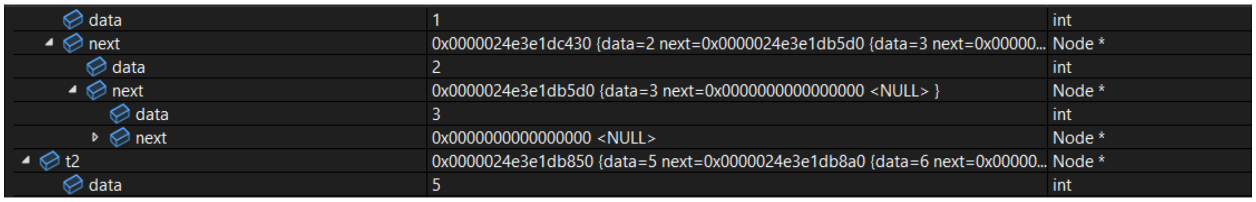
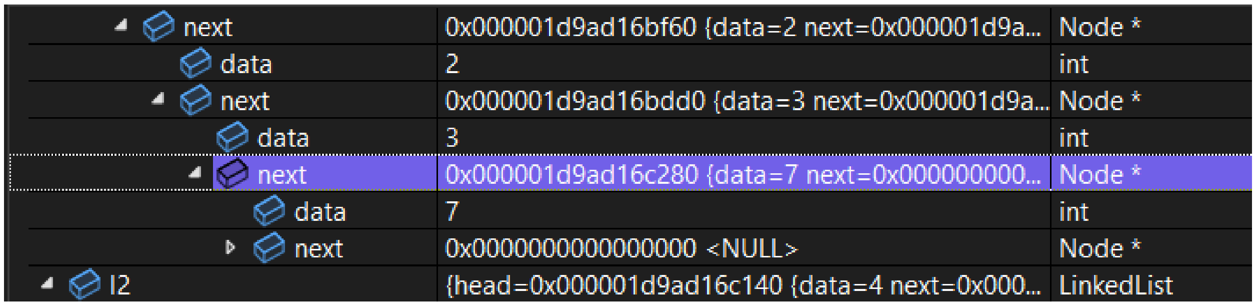
1. On line 41 I change the if (t1 == nullptr) to if (t1 != nullptr)



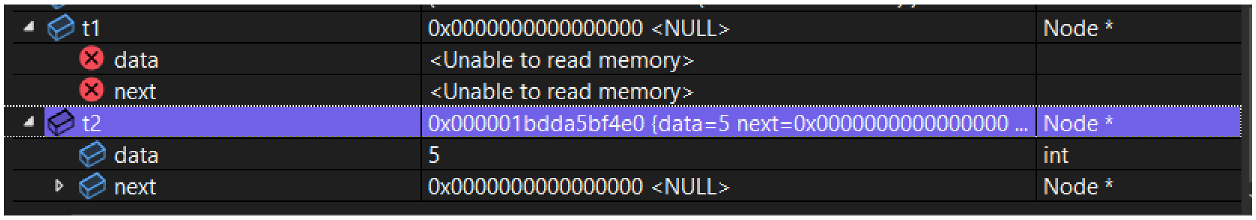
data	1	int
next	0x0000024e3e1dc430 {data=2 next=0x0000024e3e1db5d0 {data=3 next=0x00000...	Node *
data	2	int
next	0x0000024e3e1db5d0 {data=3 next=0x0000000000000000 <NULL> }	Node *
data	3	int
next	0x0000000000000000 <NULL>	Node *
t2	0x0000024e3e1db850 {data=5 next=0x0000024e3e1db8a0 {data=6 next=0x00000...	Node *
data	5	int

2. On line 40, replace the && in the while loop with ||: while (t1 != nullptr || t2 != nullptr)



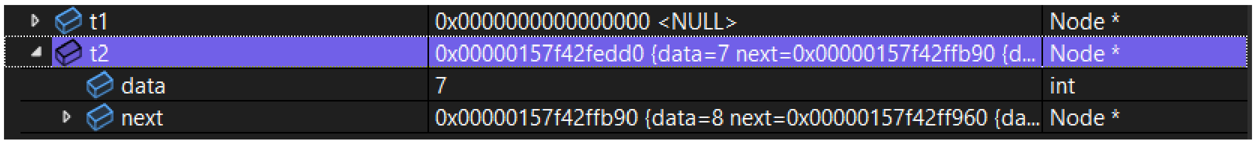
next	0x000001d9ad16bf60 {data=2 next=0x000001d9a...	Node *
data	2	int
next	0x000001d9ad16bdd0 {data=3 next=0x000001d9a...	Node *
data	3	int
next	0x000001d9ad16c280 {data=7 next=0x000000000...	Node *
data	7	int
next	0x0000000000000000 <NULL>	Node *
l2	{head=0x000001d9ad16c140 {data=4 next=0x000...	LinkedList

3. Line 40 change fixes the issue found in test case 3, as the or checks that both lines are equal to each other



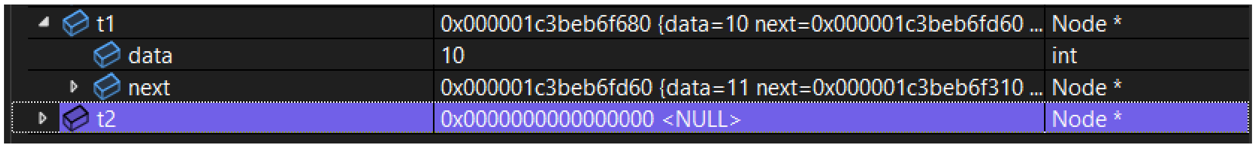
t1	0x0000000000000000 <NULL>	Node *
data	<Unable to read memory>	
next	<Unable to read memory>	
t2	0x000001bdda5bf4e0 {data=5 next=0x0000000000000000 ...	Node *
data	5	int
next	0x0000000000000000 <NULL>	Node *

4. The change to line 40 works the same here, as the change to comparing the size to saying if either of the lists are full runs the program to completion



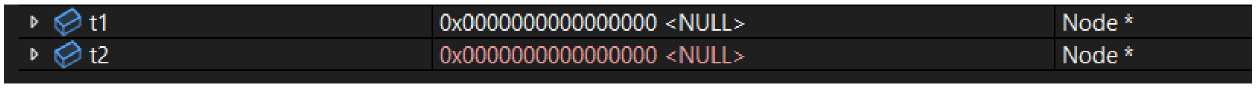
t1	0x0000000000000000 <NULL>	Node *
t2	0x00000157f42fedd0 {data=7 next=0x00000157f42ffb90 {d...	Node *
data	7	int
next	0x00000157f42ffb90 {data=8 next=0x00000157f42ff960 {da...	Node *

5. The change to line 40 fixes the second list being empty because the program will run until the end until the lists are at the end



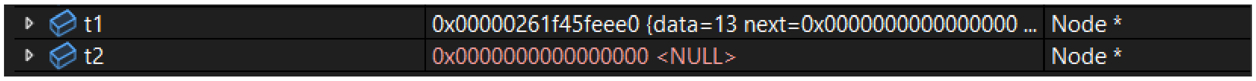
```
◢ t1      0x000001c3beb6f680 {data=10 next=0x000001c3beb6fd60 ...  Node *
    data  10                                                        int
  ▷ next  0x000001c3beb6fd60 {data=11 next=0x000001c3beb6f310 ...  Node *
▷ t2      0x0000000000000000 <NULL>                                 Node *
```

6. This works with both the && and || functions, as both return nothing when the list is empty



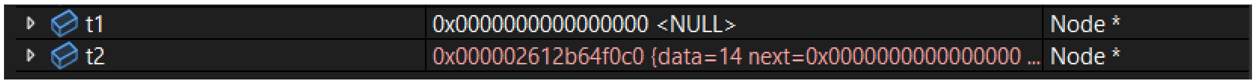
```
▷ t1   0x0000000000000000 <NULL>   Node *
▷ t2   0x0000000000000000 <NULL>   Node *
```

7. The || function fixes the issue with only one element in l1, and only one element will not make a change to the operations of the function



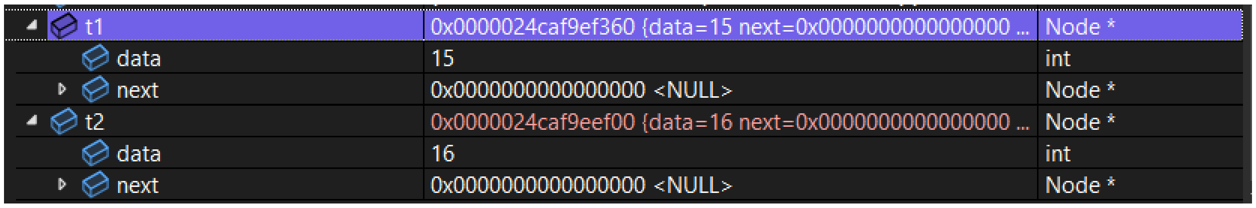
```
▷ t1   0x00000261f45feee0 {data=13 next=0x0000000000000000 ...  Node *
▷ t2   0x0000000000000000 <NULL>                                  Node *
```

8. The || function fixes the issue with only one element in l2, and only one element will not make a change to the operations of the function



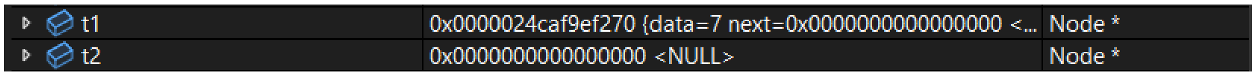
```
▷ t1   0x0000000000000000 <NULL>                                  Node *
▷ t2   0x000002612b64f0c0 {data=14 next=0x0000000000000000 ...  Node *
```

9. The || function does the same thing as the && function, and the && function works here so the program complies in every scenario



```
◢ t1      0x0000024caf9ef360 {data=15 next=0x0000000000000000 ...  Node *
    data  15                                                        int
  ▷ next  0x0000000000000000 <NULL>                                 Node *
◢ t2      0x0000024caf9eef00 {data=16 next=0x0000000000000000 ...  Node *
    data  16                                                        int
  ▷ next  0x0000000000000000 <NULL>                                 Node *
```

10. This is the same scenario as test 2. Using || works here as well



```
▷ t1   0x0000024caf9ef270 {data=7 next=0x0000000000000000 <...  Node *
▷ t2   0x0000000000000000 <NULL>                                 Node *
```

Changed Code Below:

```
LinkedList interweave(LinkedList& l1, LinkedList& l2) {
        LinkedList result;
        Node* t1 = l1.head;
        Node* t2 = l2.head;
        while (t1 != nullptr || t2 != nullptr) {
                if (t1 != nullptr) {
                        result.append(t1->data);
                        t1 = t1->next;
                }
                if (t2 != nullptr) {
                        result.append(t2->data);
                        t2 = t2->next;
                }
        }
        return result;
}
```