Module 10. 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 the program efficiently

Why a Debugger?

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

Key Features of 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 the value of a variable at a specific point in the execution is
    - 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:
Compile the *hello.cpp* 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 1:

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

`(gdb) r`
`Starting program: /home/engg1340/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`.
	
	From the information, we know that when **`i = 0`** in the for loop, **`current = 0x0`**, which is a NULL pointer. Therefore, `current -> next` is an invalid operation. The bug is found.

`(gdb) q`	Type “q” to quit GDB

Other useful GDB commands

- b – Set breakpoint
- clear – Clear breakpoint
- n – Execute next line of code
- l (small case of L) – Display the code

Example 2:

Console	**Action**
`$ gdb hello`	Launch GDB with *hello*
`(gdb) b 8` `Breakpoint 1 at 0x792: file hello.cpp, line 8.`	Type “b 8” or “break 8” to set a breakpoint at line 8

`(gdb) r`
`Starting program: /home/engg1340/hello`
 	
`Breakpoint 1, main () at hello.cpp:8`
`8 Node *current = NULL;`
	Type “r” or “run” to run the program.
	
	The running program stops at the breakpoint (line 8).

`(gdb) p i`
`No symbol "i" in current context.`
`(gdb) p current`
`$1 = (Node *) 0x7fffffffe950`
	Use “p” or “print” command to view the current values of i and current.
	
	The program haven’t seen `i` at this point. The program just saw `current` but the statement in line 8 hasn’t been executed yet and so `current` contains some garbage values.

`(gdb) n`
`10 for (int i = 0; i < 5; i++) {`
`(gdb) p current`
`$2 = (Node *) 0x0`
	Use “n” or “next” command to execute the next line of code.
	
	`current = 0x0`, which is a NULL pointer.

`(gdb) n`
`11 Node *_newNode = new Node;`
`(gdb) n`
`12 _newNode->id = i;`
`(gdb) n`
`13 _newNode->next = NULL;`
`(gdb) n`
`15 current->next = _newNode;`
`(gdb) n`
 	
`Program received signal SIGSEGV, Segmentation fault.`
`0x00005555555547d2 in main () at hello.cpp:15`
`15 current->next = _newNode;`
	Use “n” or “next” command again and again to execute the next few lines of code.
	
	The running program throws a runtime error at line 15 and pauses at that line.
	
	You can repeat the steps in Example 1 to look for the bug.

`(gdb) q`	Type “q” to quit GDB

Useful links:

This website introduces 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 gives you an online GDB platform to debug a program. Apart from the command line environment, it also includes an interactive panel to show current values of variables and control breakpoints.
https://www.onlinegdb.com/