Why to Debug a Program?

When you write small programs (say 3-4 lines of code), and your program is not behaving correctly, you can debug your program inserting printf statements to print value of variables, manually trace the variables and values, find the statement that is written incorrectly, rectify and finally run the program successfully.

But, when you write larger programs, you should not debug your programs using printf statements. Instead use a debug tool, we use gdb tool in Linux environment.

What is gdb ?

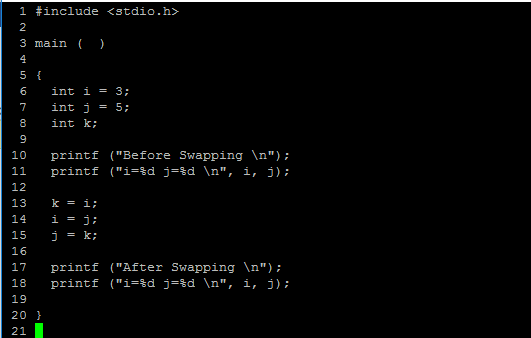
You can use GDB to debug programs written in C, C++. The purpose of a debugger such as GDB is to allow you to see what is going on ``inside'' a program while it executes.

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.

GDB is invoked with the shell command gdb. Once started, it reads commands from the terminal until you tell it to exit with the GDB command- quit.

First, let us take a simple program, say swap.c that swaps the values of two variables.



We compile the program using –g option.

gcc swap.c –g –o swap

The –g option compiles swap.c, embeds debugging information for the gdb tool to support debugging.

To start debugging our program swap , we type

gdb swap

Then gdb loads our program and prompts us to issue more commands. Now we are under debugger mode. Now we can stop the program at a particular line or function, step through each and every line if we want to, print the variables and much more.

Here is the list of commands to use for debugging:

|  |  |
| --- | --- |
| run | run the program. |
| break N | put a break point at line number N |
| break main | put a break point at function main |
| break function\_name | put a break at the start of the function |
| break N if I == 5 | break if the value of I == 5 at line number N |
| step | execute next line only and stop |
| step 3 | execute three lines and stop |
| continue | continue until next break point |
| info break | print the list of break points |
| finish | finish the current function |
| next | almost same as step, but won’t stop at functions unless a break point is set |
| watch variable | stops whenever the variable changes value, |
| delete | delete all breakpoints |
| delete N | delete breakpoint number N |
| clear | delete breakpoints |
| disable N | do not delete breakpoint N, just disable now |
| enable N | enable the breakpoint |
| backtrace | Show trace, functions, and print the stack |
| set variable variable= value | sets the value to the variable |
| set logging on | enable logs |
| set logging off | disable logs, Default name of file is gdb.txt |
| Set logging file filename | change the name of the current logfile |
| save breakpoints break\_filename | saves all breakpoints in a file |
| source break\_filename | loads all breakpoints |
| list line-number | print the source code around the line number |
| list function | print the source code around the function |
| quit | logout |
| gdb a.out core | in case a.out crashes with segmentation fault |
| print variable | prints the value of variable |
| print array@5 | will print the addresses of 5 cells |
| print array[0]@5 | will print the values the five cells |
| whatis variable | prints the type of the variable |
| where |  |