Introduction to Debugging with gdb

Computer Science and Engineering
Seattle University

Invoking gdb

To use `gdb`, you must compile and link your program with the "`-g`" switch (other switches such as "`-Wall`" can be used too). For example:

```
g++ -g prog1.cpp
```

To start gdb, type (assuming a.out is the name of the executable):

```
gdb a.out
```

When gdb starts, your program is not actually running. To have it run, type:

`run` *optional-command-line-arguments*

You can also restart your program by typing "**`run`**" anytime there is a prompt. You do not need to retype the command line arguments unless you want to change them.

Setting and deleting breakpoints

The key to using a debugger is to place breakpoints in the program. A breakpoint is a point in the program where the debugger will suspend the execution of the program. While the program is suspended, the programmer can print out the values of variables and execute programs step by step using commands described later in this document.

The command to set a break point is:

`break` *place*

This creates a breakpoint at *place* - the program will stop when it gets there. The *place* field can be one of three things:

- The beginning of a function (example: "**`break swap`**" will stop at the beginning of the function **`swap`**)
- The line number of the current file (example: "**`break 20`**" will stop at line number 20)
- The file name and line number (example: "**`break board.cpp:64`**" will stop line 64 in `board.cpp`)

The current file is the file where the execution of the program currently is at. At the beginning of the program, the current file is the file that contains `main`.

To display all the current breakpoints:

`info break`

This will give a list of breakpoints with an ID number. This ID number can be used to disable, enable, or delete breakpoints using these commands respectively:

`disable` *id*	disables breakpoint id, can be re-enabled
`enable` *id*	enables breakpoint id
`delete` *id*	completely removes breakpoint id, cannot be re-enabled

Controlling the execution of the program

These commands control how much is executed:

`step`
Executes the current line of the program and stops on the next statement to be executed. If the statement includes a function call, the next statement will be the first statement of the called function.

`next`
Similar to step, except if the current line of the program contains a function call, it executes the function and stops at the next line.

`finish`
Continue until a breakpoint is hit or the end of the current function.

`continue`
Continues until a breakpoint is hit or the program stops.

Displaying information

These commands give you information about the current state of the program:

`print` *expression*
Prints the value of the expression (usually just a variable) in the program.

`display` *expression*
Prints the value of the expression every time you execute a step (using `step` or `next`).

`undisplay`
Stops the display of expressions.

`where`
Prints a chain of function calls that brought the program to its current place.

Other useful commands

`help` *command*
Provides a brief description of a GDB command or topic. The word "**`help`**", with no command, lists the possible topics

`quit`
Leave GDB.

Debugging tips

- When you encounter a run-time error with your program such as "segmentation fault" or "bus error", the first course of action is to invoke gdb. Run your program without any breakpoints. The debugger will automatically stop at the point where the error occurred indicating the file name and line number. If you stop in a system call, use the '`where`' command to see the last statement of your source code that was executed.

- gdb can also help with infinite loops or programs that seem to take a long time. While the debugger is running, hit Ctrl-C to stop the debugger (forcing a breakpoint). gdb will indicate where it stopped. Check out this code and see if you are in an infinite loop.

- For errors where the output doesn't give the proper value, you will need to narrow down where the error occurred. This can be done by setting breakpoints at function boundaries and verifying if each function is working correctly. Once you find the failing function, step through that function to find the error. It may be necessary to re-run the program several times before narrowing down the error. Just like program design, debugging is also done in a top-down fashion.

- Debugging is an art. Focus on spots of your code that you are least comfortable with first, but don't get "tunnel vision" and assume that the error is in that segment of code.