**Debugging**

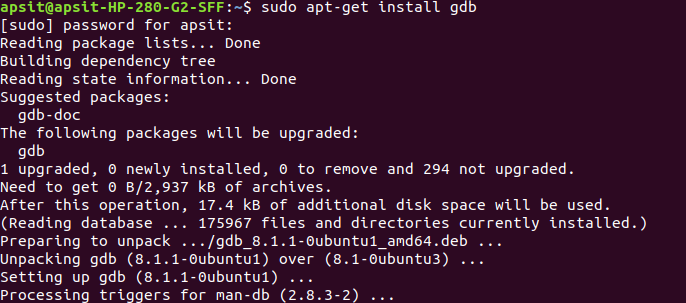
GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

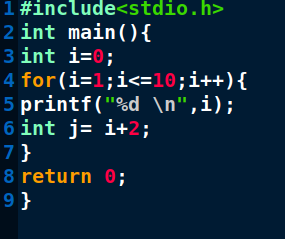
* Start your program, specifying anything that might affect its behavior.
* Make your program stop on specified conditions.
* Examine what has happened, when your program has stopped.
* Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

1.Install GDB

**sudo apt-get install gdb**



2.C program



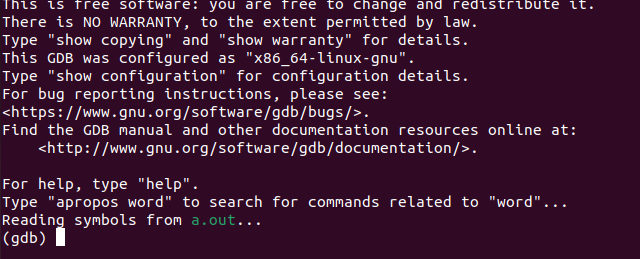
3. Compile the program specifically for GDB by using the `-g` flag:

**gcc -g first.c**

Using this flag means that GDB will preserve function and variable names and line numbers for the debugger. Normally this information is not retained in a compiled program.

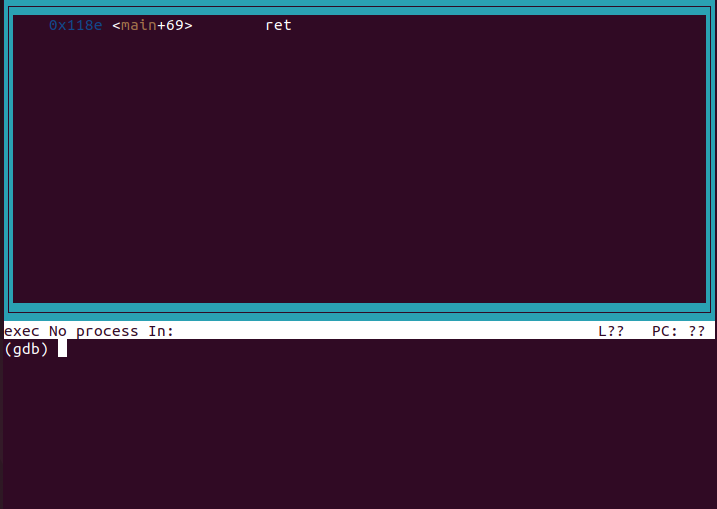
4.To start the GNU debugger running your program with command line arguments you can use:

**gdb a.out**



5. GDB will display the line of code at which the error occurred. However, it will be more helpful to see the entire source file. To do this use the Text User Interface (TUI) mode by typing:

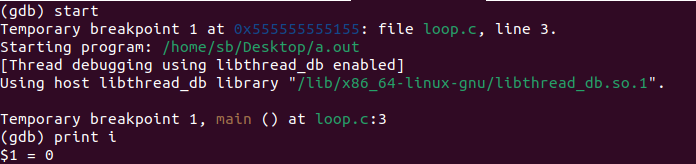
**layout next**



6.Examine some of the variable values to investigation what the issue

is. To print a variable value, use:

**print i**



**Other Useful command**

|  |  |
| --- | --- |
| h[elp] | Get help on gdb commands |
| h[elp] $ <$ cmd$ >$ | Get help on a specific gdb command |
| r[un] | Run to next [breakpoint](https://ccrma.stanford.edu/~jos/sasp/Envelope_Compression.html) or to end |
| s[tep] | Single-step, descending into functions |
| n[ext] | Single-step *without* descending into functions |
| fin[ish] | Finish current function, loop, etc. (useful!) |
| c[ontinue] | Continue to next breakpoint or end |
| up | Go up one context level on stack (to caller) |
| do[wn] | Go down one level (only possible after up) |
| l[ist] | Show lines of code surrounding the current point |
| p[rint]  <name> | Print value of variable called  <name> |
| p  \* <name> | Print what is pointed to by  <name> |
| p/x < name> | Print value of  <name> in hex format |