The GNU Debuger

For C++ Projects

Nathan Warner

Northern Illinois
University

Computer Science
Northern Illinois University
February 20, 2024
United States

Contents

Compile Your Program with Debugging Information

To compile your program with debugging information, you'll use the `-g` option with `g++`. This option tells the compiler to include extra debugging information in the executable, which GDB can use to provide more detailed information during debugging.

```
g++ -g program.cpp -o bin
```

Start GDB

To start debugging your program with GDB, you run GDB and pass the name of the executable as an argument:

```
gdb ./bin
```

Set Breakpoints

Before running your program, you might want to set breakpoints at certain lines of code or in specific functions so that execution will stop and allow you to inspect the state of your program. For example, to set a breakpoint at the **foo** function, you can use:

```
break foo
```

Note:-

If there are multiple functions with the same name (overloaded functions), GDB will ask which one you want to set the breakpoint on. You can specify by including the parameter types.

3.1 Line breakpoints

These are the most common type of breakpoints. You set them on a specific line of code. The program execution will pause **before** that line of code is executed.

```
break 20
break quicksort.cpp:20 # If you need to specify the file
```

3.2 Conditional breakpoints

These are line breakpoints with a condition. The debugger will only pause the program if the condition is true. This is useful for stopping the program in more specific scenarios, like when a variable reaches a certain value.

```
break 20 if x==5
```

Run Your Program

To start your program under GDB, use the run command. If your program requires command-line arguments, you can provide them after the run command:

```
run
```

Inspect the Call Stack

Once your program hits a breakpoint (or if you've manually paused it using the pause command), you can inspect the call stack using the backtrace command (or bt for short). This command shows you the function call stack at the current point of execution:

```
backtrace
```

Stepping Through the Code (Execution Control)

You can use various commands to step through your code:

- **step (or s):** Execute the next line of code, stepping into functions.
- **next (or n):** Execute the next line of code, stepping over functions.
- **continue (or c):** Continue execution until the next breakpoint or the program ends.
- **finish:** Runs until the current function is finished and returns to the caller.
- **kill:** Stops the execution of your program.

Inspecting Variables

To inspect the value of a variable, use the print command (or p for short):

```
print variable_name
```

7.1 Printing arrays

Printing an array in GDB can be done using the print command (p), but you'll need to specify how many elements of the array you want to print. GDB doesn't inherently know the size of your arrays (unless it's a static array whose size can be determined at compile time), so you have to tell it how many elements to print.

```
print *arr@N
```

Watching variables

To watch a variable, you use the watch command followed by the variable name. For example, if you want to watch a variable named counter, you would do:

```
watch counter
```

Exiting GDB

To exit GDB, use the quit command (or q for short).

```
q
```

Other commands

10.1 Breakpoints and Watchpoints

- **break [location] (or b [location]):** Sets a breakpoint at the specified location. The location can be a line number, function name, or address.
- **watch [expr]:** Sets a watchpoint for an expression (usually a variable). The program stops whenever the value of the expression changes.
- **info breakpoints:** Lists all breakpoints and watchpoints.
- **delete breakpoints [number]:** Deletes the breakpoint or watchpoint with the specified number. Without a number, deletes all breakpoints/watchpoints.
- **disable breakpoints [number]:** Disables the specified breakpoint or watchpoint without removing it.
- **enable breakpoints [number]:** Enables a previously disabled breakpoint or watchpoint.

10.2 Inspecting the Program

- **print [expr] (or p [expr]):** Evaluates and prints the value of an expression. The expression can involve variables, function calls, etc.
- **info locals:** Displays local variables of the current frame.
- **info args:** Shows the arguments of the current function.
- **backtrace (or bt):** Displays the call stack, showing which functions have been called to reach the current point.
- **list (or l):** Lists the source code. By default, it shows ten lines around the current line.
- **frame [number]:** Selects a frame from the call stack. Without a number, it shows the current frame.

10.3 Modifying the Program State

- **set var [variable]=[value]:** Changes the value of a variable to the specified value.
- **jump [location]:** Transfers execution to the specified location (line number or function).