Connor Brown

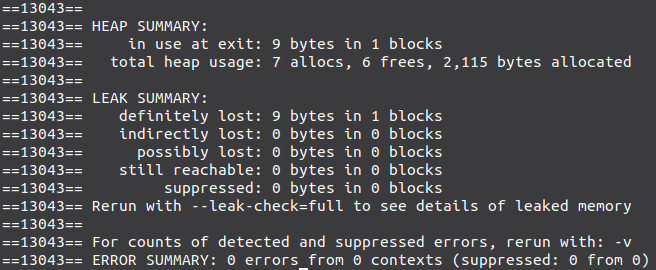
[brow1325@purdue.edu](mailto:brow1325@purdue.edu)

Q1:

I have found that in c++ and java that the modulus operator for -5 modulo 2 produces the result of -1. However, in perl and python we get the result of 1. The reason for this is because when doing the calculations in the first two it returns the modulus with the same sign as the numerator, and in the other two it returns the modulus with the same sign as the denominator. One strategy would be to make a universal standard for the operator so that it always returns it based on the numerator. Another option would be to have it return the absolute value of the modulus in order to always make it the same across different languages.

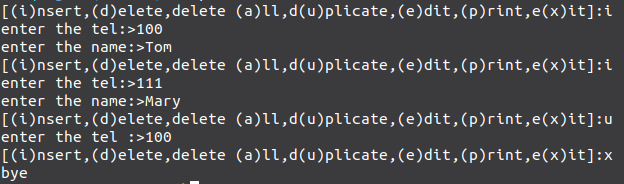
Q2:

1. Test Case 1 Report memory problem
   1. Valgrind output



* 1. The issue is that when the delete function is called that it does not free the memory properly.
  2. The bug was fixed by adding a missing free command in the delete node function for freeing the temp node’s string.

1. Test Case 2 Report memory problem
   1. The problem occurs when the exit command is called after calling the delete all command. This is because delete all gets called twice. Once when calling it directly and again when exiting. The error occurs from the global pointer p continuing to point to the wrong spot after delete all is called. Normally if it is called once then it would not be an issue. But when being called the second time it causes many invalid frees because it is pointing to the wrong section in memory.
   2. To fix the problem all that needs to be done is to set the global variable p to null at the end of the delete all function.
2. Generate Own Test case that causes bug
   1. Test case



* 1. The problem that occurs during this test case is a buffer overflow as a result of calling the duplicate function. This happens because when calling malloc when making space for the name that is being duplicated the length is too small and so when it copies it it ends up overflowing into the next space instead of into the space that it freed up.
  2. To fix this we just need to increase the length for allocation by adding 1.

Q3:

1. Control Flow Graph
2. Test Requirements
   1. TRnc = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 }
   2. TRec = { [1,2], [1,3], [2,3], [3,4], [3,5], [3,6], [3,7], [4,8], [5,8], [6,7], [7,8], [8,9], [8,10], [9,11], [10,11] }
   3. TRepc = { [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] }
   4. TRppc = { [1,2,3,4,8,9,11], [1,2,3,5,8,9,11], [1,2,3,6,7,8,9,11], [1,2,3,7,8,9,11], [1,2,3,4,8,10,11], [1,2,3,5,8,10,11], [1,2,3,6,7,8,10,11], [1,2,3,7,8,10,11], [1,3,4,8,9,11], [1,3,5,8,9,11], [1,3,6,7,8,9,11], [1,3,7,8,9,11], [1,3,4,8,10,11], [1,3,5,8,10,11], [1,3,6,7,8,10,11], [1,3,7,8,10,11] }
   5. Infeasible Test Requirements:
      1. Any test with subpaths [4,8,9], [5,8,10], [7,8,10], [6,7,8,10]
      2. In TRepc, [4,8,9], [5,8,10], [7,8,10]
      3. In TRppc, [1,3,4,8,9,11], [1,3,5,8,10,11], [1,3,7,8,10,11], [1,2,3,4,8,9,11], [1,2,3,5,8,10,11], [1,2,3,7,8,10,11], [1,3,6,7,8,10,11], [1,2,3,7,8,10,11]
      4. These are infeasible because if a path covers node 4 it means that arg.length = 0 and in order to reach node 9, arg.length must be greater than 0. So it will always end up in node 10 instead of node 9.
      5. They are also infeasible if a path covers node 5, 6, or 7 because this means that arg.length is greater than 0. So it will never reach node 10 as it will never have an arg.length less than 0 when it covers nodes 5, 6, or 7.

Q4:

1. Testing
   1. addNode()
      1. For addNode() we satisfy the NC.
      2. When testing addNode() and addNode\_duplicate() we satisfy EC
   2. addEdge()
      1. For addEdge() we satisfy NC.
      2. When testing addEdge() and addEdge\_oneNewNode() we do not satisfy the EC
      3. In order to satisfy the EC we add a new test called addEdge\_duplicate() in order to check that there is no duplicate edges and to then satisfy the EC.
   3. deleteNode()
      1. For deleteNode() we satisfy NC.
      2. When testing deleteNode() and deleteNode\_missing() we satisfy the EC.
   4. deleteEdge()
      1. For deleteEdge() we satisfy NC.
      2. When testing deleteEdge(), deleteEdge\_missing() and deleteEdge\_missingSrcNode() we satisfy the EC
   5. isReachable()
      1. For reachable\_true() and reachable\_missingSrc() we satisfy NC.
      2. When testing reachable\_true(), reachable\_unreachable(), reachable\_missingSrc(), and reachable\_missingTarget() we do not satisfy the EC
      3. In order to satisfy the EC we write a test case called unreachableNode() in order check that nodes with no edges are reachable. Which then satisfies the EC.