**Capstone Project: Debugging C Programs with GDB**

**Objective:** The goal of this capstone project is to showcase the use of **GNU Debugger (GDB)** to debug, analyze, and troubleshoot C programs. This project will highlight your understanding of debugging techniques, memory management, and problem-solving skills through practical examples and structured tasks.

**Project Outline:**

**1. Project Goals**

* Learn the fundamental operations of GDB for debugging C programs.
* Develop C programs that include intentional bugs or areas for debugging.
* Use GDB to inspect variables, trace execution, and fix issues.
* Showcase debugging techniques, such as breakpoints, backtraces, and memory inspections.

**2. Tools and Setup**

* **GCC Compiler**: Compile programs with debug symbols (-g flag).
* **GDB**: GNU Debugger for analyzing and debugging the compiled binaries.
* **Editor/IDE**: Use any text editor (e.g., VSCode, Vim) or an IDE with GDB integration.
* **System Requirements**: Linux or a compatible environment where GCC and GDB are installed.

**3. Capstone Program Modules**

The capstone will be divided into specific modules to cover various debugging scenarios.

**Capstone Modules:**

**Module 1: Debugging Basics**

**Description**: Write a simple C program with logical and runtime errors to demonstrate basic debugging using GDB.

**Debugging Tasks**:

* Compile
* Launch GDB:
* Set breakpoints:
* Run the program:
* Inspect the error using backtrace:
* Examine variable values:

**Module 2: Memory Errors with Valgrind and GDB**

**Description**: Debug a program with memory leaks and invalid memory access using GDB.

**Debugging Tasks**:

* Compile with gcc program
* Run the program in GDB
* U detect memory issues.
* Use GDB commands:
  + run to start execution.
  + watch to monitor variable changes.
  + info locals to view local variables.

**Module 3: Debugging Segmentation Faults**

**Description**: Analyze and fix a segmentation fault in a C program.

**Debugging Tasks**:

* Compile and debug:
* Set breakpoints at the dereference line.
* Inspect the value of ptr using print ptr.
* Use bt to trace the cause of the segmentation fault.

**Module 4: Using Core Dumps for Post-Mortem Debugging**

**Description**: Create and analyze core dumps for debugging crashed programs.

**Steps**:

1. Enable core dumps:
2. Compile and run a crashing program:
3. Locate the core dump (core file) and analyze it:
4. Use bt and list to locate the error.

**Module 5: Multithreaded Debugging**

**Description**: Debug a multithreaded program with race conditions.

**Debugging Tasks**:

* Compile with gcc
* Use gdb ./race to debug.
* Analyze threads:
  + Use info threads to list threads.
  + Switch threads: thread <id>.
  + Step through threads: step, next.
* Investigate the race condition and propose fixes (e.g., using a mutex).