1 What is GDB?

Gdb (GNU Debugger) is a debugger for C (and C++). It allows you to inspect what the program is doing at a certain point during execution and print out the values of certain variables at that point, or step through the program one line at a time and print out the values of each variable after executing each line. It uses a command line interface.

Online manual can be found here: https://sourceware.org/gdb/current/onlinedocs/gdb/

2 Using GDB

2.1 Compiling a Program

The syntax for compiling a program:
`gcc [flags] <source files> -o <output file>`
For example,
`gcc -o hw ./hw1.c`
To prepare your program for debugging with gdb, you must compile it with the -g flag,
`gcc -g -o hw ./hw1.c`

2.2 Running Your Program

To execute your C program,
`./<output program name>`

2.3 Starting Up GDB

- To start gdb, just type "`gdb`" or "`gdb` *`executable`*" at the command line prompt. Gdb will give you a prompt that looks like this: `(gdb)` or `(db)`.
- To exit the program just type `quit` or `q` at the `(gdb)` prompt.
- If you didnt specify a program to debug, youll have to load it in now:
  `(gdb) file hw`
  Here, `hw` is your executable file name.

2.4 Commands

- `(gdb) help [command]`
  Use the "help" command with or without an argument if youre ever confused about a command or just want more information.
- `(gdb) list` *`linenumber`*
  or simply
  `(gdb) l` *`linenumber`*
  lists source code with line numbers.
- `(gdb) run`
  `run` starts the program running under gdb. You can give command line arguments to your program on the gdb command line the same way you would on the unix command line, except that you are saying run instead of the program name. You can even do input/output redirection:

```
(gdb) run < Input.txt
```

- `(gdb) break hw1.c:8`
  or simply
  `(gdb) b hw1.c:8`
  sets a breakpoint at line 8, of hw1.c. Now, if the program ever reaches that location when running, the program will pause and prompt you for another command. You can set as many breakpoints as you want, and the program should stop execution if it reaches any of them.

  Once youve set a breakpoint, you can try using the `run` command again. This time, it should stop where you tell it to (unless a fatal error occurs before reaching that point).

- `(gdb) continue`
  or
  `(gdb) c`
  proceeds onto the next breakpoint.

- `(gdb) step`
  or
  `(gdb) s`
  single-steps (executes just the next line of code) which gives you fine-grained control over how the program proceeds.

- `(gdb) next`
  or
  `(gdb) n`
  is similar to `step`, except that if the line about to be executed is a function call, then that function call will be completely executed before execution stops again, whereas with `step` execution will stop at the first line of the function that is called. That is, `next` treats a sub-routine as one instruction.

- `(gdb) print` *`expression`*
  or
  `(gdb) p` *`expression`*
  prints out the value of the expression.

- `(gdb) watch` *`expression`*
  tracks the value of specified expression at every step. Whenever the expression's value is modified, the program will interrupt and print out the old and new values.

- `(gdb) info breakpoints`
  shows information about all declared breakpoints.

- `(gdb) delete`
  deletes all declared breakpoints.

- `(gdb) delete` *`number`*
  deletes breakpoint numbered *number*.