CS2100: Computer Organisation

Lab #1: Debugging using GDB

(3 & 6 February 2020)

[This document is available on LumiNUS and module website http://www.comp.nus.edu.sg/~cs2100]

Name: _____ Student No.: _____

Lab Group: _____

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.

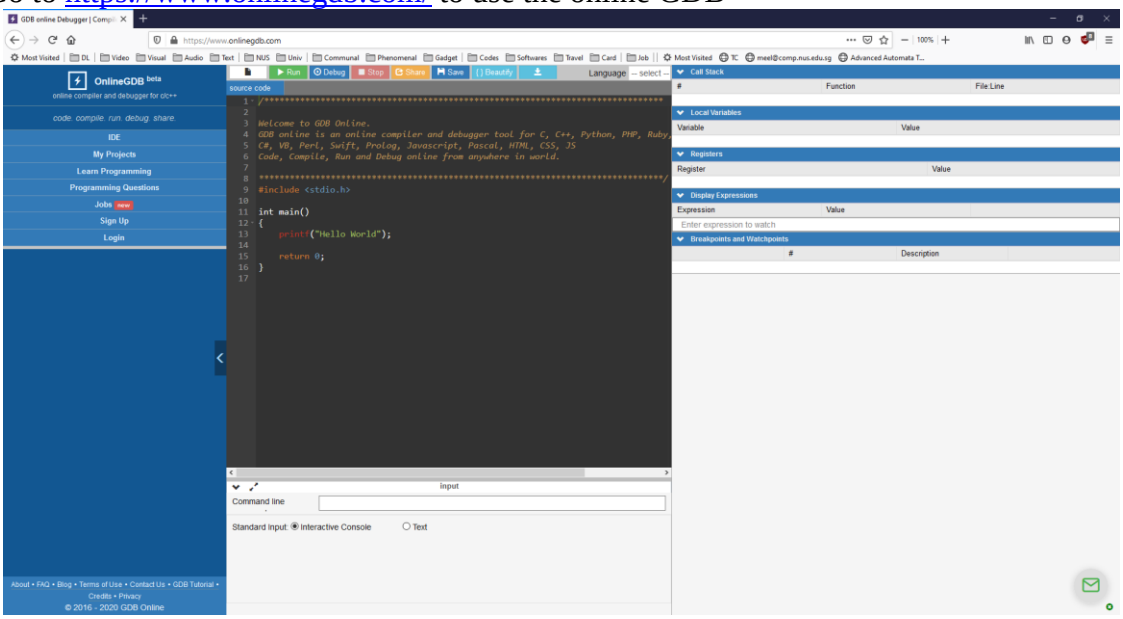
GNU Compiler Collection (GCC)

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.

Procedure:

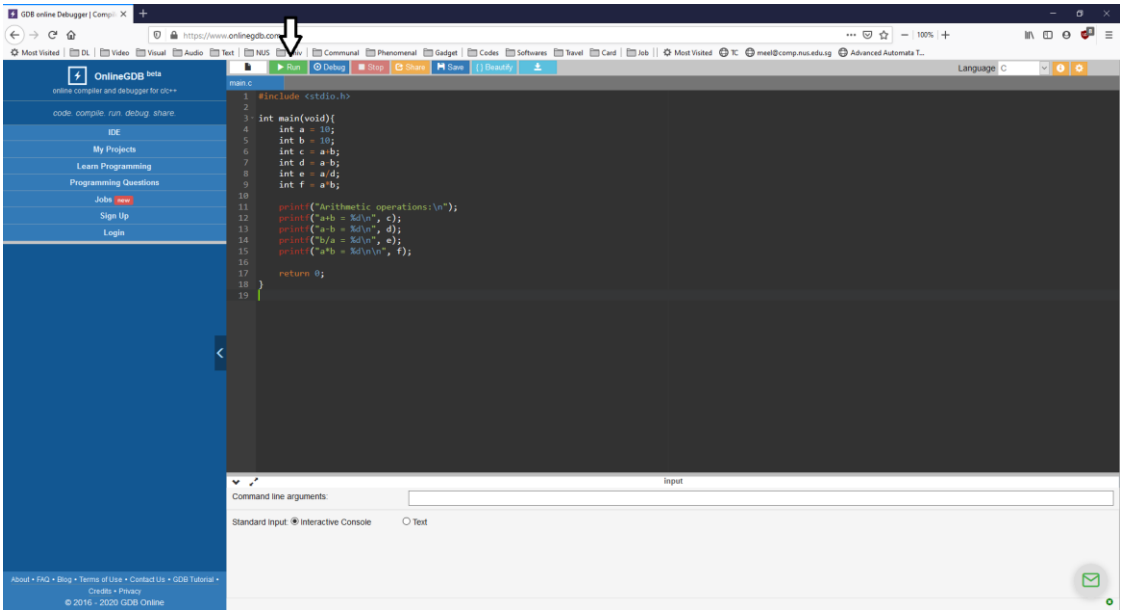
1. Go to https://www.onlinegdb.com/ to use the online GDB



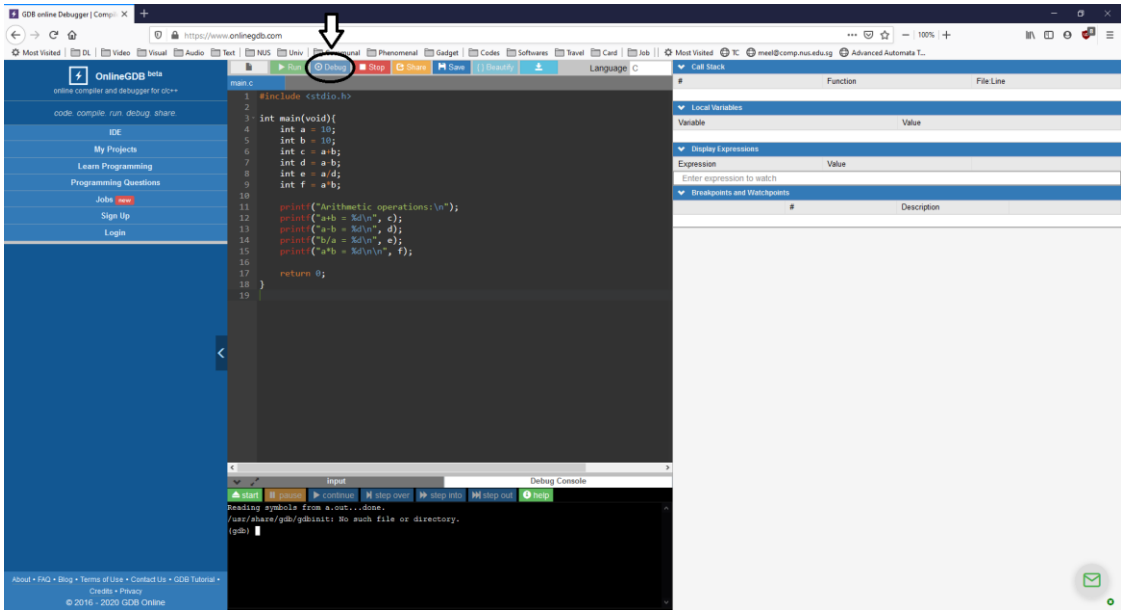
2. Download the file **lab1a.c** from the CS2100 website "Labs page": http://www.comp.nus.edu.sg/~cs2100/3_ca/labs.html

3. Copy the content of the file and select C as your language from the top-right dropdown. Run the program by clicking the "Run" button. What is the error encountered (if any!)?

   Answer: _____

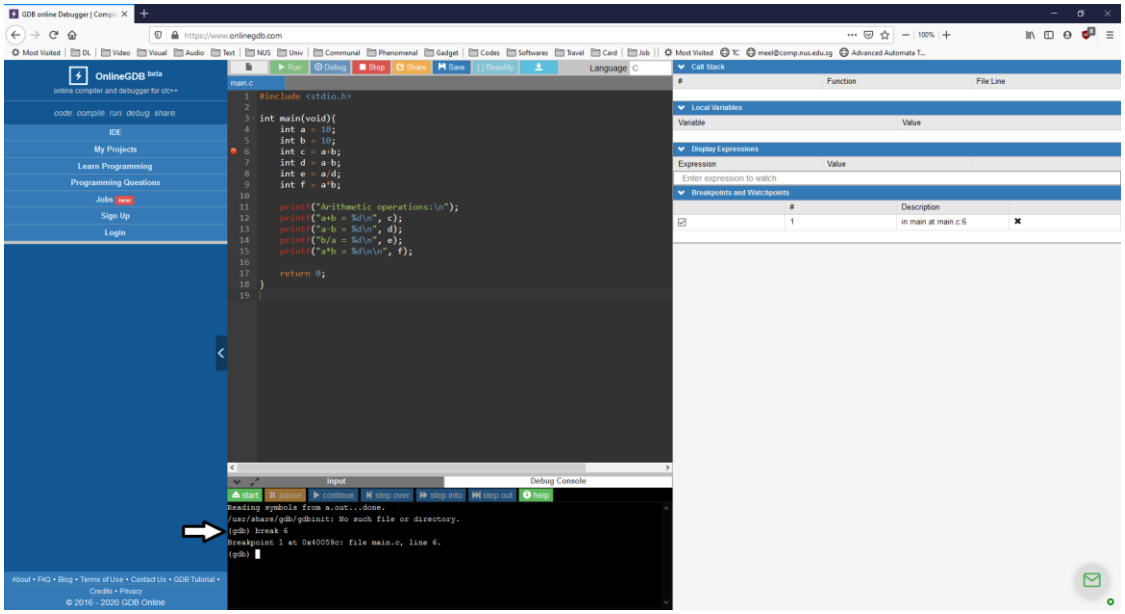


4. Click the “Debug” button to start debugging.



- 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.

5. 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 using the **start** command, **it will pause at line 6.** You can continue execution (till end or the next breakpoint) using the **continue** command.

6. 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

7. 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: _____

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

   > info registers

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

10. Debug and modify **lab1a.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
```

The code **lab1.c** is also shown on the next page for your reference.
Show your labTA the output of your corrected program.

11. Run **lab1b.c** using the online GDB.

12. Can you give 2 ways of displaying the stored value and address value of the first element of an array?

13. Can you define the function **`reverseArray(int arr[], size_t size)`** in the **lab1b.c** using pointers to traverse the array? Write your function below and show your labTA the output.

14. Why do we pass the size of the array to the **`reverseArray`** function in **lab1b.c**? Can we calculate the size of the array inside the function?

15. 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.

Program lab1a.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;

  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);

  return 0;
}
```

Program lab1b.c

```c
#include <stdio.h>
#define MAX 10

int readArray(int [], int);
void printArray(int [], int);
void reverseArray(int [], int);

int main(void) {
  int array[MAX], numElements;

  numElements = readArray(array, MAX);
  reverseArray(array, numElements);
  printArray(array, numElements);

  return 0;
}
```

```c
int readArray(int arr[], int limit) {
  int i, input;

  printf("Enter up to %d integers, terminating with a negative integer.\n",
limit);
  i = 0;
  scanf("%d", &input);
  while (input >= 0) {
    arr[i] = input;
    i++;
    scanf("%d", &input);
  }
  return i;
}

void reverseArray(int arr[], size_t size) {
  // complete the function body
}


void printArray(int arr[], int size) {
  int i;

  for (i=0; i<size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}
```