Programming Assignment 0: Development Environment Setup/C++/GDB

Due on 8/31/19 at 11:59pm

Introduction

The objective of this programming assignment is to help you ready for other programming assignments by covering introductory subjects. Through this assignment, you will complete setting up your own development environment where you will work until the end of the semester, recap the basics of C++ programming and gain practical experience using gdb to debug a C++ program. You are given a simple program, *buggy.cpp* about a linked list data structure that has several bugs you are to find out. Once you have corrected the program, submit a report documenting your findings and the process that you followed. More about the report coming up in the following.

Exercises

1. **[System Setup]** Set up the development environment by choosing one of these options: AWS, Virtual Machine with linux system, or your own linux machine. You can find the detailed guideline under the resources of our course website. Once you complete the setup, copy a screenshot on your environment and write the system you choose.

2. **[C++ Compilation]** Compile your buggy C++ program to have an executable program called "buggy", (not a.out). What command do you have to use?

3. **[Compile Time Error Debugging]** While compiling the buggy program, you may see several syntax errors. Fix the following bugs in a given source file, buggy.cpp, referring to Table 1 including the program.

    Bug 1: Write a statement in ***Blank A*** to include a header file of a container class, *vector*?

    Bug 2: Write a statement in ***Blank B*** to allow all symbols in the namespace *std* to be visible without adding the namespace prefix?

    Bug 3: Write an assess specifier in ***Blank C*** to allow variables to be used by outside of a class?

    Bug 4: Correct the statements in **Line 19/20/25/33/34** where a member variable of an object is access through a pointer variable.

4. **[C++ Program Execution]** Now you should be able to successfully compile the buggy program. Try to run the program. Copy a screen shot including how to run a program and what errors you encounter.

5. **[C++ Compilation in Debug Mode]** From now on, you begin to fix the runtime errors. First, compile your program **in *debug mode*** to use gdb. Write a command for the compilation.

6. **[GDB Start/Run/Backtrace]** Once compiled, (1) start gdb with the executable program name (i.e. buggy) and (2) run the program inside gdb. The gdb may stop after dumping a segmentation fault error. (3) Print a backtrace to see how your program reached the last statement with the error. Copy a screenshot including three exercises

7. **[GDB Breakpoint/Print]** The backtrace summary shows that the program stops at the statement in Line 19 (unless you change program to have different line numbers). Let's dig into what causes the segmentation fault error. (1) Set a breakpoint in Line 19, (2) run your program inside gdb. When you see a question, "Start it from the beginning? (y or n)", choose y. Then, your program starts again from the beginning and stop at the breakpoint. (3) Then, print what is stored in mylist[i] by using a gdb command (neither cout nor printf). Copy a screenshot including three exercises.

8. **[C++ Runtime Error Fix (Null-Pointer)]** Write a value stored in mylist[i] and write a statement in Blank D to fix the segmentation error (Use the value in mylist[i] as a hint).

9. **[C++ Runtime Error Fix (non-NULL garbage value Pointer)]** Compile the program again and run it inside gdb. You still face another segmentation fault in function *add_LL*. By using commands you practiced in Exercise 7, fix the segmentation fault error. Explain how to fix it (Hint: The second part of function *create_LL* should be corrected)

10. **[Dynamically Allocated Memory Deletion]** The function *create_LL* dynamically allocates memory spaces for elements (i.e. nodes) of mylist. Write the code that deletes the nodes in Blank E to avoid memory leak.

Report

Submit a report with the corrected code on ecampus. The report should document the steps you followed to dbug each problem briefly, but accurately. Make sure to write your logical thought process along the way. Start with identifying the line causing a crash/problem, and then describe what you suspected first, what changes you made to confirm your suspicion, were you right about the suspicion. If you suspected a wrong piece of code, that is totally fine. You should the start over and suspect another part of your code, which is how every programmer solves a problem. The suspect becomes more and more educated and accurate as you practice debugging more.

```
01: #include <iostream>
02: //Blank A
03:
04: //Blank B
05:
06: class node {
07: //Blank C
08:     int val;
09:     node* next;
10: };
11:
12: void create_LL(vector<node*>& mylist, int node_num)
13: {
14:     mylist.assign(node_num, NULL);
15:
16:     //create a set of nodes
17:     for (int i = 0; i < node_num; i++) {
18:         //Blank D
19:         mylist[i].val = i;
20:         mylist[i].next = NULL;
21:     }
22:
23:     //create a linked list
24:     for (int i = 0; i < node_num; i++) {
25:         mylist[i].next = mylist[i+1];
26:     }
27: }
28:
29: int add_LL(node* ptr)
30: {
31:     int ret = 0;
```

```
32:       while(ptr) {
33:                 ret += ptr.val;
34:                 ptr = ptr.next;
35:       }
36:       return ret;
37: }
38:
39: int main(int argc, char ** argv)
40: {
41:       const int NODE_NUM = 3;
42:       vector<node*> mylist;
43:
44:       create_LL(mylist, NODE_NUM);
45:       int ret = add_LL(mylist[0]);
46:       cout << "The sum of nodes in LL is " << ret << endl;
47:
48:       //Step4: delete nodes
49:       //Blank E
50:       //
51: }
```

Table 1. Sample Buggy Code (buggy.cpp)