Patel Harshank
Dr. Ahmed
UIN: 527009145

Assignment Report

CSCE-313-512

PA0 Environment Setup, AddressSanitizer and GDB

Objective: The objective of this programming assignment is to help me get ready for other programming assignments by covering some introductory topics.

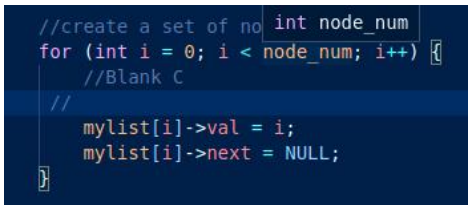
Procedure: The assignment was divided into five parts as described below:

1. **System Setup:** The system setup required download and installation of various tools to setup personal dev environment. The tools included g++, gdb, AddressSanitizer, Favorite IDE (VSCode), Ubuntu and Oracle Virtual Machine.

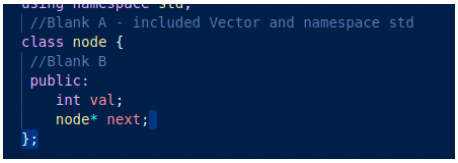
2. **Compilation Errors:** Upon downloading the buggy.cpp program and compiling it with regular g++ gave some compilation errors. Before the program debugger can be used to fix any errors, I came across the first error of not having the appropriate libraries installed for the program to run. I included the following libraries:

   ```
   #include <vector>
   using namespace std
   ```

   After adding these libraries, I still encountered several errors with the member variable being accessed via the dot '.' notation where it should be accessed via the arrow '->' notation.

   Fixing these errors still had a compilation issues where the access specifier for the attributes in the class node was not defined. Adding the keyword `public` did the trick and upon compiling the program, we encountered an segmentation fault.

3. **Segmentation Fault Errors - 1:** To get more details about how and where the program was giving segmentation fault, I began to compile my program using the

Symbol Table argument before enabling GDB debugger. The *Symbol Table* contains debugging information that tells a debugger what memory locations correspond to which symbols (like function names and variable names) in the original source code file. The following syntax was used to compile the program using the symbol table:

```
g++ -o buggy buggy.cpp -g
gdb ./buggy
```

4. **Segmentation Fault Errors - 2:** After enabling GDB, I typed in the command **run** to check where the program stops executing. I found out that the program was not able to go further the Blank – C. The error displayed on the screen from the GDB was:

```
Program received signal SIGSEGV, Segmentation fault.
0x00005555555552f7 in create_LL (mylist=std::vector of length 3, capacity 3 = {...}, node_num=3) at buggy.cpp:19
19              mylist[i]->val = i;
(gdb)
```

Which meant that there has to be some issue with the memory allocation since the length of the vector is 3, capacity is 3 which is what we want but the assignment of the variable 'i' failed. Therefore, I looked for the memory allocation of the nodes inside the vector and I didn't find one. I noticed by going up the stack by commands I learned from the GDB Cheat Sheet like ***backtrace*** and ***step***, the program doesn't allocate the memory for nodes on the HEAP. Therefore, the next step was to add the break point and check if the program stops at this line. Therefore the Blank – C with allocating the null vector with empty blank nodes and it seemed to have fixed the error.

```
//create a set of nodes
for (int i = 0; i < node_num; i++) {

    //Blank C - Added the new node on the ith location
    mylist[i] = new node;


    mylist[i]->val = i;
    mylist[i]->next = NULL;
}
```

5. **Segmentation Fault Errors - 3:** The program then resulted an error in the *"create a linked list"* function where the function was trying to access the next value which didn't exist. Therefore, the only logical conclusion was that the errors exist in the for-loop deceleration where the for loop goes out of bounds while creating the link list of size N.

```
(gdb) run
Starting program: /home/osboxes/Desktop/ProgramCode/PA0/buggy

Program received signal SIGSEGV, Segmentation fault.
0x00005555555553cc in sum_LL (ptr=0x21) at buggy.cpp:33
33              ret += ptr->val;
(gdb)
```

To overcome this issue, I added the termination condition a -1 so the loop can only go from 0 to n-1 elements connecting the current element to the next/following element and therefore creating a linked list. As shown in the code below, the following errors were fixed by the use of next and step as well as looking at the error stack provided by the gdb.

```
//create a linked list
for (int i = 0; i < node_num-1; i++) {
    mylist[i]->next = mylist[i+1];
}
```

6. **AddressSanitizer Memory Error:** The addressSanitizer was used to detect for memory leaks and other errors related with memory. I compiled the address sanitizer using the following command:

```
g++ -o buggy buggy.cpp -g -fsanitizer=address
./buggy
```

After compiling the program, the program gave out the correct output but also threw memory leak error(s) as shown in the photo below:

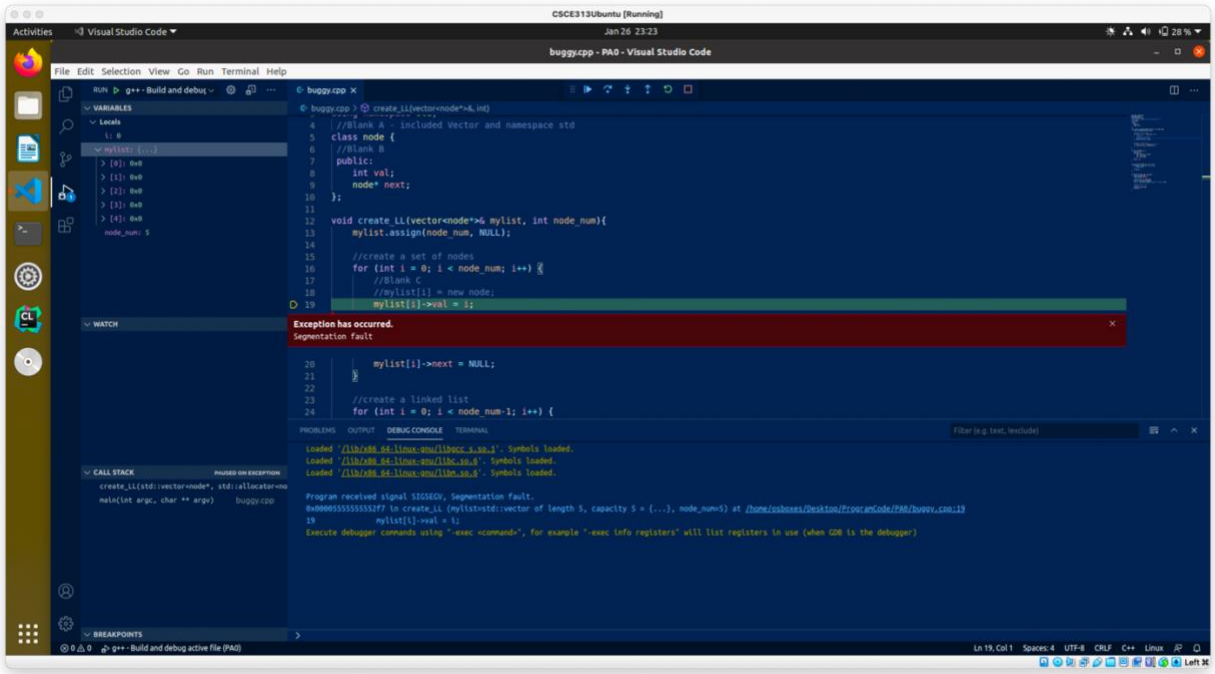
```
The sum of nodes in LL is 3

=================================================================
==3077==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 16 byte(s) in 1 object(s) allocated from:
    #0 0x7f1720484f17 in opera                    /lib/x86_64-linux-gnu/libasan.so.6+0xb1f17)
    #1 0x55e732eb6559 in creat  Open file in editor (ctrl + click)  std::allocator<node*> >&, int) /home/osboxes
    #2 0x55e732eb6989 in main /home/osboxes/Desktop/ProgramCode/PA0/buggy.cpp:52
    #3 0x7f1720015cb1 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x28cb1)

Indirect leak of 32 byte(s) in 2 object(s) allocated from:
    #0 0x7f1720484f17 in operator new(unsigned long) (/lib/x86_64-linux-gnu/libasan.so.6+0xb1f17)
    #1 0x55e732eb6559 in create_LL(std::vector<node*, std::allocator<node*> >&, int) /home/osboxes
    #2 0x55e732eb6989 in main /home/osboxes/Desktop/ProgramCode/PA0/buggy.cpp:52
    #3 0x7f1720015cb1 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x28cb1)

SUMMARY: AddressSanitizer: 48 byte(s) leaked in 3 allocation(s).
pplications boxes:~/Desktop/ProgramCode/PA0$
```

To fix these memory errors, there needed to be a destructor to delete the pointers and therefore I created the function del into the program.

```
void del(vector<node *>&a, int size){
    for(int i = size-1; i >= 0; i--){
        delete a[i];
    }
    a.clear();
}
```

7. **GUI Debugger –** The GUI debugger that was installed with the VSCode IDE was used to debug the entire program from scratch. Here is a picture of my debugging into VSCode for the same assignment.

Patel Harshank
Dr. Ahmed
UIN: 527009145



Key Findings Summary: The following errors were found in the assignment:

1. Required Libraries Not Installed – The program was missing a few libraries that were not imported like the Vector and namespace std libraries.
2. Missing Access Specifiers – The program didn't completely declare the access specifiers for Node class which was then declared to public so the attributes of node class can be used outside the class by other components of the program.
3. Missing Instantiation of Blank Nodes inside the vector – The program allocated 3 null spaces into the vector and forgot to allocate empty nodes in each of them, so I had to add a line mylist[i] = new node; in order to allocate a node inside the vector mylist.
4. Out of Bounds Error – The program was trying to make a linked list with the first n elements by connecting them together while the algorithm should traverse first n-1 elements, it went to first n elements, and therefore created a segmentation fault.
5. Memory Dump Errors – Finally, the program didn't free up the space on *HEAP* after using it, so I had to declare a destructor in-order to delete the used-up memory by the program.

Conclusion: The GUI debugger was one of the most helpful elements of the assignment but also one of the worst one to get started with. If I hadn't installed VSCode and installed other IDE like ATOM, I would have not been able to use ***GUI Debugger*** it for debugging this assignment. I like to use the Terminal for debugging since it's easy to use.

Patel Harshank
Dr. Ahmed
UIN: 527009145

Source Code:

```
1.  // Harshank Patel
2.  // 527009145
3.  // CSCE-313-512
4.  // PA 0: Environment Setup, AddressSanitizer and GDB
5.
6.  #include <iostream>
7.
8.  //Blank A - Added Vector & standard template library
9.  #include<vector>
10. using namespace std;
11.
12.
13. class node {
14.  //Blank B  - Made public as the access specifiers
15.  public:
16.     int val;
17.     node* next;
18. };
19.
20.
21. void create_LL(vector<node*>& mylist, int node_num){
22.     mylist.assign(node_num, NULL);
23.
24.     //create a set of nodes
25.     for (int i = 0; i < node_num; i++) {
26.
27.         //Blank C - Added the new node on the ith location
28.         mylist[i] = new node;
29.
30.
31.         mylist[i]->val = i;
32.         mylist[i]->next = NULL;
33.     }
34.
35.
36.     //create a linked list
37.
38.
39.     //Black Z - Added the -1 in the index so it doesnt go out of bounds!
40.     for (int i = 0; i < node_num-1; i++) {
41.         mylist[i]->next = mylist[i+1];
42.     }
43. }
44.
45. // *** NO CHHANGE *** //
46. int sum_LL(node* ptr) {
47.     int ret = 0;
48.     while(ptr) {
49.         cout <<ptr->val <<endl;
50.         ret += ptr->val;
51.         ptr = ptr->next;
52.     }
53.     return ret;
54. }
55.
56.
57. //Answer to Blank D - Delete Function
58. void del(vector<node *>&a, int size){
```

```
59.     for(int i = size-1; i >= 0; i--){
60.         delete a[i];
61.     }
62.     a.clear();
63. }
64.
65.
66. int main(int argc, char ** argv){
67.     const int NODE_NUM = 3;
68.     vector<node*> mylist;
69.
70.     create_LL(mylist, NODE_NUM);
71.     int ret = sum_LL(mylist[0]);
72.     cout << "The sum of nodes in LL is " << ret << endl;
73.
74.     //Step4: delete nodes
75.     //Blank D - Added a delete function to clear all the memory leaks!
76.     del(mylist,NODE_NUM);
77.
78.
79.     return 0;
80. }
```

Sources:

Resources and Installation Guide - https://sites.google.com/tamu.edu/csce-313-tanzir/others
GDB Cheat Sheet - https://cheatography.com/oddcoder/cheat-sheets/gdb/
GUI Debugger Tutorial - https://docs.microsoft.com/en-us/cpp/linux/linux-asan-configuration?view=msvc-160