CS2100: Computer Organisation
Lab #1: Debugging using GDB
(Week 3: 28 August – 1 September 2023)
[This document is available on Canvas and course website https://www.comp.nus.edu.sg/~cs2100]

Name: ___Neo Haowei_____   Student No.: ___A0264683U_____

Lab Group: ___14___

SPECIAL NOTE FOR APPLE SILICON MACOS USERS

GDB does not work on MacOS on Apple Silicon (M1 or M2) MacBooks. If you are using an M1 or M2 based MacOS system, you have two choices:

i)   Install Parallels and Ubuntu, then run gdb on Ubuntu. However, as Parallels is (expensive) paid software, this may not be a viable option for you. If you are interested to use Parallels, there is student pricing available: https://www.parallels.com/landingpage/pd/education/

ii) Use LLDB instead. The commands aren't exactly the same, but you should be able to accomplish the steps below using equivalent commands, though it will be harder work for you. You can find an LLDB tutorial here: https://lldb.llvm.org/use/tutorial.html

GNU Debugger (GDB) https://www.gnu.org/software/gdb/

A debugger is used to analyze program execution in a step-by-step and detailed manner. It is used to find bugs in a program. Using a debugger, we can execute a program partially and view the status of the variables and resources being used the program to identify any discrepancies. **GDB** is an open source, freely available debugger which can be used for multiple languages.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behaviour.

- Make your program stop on specified conditions.

- Examine what has happened, when your program has stopped.

- Change things in your program; so that you can experiment with correcting the effects of one bug and go on to learn about another.

GNU Compiler Collection (GCC)

GCC is an open-source compiler system used to compile C/C++ programs: https://gcc.gnu.org/

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

Preparation (before the lab):

1. Install GDB:
   a. **Ubuntu:** sudo apt-get install gdb
   b. **OSX:** brew install gdb (Does not work on Apple Silicon. See special note above.)
   c. **Windows:**
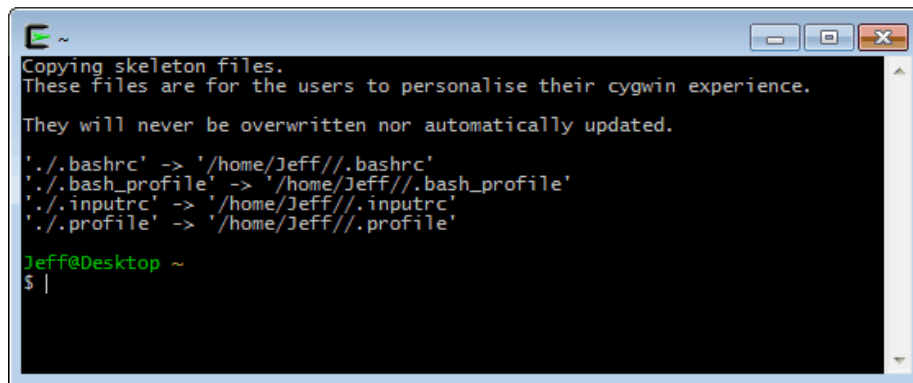   - Download Cygwin (https://cygwin.com/install.html)
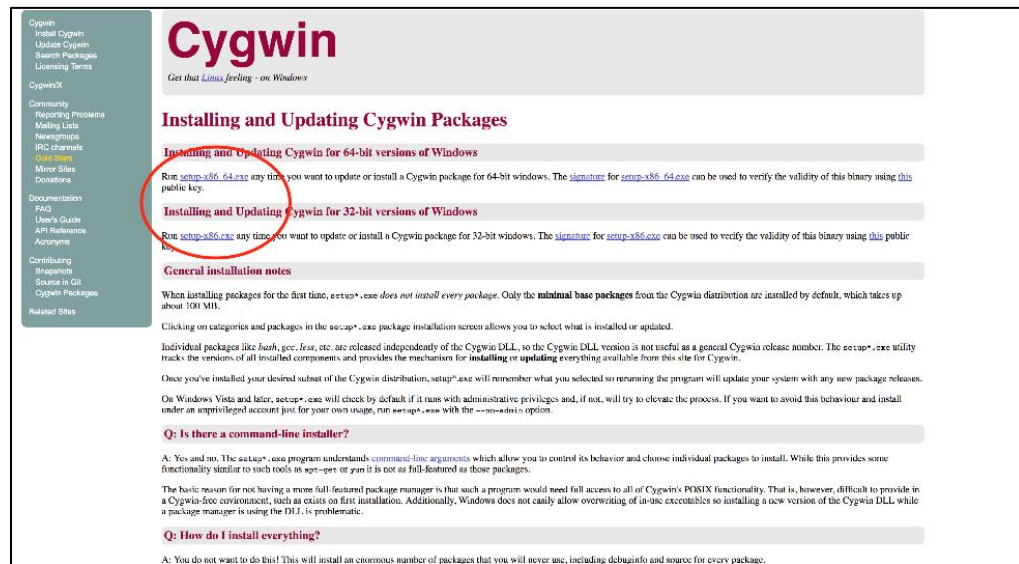
   Note: The installer will ask you to choose a mirror to download from. If possible, choose https://download.nus.edu.sg. If this is not available, choose an Australian or US server as these are generally faster.

      - Select **develop** packages during installation:
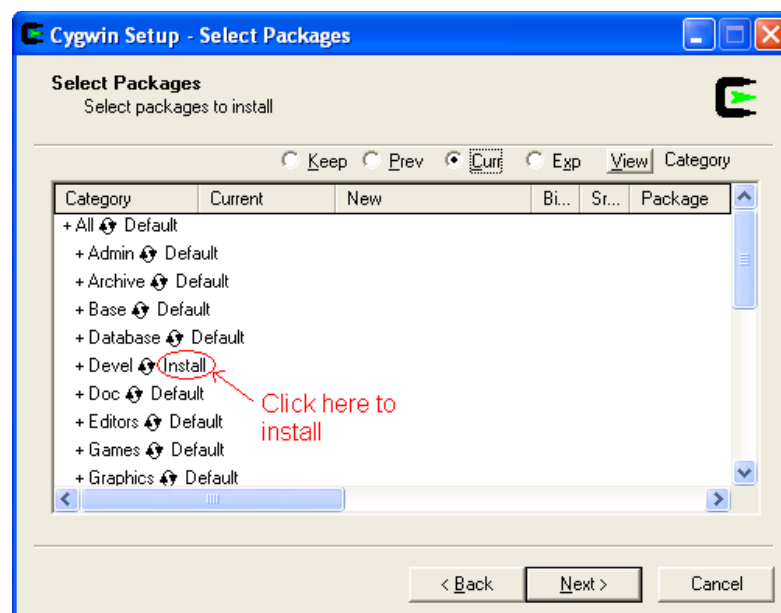
- Start Cygwin terminal from the start menu:
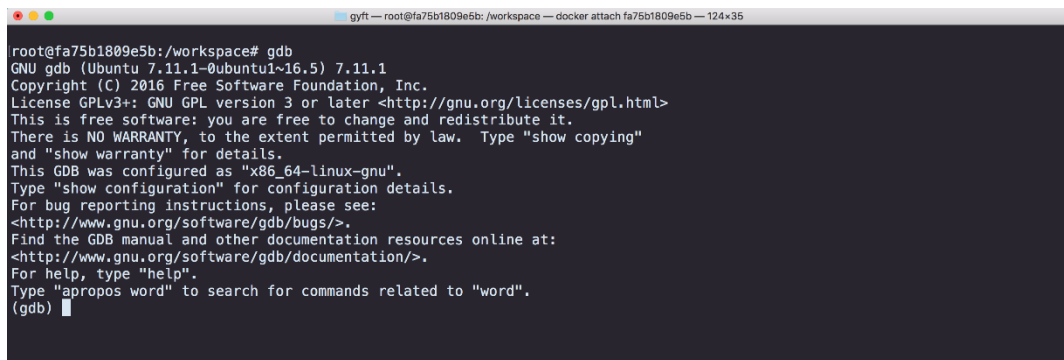
2. Starting GDB (all platforms): Type GDB in the terminal (Cygwin terminal for windows users). **Help** command lists the help and **quit** command exits GDB.

Procedure:

1. Download the file **lab1.c** from Canvas.

2. Compile **lab1.c** with **gcc** using the following command: **gcc –g –o lab1 lab1.c**

3. What is the purpose of the flags "g" and "o" in gcc?

> -g: Include debugging information in the binary.
>
> -o: put the result in the specified file instead of the default filename (abc.o for object files, a.out for linked executable files).
>
> -o lab1: Specify the name of the output binary file as "lab1".

4. Execute the program you just compiled using the command: **./lab1** (for windows type: **lab1**). What is the error encountered (if any!)?

Answer: _____Segmentation fault_____

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

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

7. 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.

8. A **breakpoint** is a command to put an intentional pause in the program execution to inspect the variable values and resources in the program. You can set multiple breakpoints in a program. In GDB you can put a breakpoint at any line number using the command:

   > break lineNumber
   or
   > b lineNumber

   Example: This will put a breakpoint on line 6 **> break 6**

   Now if you run the program, **it will pause at line 6.** You can continue execution (till end or the next breakpoint) using the **continue** command.

   Which line(s) will you set the breakpoint(s) in?

   Answer: _____28_____

9. A **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

   E.g. **> 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**

10. 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: ___c = 20 , d = 0_____

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

    > info registers

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

13. 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
    ```

    The code **lab1.c** is also shown on the next page for your reference.

    (**Note:** Due to compiler quirks on some platforms – particularly on MacOS – the program may not crash as expected. Nonetheless it is still faulty and needs fixing, use the commands in gdb to find these faults.)

    Show your labTA the output of your corrected program.

14. Submit this report to your labTA at the end of the lab. You do not need to submit the corrected program. You are <u>not</u> to email the report to your labTA.

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

Program lab1.c

```c
#include <stdio.h>

int main(void){
    int a = 10;
    int b = 10;
    int c = a+b;
    int d = a-b;
    int e = a/d;
    int f = a*b;
    int i;
    char *day[7] = {
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday",
        "Sunday"
    };

    printf("Arithmetic operations:\n");
    printf("a+b = %d\n", c);
    printf("a-b = %d\n", d);
    printf("b/a = %d\n", e);
    printf("a*b = %d\n\n", f);

    printf("Days of the week:\n");
    for (i=0; i<8; i++) {
        printf("Day[%d] = %s\n", i, day[i]);
    }

    return 0;
}
```