GDB Debugger

What is debug?

1. Find problems (bugs) in a program
2. Remove the bugs (debug) from the program

Objectives

Learn how to debug a program by

1. Adding print statements
2. Using a debugger

Printing Statements

Insert cout statements between lines of code

- show values of variables for verification
- see how far the program runs

Shortcoming of Printing Statements

- Too many printing statements make output hard to read (overload output)
- May forget to remove the statements when the program is done
- Not able to follow through the program efficiently

Why a Debugger?

- Help developers investigate a program
- Study / trace a program **during** execution
- Useful for catching runtime errors

Key Features in Debugger

- Breakpoint
    - a break in the program execution
        - execution pauses at the statement with a breakpoint
    - there may be more than one breakpoint in the program
    - After pausing the program, you can follow the execution line by line (step by step)
- Watching variables
    - track variable values during program execution
        - see what is the value of a variable at a specific point in the execution
    - watch more than one variable at the same time

GNU Debugger

In this course, we will use the debugger called **GNU debugger** (GDB)

Before Debugging

Programs need to be compiled with debugging information

- Invoke g++ with option -g

Example:
Let' say we have the *hello.cpp* and compile it with -g flag

```
$ cat hello.cpp
#include<iostream>
struct Node {
      int id;
      Node *next;
};

int main() {
      Node *current = NULL;

      for (int i = 0; i < 5; i++) {
            Node *_newNode = new Node;
            _newNode->id = i;
            _newNode->next = NULL;

            current->next = _newNode;
      }

      return 0;
}
```

```
$ g++ -pedantic-errors -std=c++11 -g hello.cpp -o hello
```

Basic GDB commands

- r - Start program execution
- q - Quit GDB
- p - Print the current value of a variable.

Example:

Console	**Action**
`$ gdb hello`	Launch GDB with the hello

`(gdb) r`
`Starting program: /home/research/ra/1801/cklai/t/hello`
 	
`Program received signal SIGSEGV, Segmentation fault.`
`0x00005555555547d2 in main () at hello.cpp:15`
`15 current->next = _newNode;`
	Type "r" or "run" to run the program.
	
	The running program throws a runtime error at line 15 and pauses at that line.

`(gdb) p i`
`$1 = 0`
`(gdb) p current`
`$2 = (Node *) 0x0`
	Use "p" or "print" command to view the current values of i and current.
	
	We know that current = 0, which is a NULL pointer. So, current->next is invalid. We found the bug.

`(gdb) q`	Quit GDB

Useful links:

This website introduce more GDB commands
https://www.tutorialspoint.com/gnu_debugger/gdb_debugging_programs.htm

Example: Another use-case scenario of using GDB to debug a program
https://www.tutorialspoint.com/gnu_debugger/gdb_debugging_example1.htm

Example: Setting breakpoints to help debugging a program
https://www.geeksforgeeks.org/gdb-step-by-step-introduction/

Full set of commands:
http://www.yolinux.com/TUTORIALS/GDB-Commands.html

Online GDB
This website give you a online GDB platform to debug a program. (requires login) Other than command line environment, it also include a interactive panel to show current values of variables and control breakpoints.
https://www.onlinegdb.com/