Project 1

Sharan Sivakumar
CS 408 - Software Testing
Lin Tan
Purdue University

Question 1

- Refer to files in the q1 folder
- What I found was there was a slight discrepancy between the results. Some of the languages returned -1 or 1. For example, in Python, the result is 1, while in Java, the result is -1. The reason for this discrepancy is that in most languages, the value of the modular operation takes whatever sign the dividend is (the top number). In Python, however, the result of modular operation takes the sign of the divisor(the bottom number)
    - - 5 % 2
    - -5 is the dividend
    - 2 is the divisor

- Strategies to cope with this issue:
    - One strategy that we can use would be to modify/design compilers that are able to adhere to a certain standard when it comes to whether or not the result of modular division takes the sign of the dividend or the divisor.
    - Another strategy that we can take would be to change how we educate our developers on the discrepancies between modular divisions in different languages. If we can get them to change how they deal with this issue, we can create a new standard when dealing with this issue. We can also modify the IDEs that developers use so that they notify the developers when they run into the discrepancy.

Question 2

2.a

Error documentation:
To find out where the memory leaks were I used this valgrind command: valgrind --leak-check=full --track-origins=yes ./sll_buggy

```
==162102==
[(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:i
enter the tel:>100
enter the name:>Tom
[(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:i
enter the tel:>111
enter the name:>Mary
[(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:d
enter the tel :>111
[(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:x
bye
==162102==
==162102== HEAP SUMMARY:
==162102==     in use at exit: 9 bytes in 1 blocks
==162102==   total heap usage: 7 allocs, 6 frees, 2,115 bytes allocated
==162102==
==162102== 9 bytes in 1 blocks are definitely lost in loss record 1 of 1
==162102==    at 0x484DCD3: realloc (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==162102==    by 0x1093A9: fgets_enhanced (sll_buggy.c:45)
==162102==    by 0x109A60: main (sll_buggy.c:277)
==162102==
==162102== LEAK SUMMARY:
==162102==    definitely lost: 9 bytes in 1 blocks
==162102==    indirectly lost: 0 bytes in 0 blocks
==162102==      possibly lost: 0 bytes in 0 blocks
==162102==    still reachable: 0 bytes in 0 blocks
==162102==         suppressed: 0 bytes in 0 blocks
==162102==
==162102== For lists of detected and suppressed errors, rerun with: -s
==162102== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Explanation of Issue: The issue is shown in the Valgrind output, which shows that we have a memory leak of 9 bytes somewhere in the program. Valgrind tells us that the memory leak is coming from the ‘realloc’ function call inside the ‘fgets_enhanced’ function. Upon closer inspection, we can see that there is a memory leak because we didn’t free the memory allocated by ‘realloc’ properly. To fix this, we need to make sure that we free the memory whenever we have a ‘malloc’ or ‘realloc’ call. Our specific issue comes in the ‘delete_node’ function where we don’t free the memory allocated for the string (‘temp->str) and to fix that we need to add a ‘free(temp->str)’ before we free ‘temp’

2.b

To find out where the memory issues were, I used the valgrind line: valgrind --leak-check=full --track-origins=yes ./sll_buggy

```
==162121== Memcheck, a memory error detector
==162121== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==162121== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==162121== Command: ./sll_buggy
==162121==
[(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:i
enter the tel:>100
enter the name:>Tom
[(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:i
enter the tel:>111
enter the name:>Mary
[(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:i
enter the tel:>112
enter the name:>John
[(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:e
enter the old tel :>111
enter the new tel :>111
enter the new name:>Mary
[(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:a
[(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:x
==162121== Invalid read of size 8
==162121==    at 0x109522: delete_all (sll_buggy.c:102)
==162121==    by 0x109BD0: main (sll_buggy.c:323)
==162121==  Address 0x4ab5920 is 16 bytes inside a block of size 24 free'd
==162121==    at 0x484B27F: free (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
```

```
==162121==    by 0x109544: delete_all (sll_buggy.c:105)
==162121==    by 0x109B87: main (sll_buggy.c:310)
==162121==  Block was alloc'd at
==162121==    at 0x4848899: malloc (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==162121==    by 0x10982E: append (sll_buggy.c:205)
==162121==    by 0x109A72: main (sll_buggy.c:277)
==162121==
==162121== Invalid read of size 8
==162121==    at 0x10952E: delete_all (sll_buggy.c:104)
==162121==    by 0x109BD0: main (sll_buggy.c:323)
==162121==  Address 0x4ab5910 is 0 bytes inside a block of size 24 free'd
==162121==    at 0x484B27F: free (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==162121==    by 0x109544: delete_all (sll_buggy.c:105)
==162121==    by 0x109B87: main (sll_buggy.c:310)
==162121==  Block was alloc'd at
==162121==    at 0x4848899: malloc (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==162121==    by 0x10982E: append (sll_buggy.c:205)
==162121==    by 0x109A72: main (sll_buggy.c:277)
==162121==
==162121== Invalid free() / delete / delete[] / realloc()
==162121==    at 0x484B27F: free (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==162121==    by 0x109538: delete_all (sll_buggy.c:104)
==162121==    by 0x109BD0: main (sll_buggy.c:323)
==162121==  Address 0x4ab58c0 is 0 bytes inside a block of size 5 free'd
==162121==    at 0x484B27F: free (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==162121==    by 0x109538: delete_all (sll_buggy.c:104)
==162121==    by 0x109B87: main (sll_buggy.c:310)
==162121==  Block was alloc'd at
==162121==    at 0x4848899: malloc (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==162121==    by 0x109315: fgets_enhanced (sll_buggy.c:29)
==162121==    by 0x109A60: main (sll_buggy.c:277)
==162121==
==162121== Invalid free() / delete / delete[] / realloc()
```

```
==162121==    at 0x484B27F: free (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==162121==    by 0x109544: delete_all (sll_buggy.c:105)
==162121==    by 0x109BD0: main (sll_buggy.c:323)
==162121==  Address 0x4ab5910 is 0 bytes inside a block of size 24 free'd
==162121==    at 0x484B27F: free (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==162121==    by 0x109544: delete_all (sll_buggy.c:105)
==162121==    by 0x109B87: main (sll_buggy.c:310)
==162121==  Block was alloc'd at
==162121==    at 0x4848899: malloc (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==162121==    by 0x10982E: append (sll_buggy.c:205)
==162121==    by 0x109A72: main (sll_buggy.c:277)
==162121==
bye
==162121==
==162121== HEAP SUMMARY:
==162121==     in use at exit: 0 bytes in 0 blocks
==162121==   total heap usage: 12 allocs, 18 frees, 2,167 bytes allocated
==162121==
==162121== All heap blocks were freed -- no leaks are possible
==162121==
==162121== For lists of detected and suppressed errors, rerun with: -s
==162121== ERROR SUMMARY: 12 errors from 4 contexts (suppressed: 0 from 0)
```

Explanation of Issue: When we run this test case, Valgrind shows us two errors. There is an invalid read of size 8 at the ‘delete_all’ function and there is a double-free error in the ‘delete_all’ function. In the invalid read error, the function is trying to access memory that has already been freed(trying to access the next attribute of a node that has already been freed). To fix this, we set ‘temp2’ equal to null. To fix the double-free error, we added an ‘if’ statement in the ‘x’ case of the switch statement that checks if ‘p’ is not equal to NULL.

2.c

In ‘fgets_enhanced” there could be a potential bug when we handle the null terminator for the strings. If ‘nPos’ is null, this would cause a segmentation fault.

This is the test case:

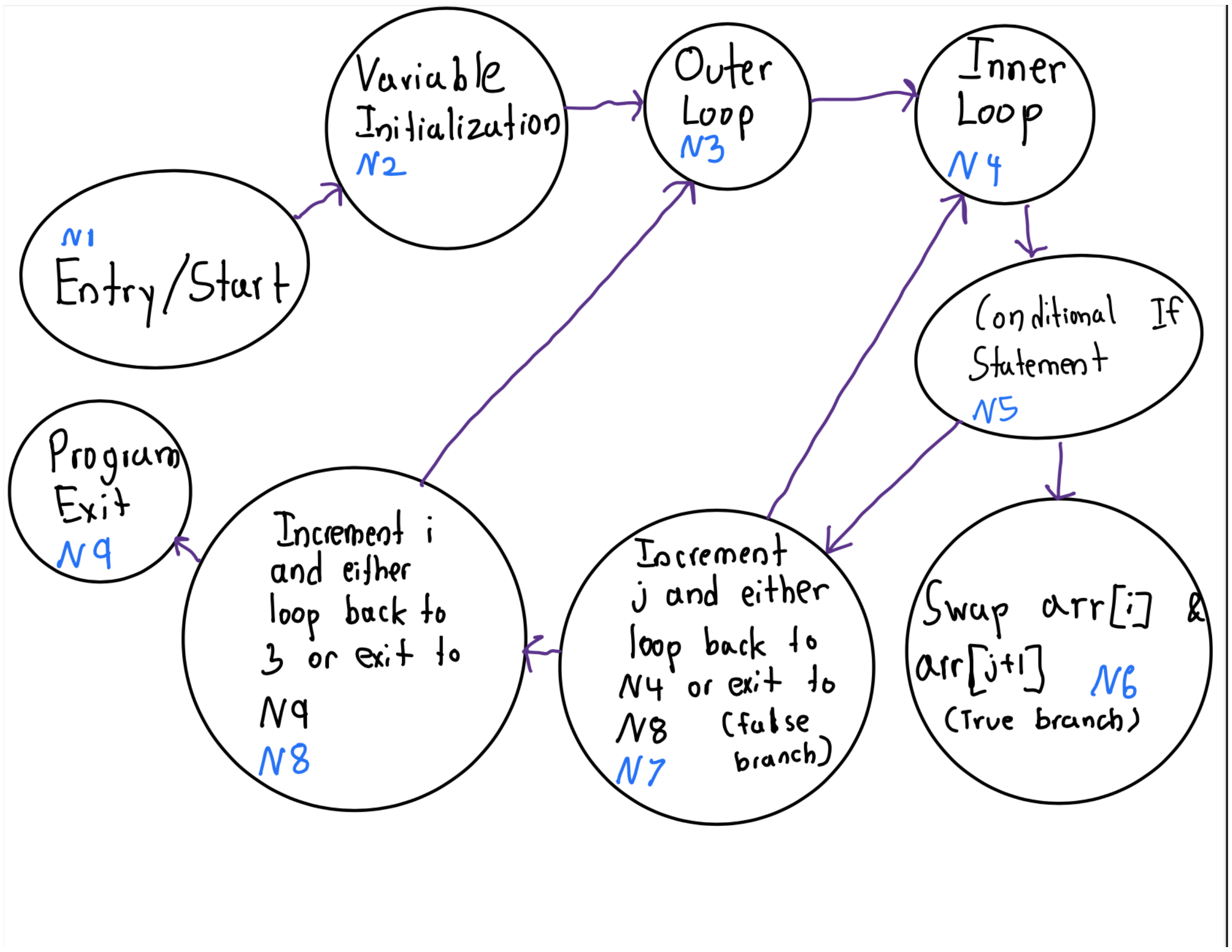
```
(i)nsert,(d)elete,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]: i
enter the tel:> 54321
```

enter the name:> TestWithoutNewLineAtEnd

If there is any problem with the file read, this test case would expose a segmentation fault in the code. I fixed this bug by modifying the ‘fgets_enhanced” function by adding the respective if conditional statement so that it handles when ‘nPos’ is NULL.

Question 3

- In order to draw the control flow graph(CFG) of this function with the minimum number of nodes, we need to identify the basic blocks of the function. The basic blocks are the sequences(blocks) of statements that are always executed together
- Basic Blocks:
    - Entry/Start of Program (Node 1)
    - Variable initialization (Node 2)
    - Outer for loop (Node 3)
    - Inner for loop (Node 4)
    - Conditional if statement (Node 5)
    - Swapping section inside the if statement (Node 6)
    - End of inner loop (Node 7)
    - End of out loop (Node 8)
    - Program exit/Return (Node 9)



-

Question 4

4.a

- Node coverage Test Requirements(TR_{NC}):
- Edge coverage (TR_{EC}):
    - In edge coverage, we must cover all edges at least once
    - TR_{EC}: [N1, N2], [N1, N3], [N2, N3], [N3, N4], [N3, N5], [N3, N6], [N3, N7], [N4, N8], [N5, N8], [N6, N7], [N7, N8], [N8, N9], [N8, N10], [N9, N11], [N10, N11].
- Edge-pair coverage (TR_{EPC}):
    - In edge-pair coverage, we want the set of each reachable path of length up to two.
    - TR_{EPC} = {[1, 2, 3], [1, 3, 4], [1, 3, 5], [1, 3, 6], [1, 3, 7], [2, 3, 4], [2, 3, 5], [2, 3, 6], [2, 3, 7], [3, 4, 8], [3, 5, 8], [3, 6, 7], [3, 7, 8], [4, 8, 9], [4, 8, 10], [5, 8, 9], [5, 8, 10], [6, 7, 8], [7, 8, 9], [7, 8, 10], [8, 9, 11], [8, 10, 11]}

- Prime path coverage (TR_{PPC}):
    - TR_{PPC} = {[1,2,3,6,7,8,9,11], [1,2,3,6,7,8,10,11], [1,2,3,5,8,10,11], [1,2,3,5,8,9,11], [1,2,3,4,8,9,11], [1,2,3,4,8,10,11], [1,2,3,7,8,9,11], [1,3,6,7,8,10,11], [1,2,3,7,8,10,11], [1,3,6,7,8,9,11], [1,3,4,8,10,11], [1,3,4,8,9,11], [1,3,5,8,9,11], [1,3,5,8,10,11], [1,3,7,8,10,11], [1,3,7,8,9,11]}
    - Reachable Prime Paths: {[1,2,3,6,7,8,9,11], [1,2,3,5,8,9,11], [1,2,3,4,8,10,11], [1,2,3,7,8,9,11], [1,3,6,7,8,9,11], [1,3,4,8,10,11]}
    - Unreachable Prime Paths: {[1,2,3,6,7,8,10,11], [1,2,3,5,8,10,11], [1,2,3,4,8,9,11], [1,3,6,7,8,10,11], [1,2,3,7,8,10,11], [1,3,4,8,9,11], [1,3,5,8,10,11], [1,3,7,8,10,11]}

4.b

- 1 is possible
- 2 is possible
- 3 is not possible because we cannot cover all of the prime paths of any given CFG without covering the Edge Paths of said CFG
- 4 is possible

Question 5

- 5.f

  For the ‘addNode’ method in CFGTest.java, it satisfies node coverage because all lines of the ‘addNode’ method in CFG.java are covered. However, because the ‘addNode’ test method in CFGTest.java only tests the cases where the node is not already present in the graph, this method doesn’t satisfy edge coverage.

  The ‘addEdge’ method in CFGTest.java, definitely satisfies node coverage because in order to cover all the edges, we would have to cover all the graphs. However, more test cases would be needed to satisfy edge coverage because currently, this test class doesn’t satisfy edge coverage.

  The ‘deleteNode’ method in CFGTest.java, covers both node coverage and edge coverage due to the fact that Deleting an existing edge and verifying that the edge is removed correctly satisfies of all the necessary edges and nodes.

  For the ‘isReachable’ method in CFGTest.java, because of the fact that the execution of the method that I wrote passes through all of the necessary nodes and edges, it satisfies both node and edge coverage.