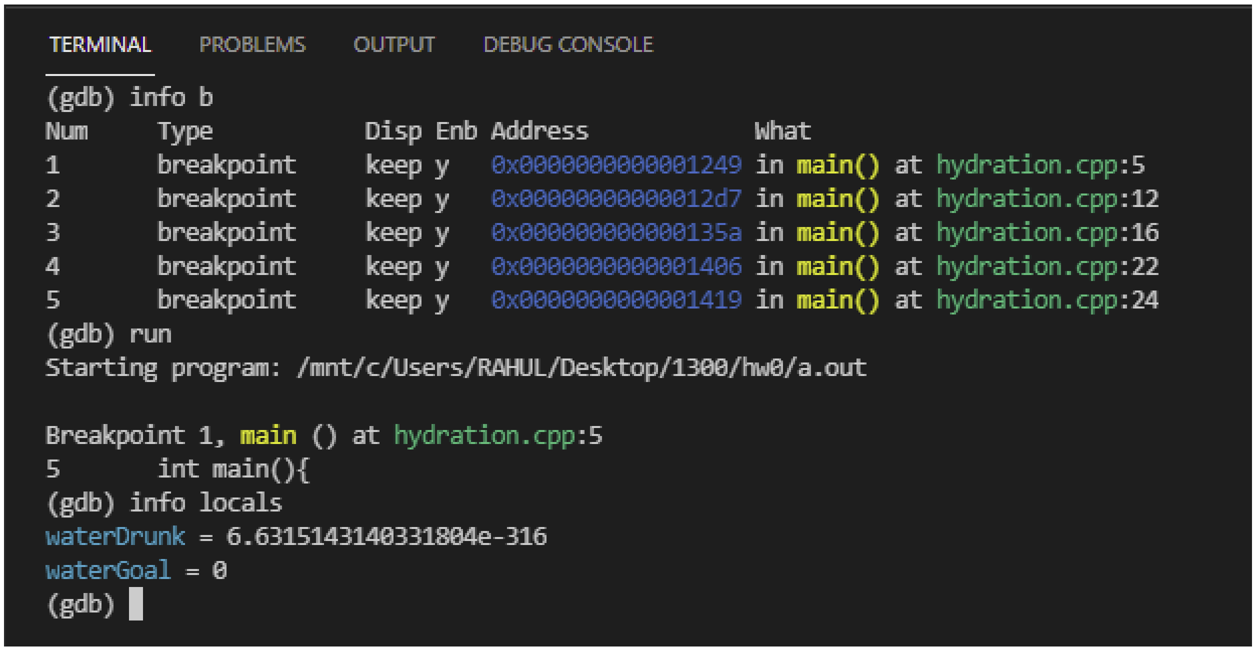
```
$(gdb)run
```



```
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001249 in main() at hydration.cpp:5
2       breakpoint     keep y   0x00000000000012d7 in main() at hydration.cpp:12
3       breakpoint     keep y   0x000000000000135a in main() at hydration.cpp:16
4       breakpoint     keep y   0x0000000000001406 in main() at hydration.cpp:22
5       breakpoint     keep y   0x0000000000001419 in main() at hydration.cpp:24
(gdb) run
Starting program: /mnt/c/Users/RAHUL/Desktop/1300/hw0/a.out

Breakpoint 1, main () at hydration.cpp:5
5       int main(){
(gdb) 
```

11. Next we will print all the variables in the local scope using the following example.

```
$(gdb)info locals
```



```
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001249 in main() at hydration.cpp:5
2       breakpoint     keep y   0x00000000000012d7 in main() at hydration.cpp:12
3       breakpoint     keep y   0x000000000000135a in main() at hydration.cpp:16
4       breakpoint     keep y   0x0000000000001406 in main() at hydration.cpp:22
5       breakpoint     keep y   0x0000000000001419 in main() at hydration.cpp:24
(gdb) run
Starting program: /mnt/c/Users/RAHUL/Desktop/1300/hw0/a.out

Breakpoint 1, main () at hydration.cpp:5
5       int main(){
(gdb) info locals
waterDrunk = 6.6315143140331804e-316
waterGoal = 0
(gdb) 
```

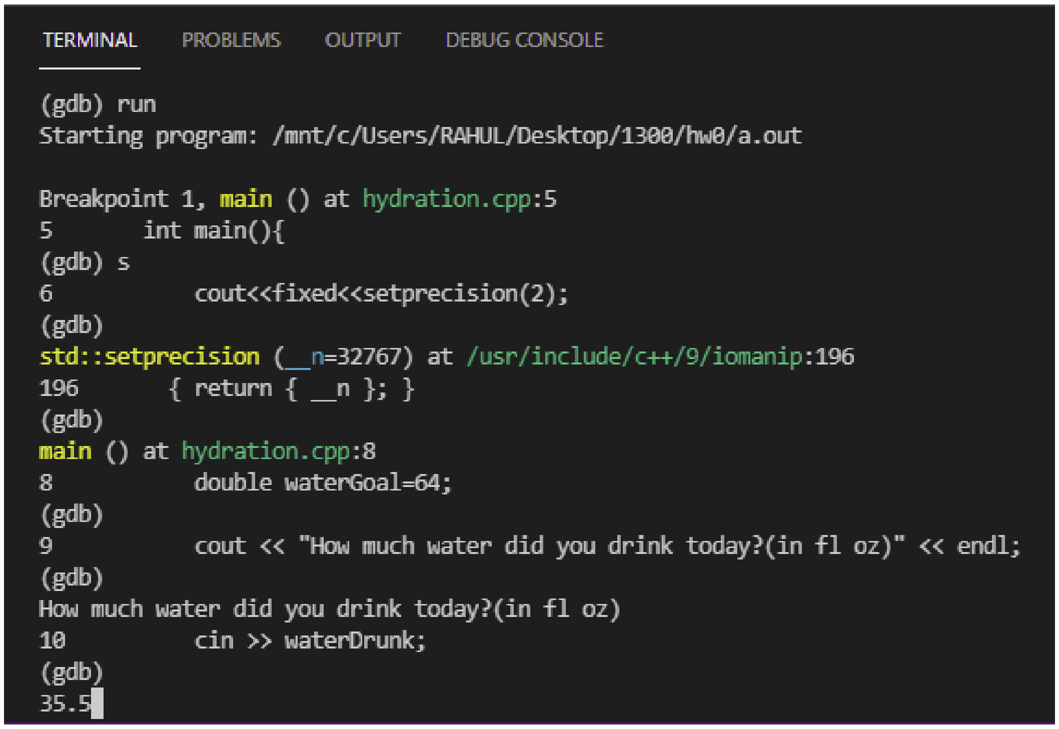
As we can see the variables haven't been assigned any value yet, hence they assign random values. The variable **waterDrunk** stores the input from the user and the variable **waterGoal** will have a fixed value of 64.00.

To print all variables including the ones in global scope, use `$(gdb)info variables`

12. We can debug each line using two concepts called 'Step into' and 'Step over'.

13. The command for 'Step into' is `$(gdb)step` or `$(gdb)s` and command for 'Step over' is `$(gdb)next` or `$(gdb)n`

14. 'Step into' program, proceeds through subroutine calls. Whereas 'Step over' command does not enter the subroutine, but instead steps over.

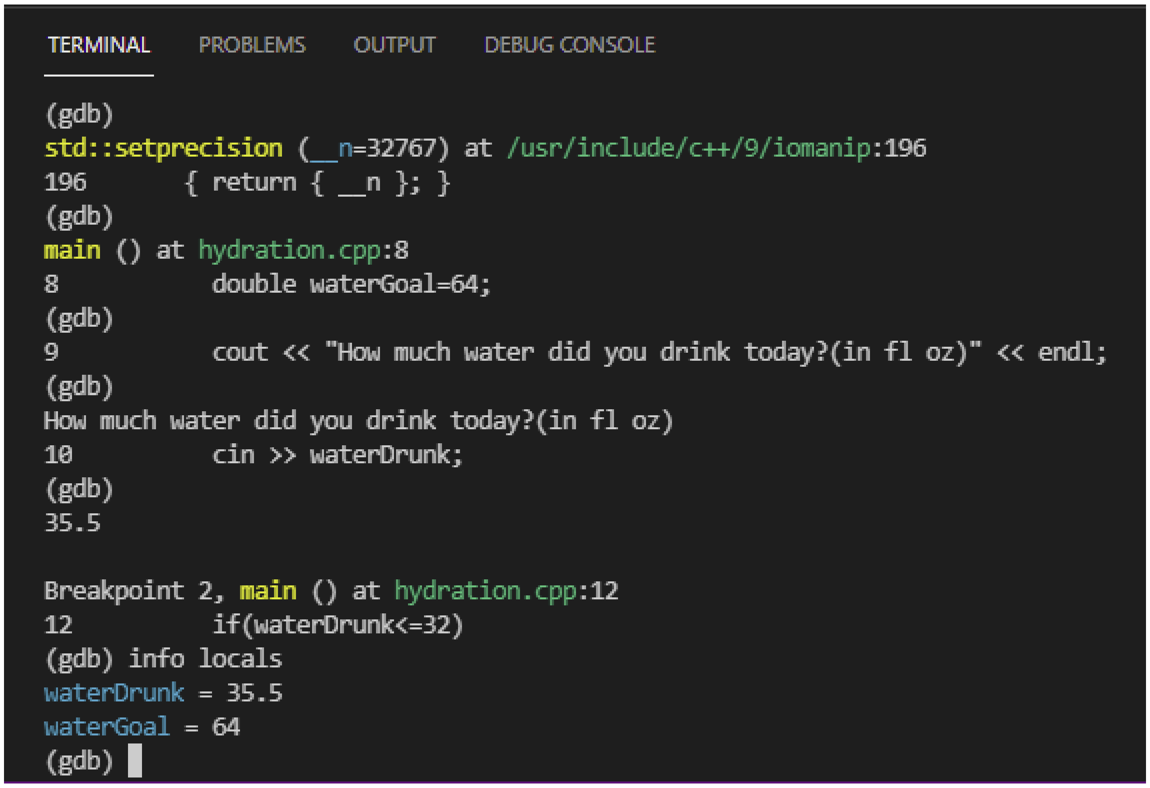


TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
(gdb) run
Starting program: /mnt/c/Users/RAHUL/Desktop/1300/hw0/a.out

Breakpoint 1, main () at hydration.cpp:5
5       int main(){
(gdb) s
6           cout<<fixed<<setprecision(2);
(gdb)
std::setprecision (__n=32767) at /usr/include/c++/9/iomanip:196
196       { return { __n }; }
(gdb)
main () at hydration.cpp:8
8           double waterGoal=64;
(gdb)
9           cout << "How much water did you drink today?(in fl oz)" << endl;
(gdb)
How much water did you drink today?(in fl oz)
10          cin >> waterDrunk;
(gdb)
35.5
```

As we can see, the command steps into the setprecision function.

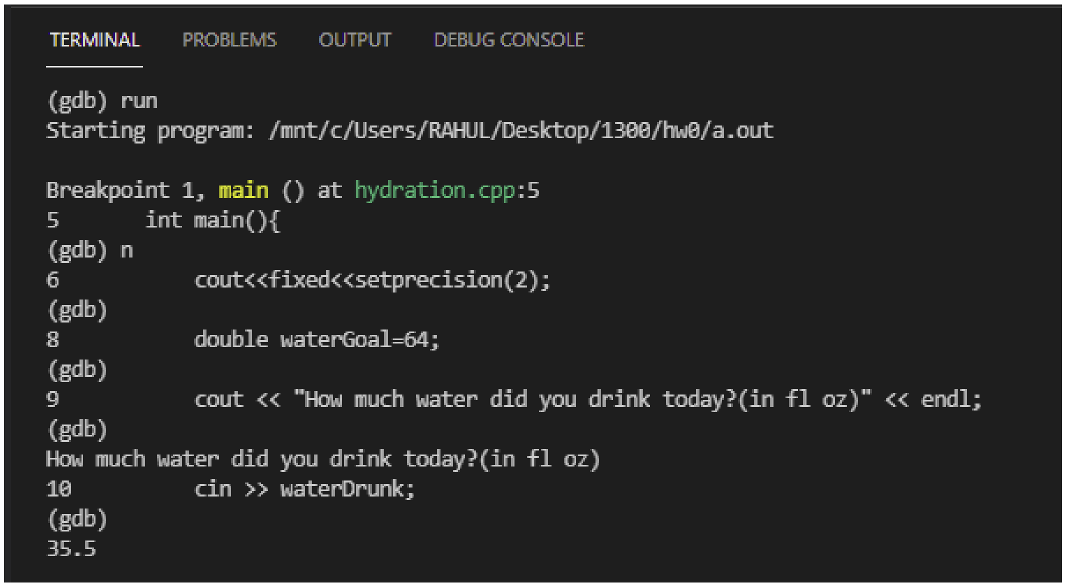
15. Print the variables now. They will have the correct values assigned to them.



TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
(gdb)
std::setprecision (__n=32767) at /usr/include/c++/9/iomanip:196
196       { return { __n }; }
(gdb)
main () at hydration.cpp:8
8           double waterGoal=64;
(gdb)
9           cout << "How much water did you drink today?(in fl oz)" << endl;
(gdb)
How much water did you drink today?(in fl oz)
10          cin >> waterDrunk;
(gdb)
35.5

Breakpoint 2, main () at hydration.cpp:12
12          if(waterDrunk<=32)
(gdb) info locals
waterDrunk = 35.5
waterGoal = 64
(gdb)
```

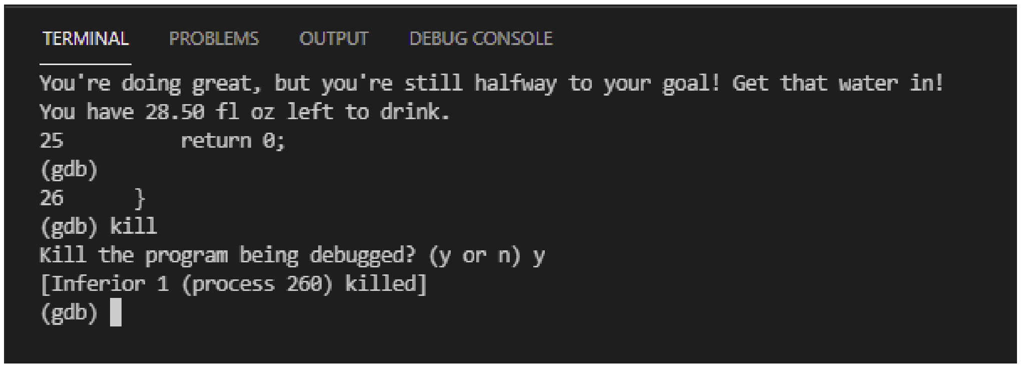
16. Re debugging the program with ‘step over’, we can see that the command steps over the setprecision function.


TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
(gdb) run
Starting program: /mnt/c/Users/RAHUL/Desktop/1300/hw0/a.out

Breakpoint 1, main () at hydration.cpp:5
5       int main(){
(gdb) n
6           cout<<fixed<<setprecision(2);
(gdb)
8           double waterGoal=64;
(gdb)
9           cout << "How much water did you drink today?(in fl oz)" << endl;
(gdb)
How much water did you drink today?(in fl oz)
10          cin >> waterDrunk;
(gdb)
35.5
```

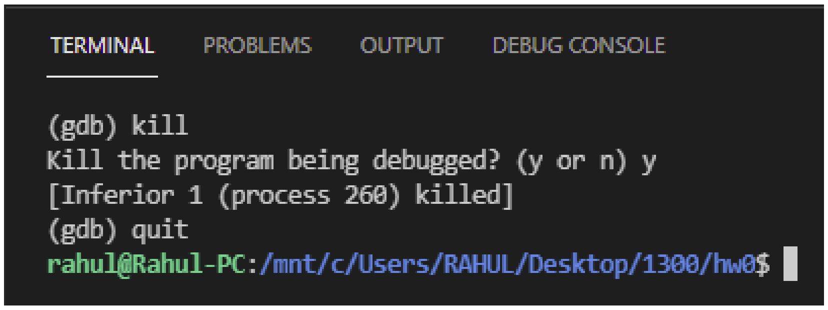
17. Finish execution of the program by either killing the debugging session or by quitting out of the gdb console.

18. Command for killing the session is `$ (gdb) kill`


TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
You're doing great, but you're still halfway to your goal! Get that water in!
You have 28.50 fl oz left to drink.
25          return 0;
(gdb)
26      }
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 260) killed]
(gdb) 
```

19. Command for quitting out of gdb is `$ (gdb) quit`


TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 260) killed]
(gdb) quit
rahul@Rahul-PC:/mnt/c/Users/RAHUL/Desktop/1300/hw0$ 
```