Remember to bring this along to your lab. Prepare your report before attending the lab!

CS2100: Computer Organisation

Lab #1: Debugging using GDB

(Week 3: 2 Sep – 6 Sep 2024)

Name: _____ Student No.: _____

Lab Group: _____

Objective:

You will learn how to use **GDB** to debug a C program.

Preparation (before the lab): Please refer to the setup document on Canvas.

Procedure:

1. Download the file **`lab1.c`** from Canvas "Labs page".

2. Enter your Linux environment. (**`lima`** in MacOS, **`wsl`** in Windows)

3. Compile **`lab1.c`** with **`gcc`** using the following command: **`gcc -g -o lab1 lab1.c`**

4. What is the purpose of the flags "**`g`**" and "**`o`**" in gcc?

5. Execute the program you just compiled using the command: **`./lab1`**. What is the error encountered? (if any)

   Answer: _____

6. Start the GDB debugger by using the command: **`gdb lab1`**

7. To run the program in GDB, you can use the command **`run`** at the GDB prompt**.** This will run the whole program without any pause. Type **`run`** to execute the program.

8. To get into the debug mode use the **`start`** command

- You can use the **`list`** command to view the source code at any point
- You can also use **`layout src`** and **`layout asm`** commands to view source code and assembly code in a split screen.

9. The **breakpoint** command puts an intentional pause in the program execution. Once paused, you can inspect the variable values and resources at that point in the program execution. Multiple breakpoints can be set. Set a breakpoint in GDB at any line number using the command:
   `> break` <lineNumber>
   or
   `> b` <lineNumber>

   Example: **`> break 6`**

   This sets a breakpoint at line 6. Now if you run the program, **it will pause at line 6.** You can continue execution (till the next breakpoint or end) using the **`continue`** (or '**`cont`**' in short) command.

   Which line(s) would you set the breakpoint(s) at to debug the earlier error?

   Answer: _____

10. The **`step`** command is used to carry out step-by-step execution of the program. You can *step* through the program using the following command:

    `> step`
    This will execute only the next line of code, or
    `> step` <numberOfLines>

    Example "**`> step 3`**" will execute next three lines of code

- You can "switch on" display of the associated assembly code related to the instruction being executed using the command:
  `set disassemble-next-line on`

11. At every step (or breakpoint) you can view a variable value using the **`print`** command:

    `> print a`

    You can view all local variable values using the command:

    `> info locals`

    What are the values of variables **`c`** and **`d`** **at the start of line 8** (*before executing line 8*)**?**

    Answers: _____

12. You can view the register values at any step or breakpoint using this command:

    `> info registers`

13. Besides breakpoints, another very useful facility of GDB is **watchpoint**. While a breakpoint pauses execution at a particular point in the code (i.e., "reach line x, pause execution"), a watchpoint will pause execution when a particular condition is satisfied (i.e., "a certain condition met, pause execution"). For example,

    > **`watch (i==2)`**

    will pause execution whenever the condition, in this case "**`i==2`**" is satisfied. The condition is written using the C syntax.

    Do note that the variable has to be in scope. This may mean that you have to first set a breakpoint to where the (local) variable comes into scope, then set a watchpoint. Otherwise, GDB may complain "no idea what variable i you are talking about".

    You can see what are the watchpoints set by:

    > **`info watchpoints`**

    and remove them by

    > **`delete`**
    (No argument means all watchpoints will be deleted.) Or, delete specific watchpoints by:

    > **`delete`** <corresponding watchpoint number obtained via "info watchpoints">

14. You can stop the debugging by using the **`stop`** command. To quit GDB, use the **`quit`** command.

15. Debug and modify **`lab1.c`** to carry out four arithmetic operations (+, -, /, *) and print the days of the week. The output of the program should look as follows:

```
Arithmetic operations:
a+b = 110
a-b = 90
b/a = 0
a*b = 1000

Days of the week:
Day[0] = Monday
Day[1] = Tuesday
Day[2] = Wednesday
Day[3] = Thursday
Day[4] = Friday
Day[5] = Saturday
Day[6] = Sunday
```

    Show your labTA the output of your corrected program.

16. Submit this report to your labTA at the end of the lab. You do not need to submit the corrected program. You are not to email the report to your labTA.

Marking Scheme: Report – 6 marks; Correct output – 4 marks; Total: 10 marks.