GDB (GNU Debugger)

Contents

[1. Introduction to GDB 2](#_Toc202376929)

[Compiling for Debugging 2](#_Toc202376930)

[Starting GDB 3](#_Toc202376931)

[Basic Commands 3](#_Toc202376932)

[2. Breakpoints 4](#_Toc202376933)

[List source code: 5](#_Toc202376934)

[3. Program Execution Flow 6](#_Toc202376935)

[4. Passing Command-Line Arguments to Executables 7](#_Toc202376936)

[Using --args when launching GDB 7](#_Toc202376937)

[Using run command with arguments after entering GDB 7](#_Toc202376938)

[Using set args before running 8](#_Toc202376939)

[Summary Table 8](#_Toc202376940)

[5. Examining Execution 9](#_Toc202376941)

[Print Values 9](#_Toc202376942)

[Examine Memory 10](#_Toc202376943)

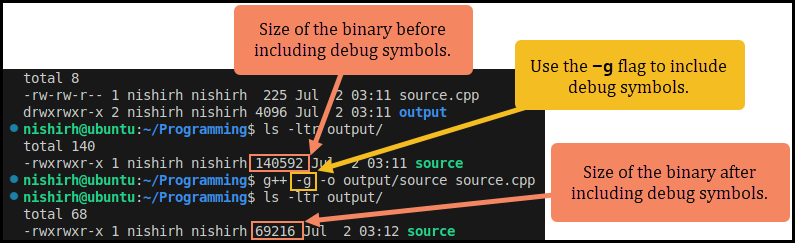
[Type Inspection 10](#_Toc202376944)

[Stack Traces & Debugging Crashes 11](#_Toc202376945)

# Introduction to GDB

* **GDB** stands for *GNU Debugger*.
* It helps in debugging **binary executables** created during compilation.
* Main Features:
  + Step-by-step execution.
  + Setting breakpoints (function/line).
  + Conditional execution control.
  + Examining variables and memory.
  + Viewing the call stack.

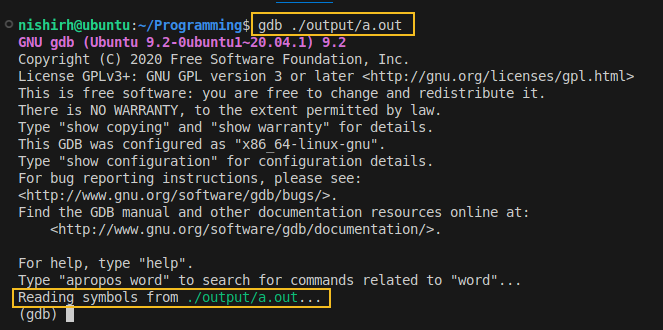
## Compiling for Debugging

* Use the -g flag to include **debug symbols**:
* This embeds:
  + Symbol names (variables/functions)
  + Types
  + File names
  + Line numbers
* Increases executable size due to metadata.

## Starting GDB

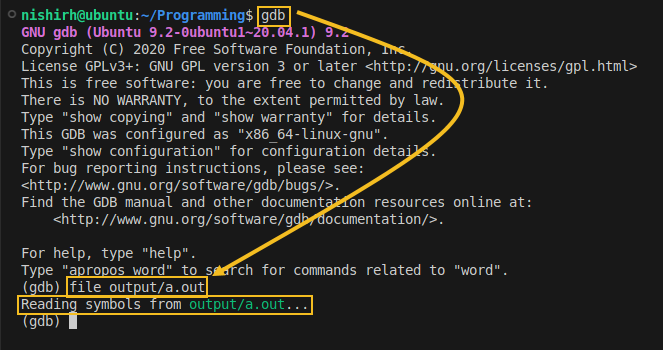
* Two ways to start gdb for debugging.

1. Direct:



gdb ./a.out

1. Load inside gdb:



Note: To start GDB in quite mode, use -q option.

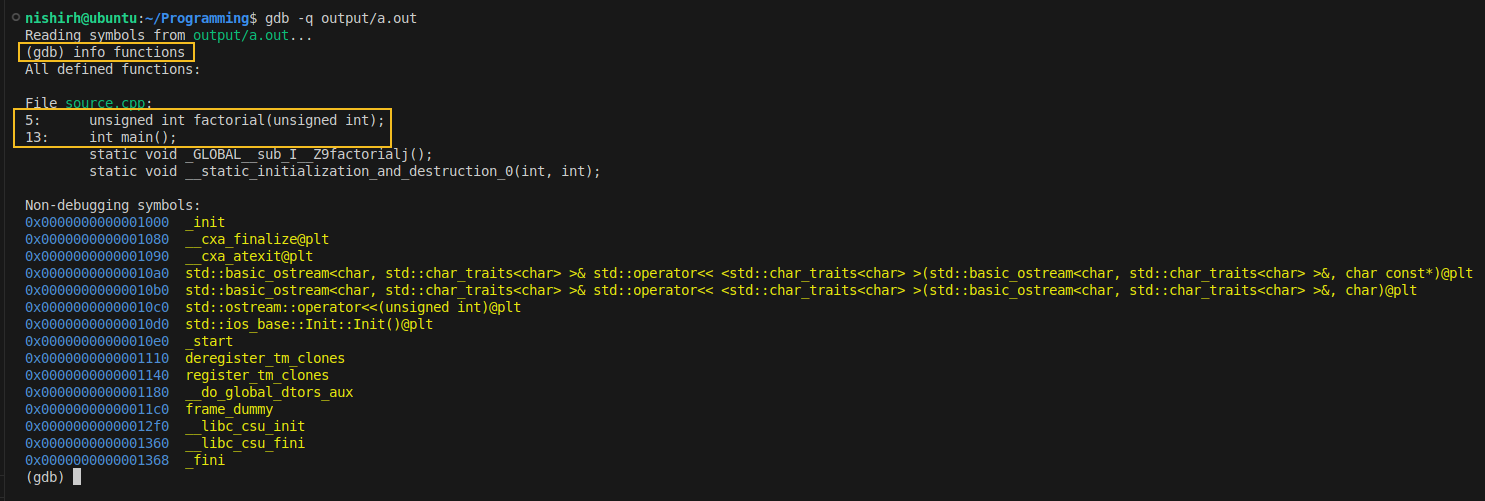
## Basic Commands

* GDB is a command line interface.

|  |  |  |
| --- | --- | --- |
| Command | Shortcut | Description |
| run | r | Start program execution |
| quit | q | Exit GDB |
| help [command] |  | Get help for commands |

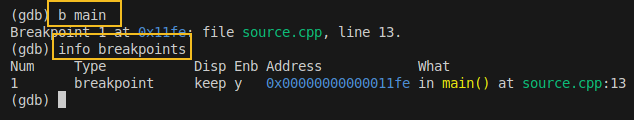
# Breakpoints

* The main purpose of GDB is to stop, observe and proceed.
* Breakpoints can be used to stop the program in the middle at a designated point.
* The simplest way of keeping a breakpoint is either using function name or a line number.
* Syntax:
  + By function: break main
  + By line number: break 12
  + Short: b main or b 12
* To find out what functions are available, use

info functions

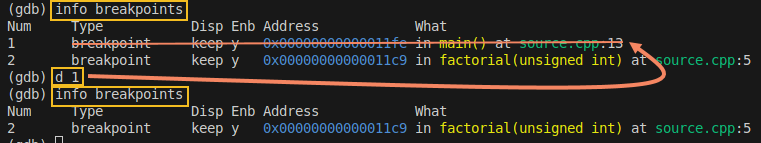
* To list all the breakpoints, use

info breakpoints



* To delete a breakpoint, use

delete 1 or d 1



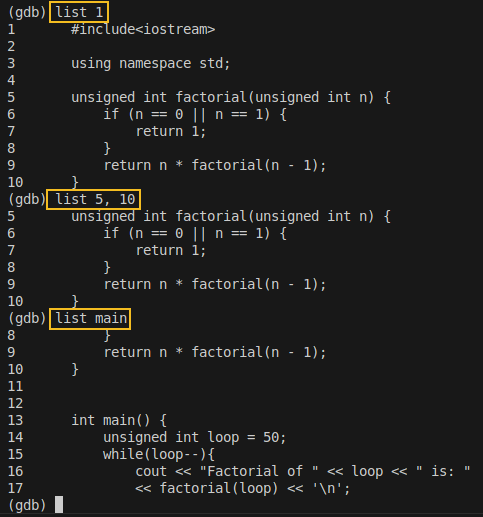
## List source code:

* To list the source code while debugging, use

list          # next 10 lines

list 1,10     # specific range

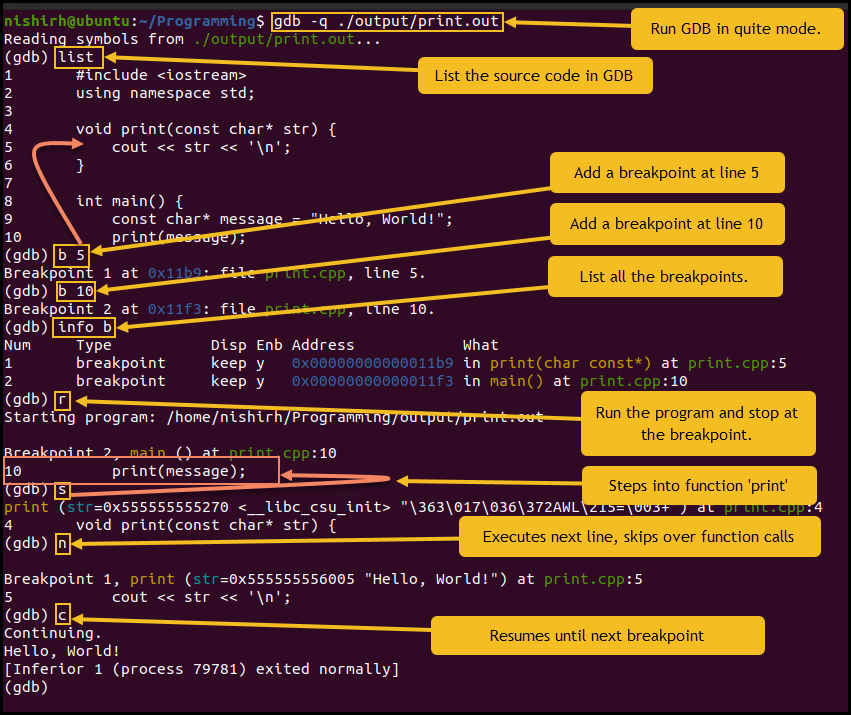
list main     # around function



# Program Execution Flow

* Whenever a breakpoint is hit during execution, the program stops.
* At that point, you can examine variables, memory, the call stack, and step through your code using GDB commands.

|  |  |
| --- | --- |
| Command | Description |
| run | Starts program (from beginning) |
| continue / c | Resumes until next breakpoint |
| next / n | Executes next line, skips over function calls |
| step / s | Steps into function |
| finish | Runs until current function returns |
| [Enter] | Repeats last command |



# Passing Command-Line Arguments to Executables

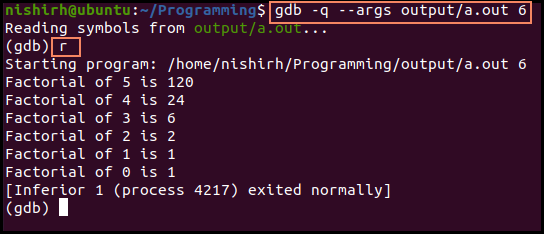
* When debugging a C or C++ program with GDB, if your **executable expects command-line arguments**, there are **multiple ways to provide them**.

## Using --args when launching GDB

* This method allows you to pass the arguments while starting GDB.

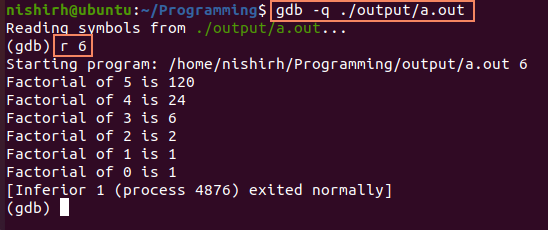
gdb --args ./my\_program source\_code 6

* The --args tells GDB to treat everything after it as the executable and its arguments.



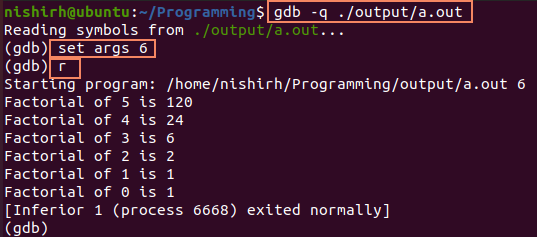
## Using run command with arguments after entering GDB

* If you've already started GDB and loaded your executable, then you can pass arguments using the run command.



## Using set args before running

* You can also set the arguments first, then run:



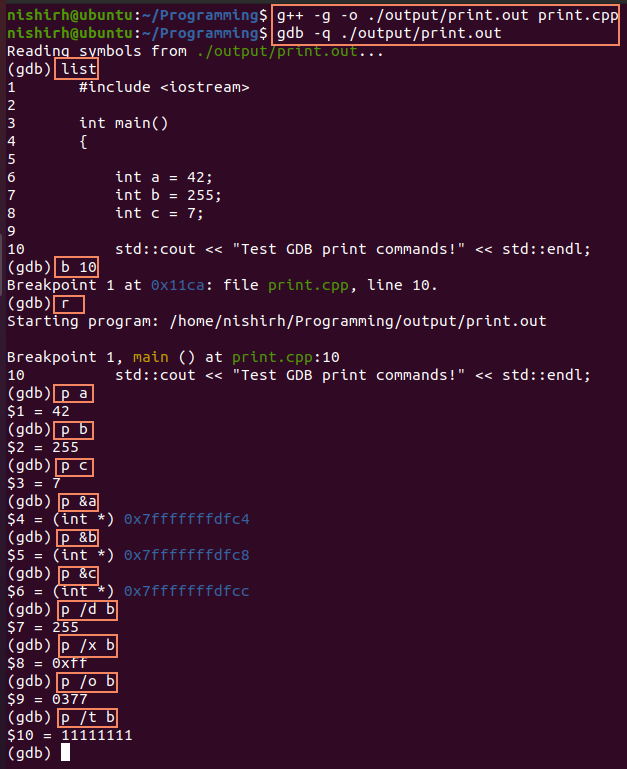
## Summary Table

|  |  |  |
| --- | --- | --- |
| Method | Command | When to Use |
| Using --args | gdb --args ./my\_program arg1 arg2 | At GDB launch |
| Using run | gdb ./my\_program → run arg1 arg2 | After loading GDB |
| Using set args + run | gdb ./my\_program → set args arg1 arg2 → run | Set once, reuse multiple run calls |

# Examining Execution

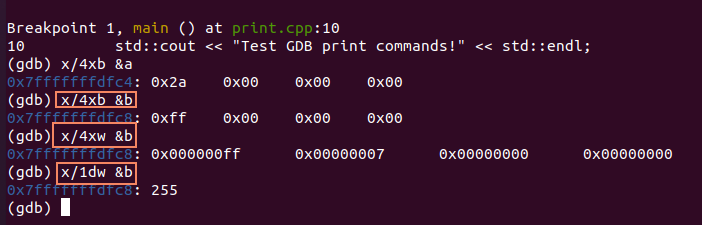
## Print Values

* print var or p var
* Show address: p &var
* Format options:
  + Decimal: p /d var
  + Hex: p /x var
  + Octal: p /o var
  + Binary (if supported): p /t var



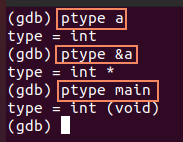
## Examine Memory

* Syntax: x /<count><format><size> address
  + e.g., x/4xb &i → 4 hex bytes at i
* Useful format specifiers:
  + x – Hex
  + d – Decimal
  + s – String
  + b – Byte
  + w – Word (4 bytes)



## Type Inspection

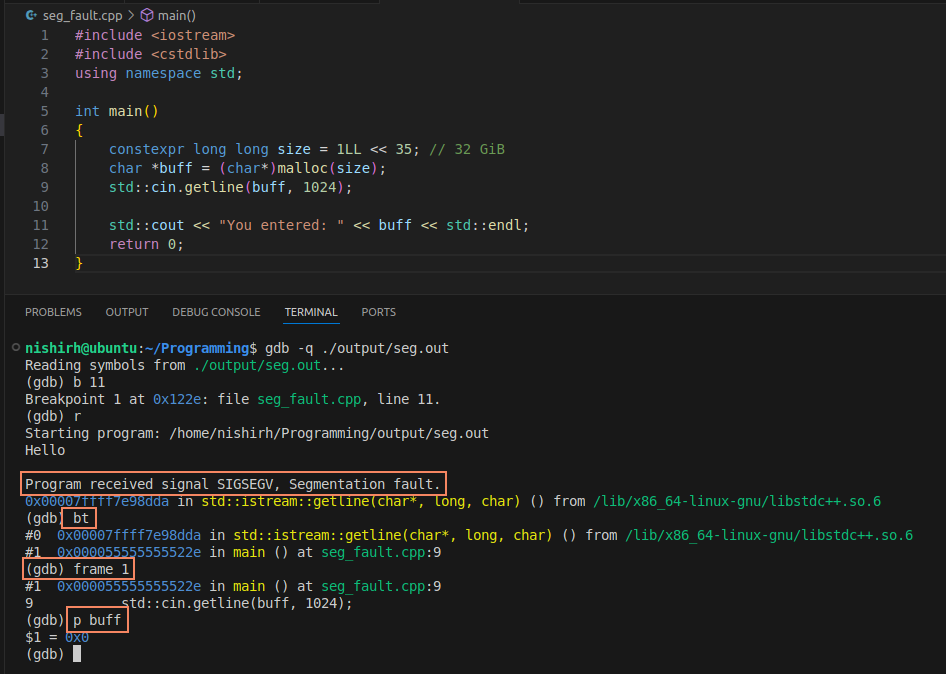
* ptype var → Show type of variable
* ptype &var → Pointer type
* ptype main → Function signature



## Stack Traces & Debugging Crashes

**Scenario: Segmentation fault**

* Use backtrace (bt) to view call stack
* Use frame <n> to switch stack frame
* Use list to view surrounding source
* Use print var to inspect suspected variables



# Call Stack & Frames

* **Call Stack**: Stack of function calls.
* **Frame**: Each function call creates a *frame* in the stack.
* **Frame info**:
  + Return address
  + Local variables
  + Arguments

**Commands:**

* backtrace (or bt): Shows current call stack.
* frame n: Switch to specific frame number.
* info frame: Details like instruction pointer, stack pointer, local variable addresses.
* list: Displays source code of current frame.
* print var: Inspects local variable of selected frame.

## INFO Commands Summary

|  |  |
| --- | --- |
| Command | Description |
| info functions | List all functions in symbol table |
| info variables | List global & static variables |
| info locals | List local variables of current frame |
| info args | Function arguments in current frame |
| info breakpoints | Show all breakpoints and watchpoints |
| info registers | Show all CPU register values |
| info register RAX | Show specific register value |

# Conditional Breakpoints

* Stop execution only **when a condition is met**.
* **Two Ways:**
  + **Add with condition directly**:

break 15 if i == 5

* + **Apply condition after setting**:

break 15

condition <breakpoint-number> i == 5

* Useful for skipping unnecessary iterations (e.g., in loops).
* info breakpoints: Show all breakpoints with their conditions.

## Watchpoints

* **Used for variables**, not line numbers.
* Execution stops when:
  + A value is **read**, **written**, or **both** (depending on watch type).

|  |  |  |
| --- | --- | --- |
| Watch Type | Trigger Condition | Command |
| watch var | On write to variable | watch x |
| rwatch | On read from variable | rwatch x |
| awatch | On read/write to variable | awatch x |

* Must be in scope to apply.
* Use disable n / enable n to toggle watchpoint.
* Show all: info breakpoints

# TUI (Text User Interface)

* Show source code while stepping through.
* Start with:

gdb -tui ./a.out

* Inside GDB:
  + Toggle TUI: Ctrl + X followed by A
  + Refresh TUI: Ctrl + L

**Benefits:**

* Live source code view
* Visual breakpoints
* Highlight current line

# Logging GDB Output

* Save session output to a file.

set logging on              # Start logging to gdb.txt

set logging off             # Stop logging

set logging file mylog.txt  # Custom file name

# Attaching to a Running Process

* Attach GDB to a live process by PID.

**Steps:**

1. Get PID:

shell ps -ef | grep myapp

1. Attach:

attach <pid>

1. Detach (resume normal execution):

detach

* Requires root permissions (sudo gdb)

# GDB start Command

* Shortcut for:

break main

run

* Just use:

Start

## Auto-Run Commands on Breakpoint

* Automate actions like printing variables when breakpoint hits.

break 15

commands 1

  print i

  continue

end

# Summary of Common GDB Features

|  |  |
| --- | --- |
| Category | Commands / Concepts |
| **Breakpoints** | break, info breakpoints, delete |
| **Run control** | run, continue, next, step, finish, start |
| **Call Stack** | backtrace, frame n, info frame |
| **Variable Inspect** | print, info locals, info args, info variables |
| **Memory Inspect** | x, x/s, x/4xb, etc. |
| **Assembly View** | disassemble main |
| **Registers** | info registers, p $rax |
| **Conditional BP** | break 12 if i == 5, condition |
| **Watchpoints** | watch, rwatch, awatch |
| **Logging** | set logging on, set logging file log.txt |
| **TUI Mode** | gdb -tui, Ctrl + X, A, Ctrl + L |
| **Attach/Detach** | attach <pid>, detach |
| **Startup Helper** | start (shortcut for break main + run) |