CSCI 3901 Winter 2021

Lab 2: Debugging

Name: Dhrumil Rakesh Shah

HfxDonairExpress.java:

Defect & The cause of them:

- Even after selecting Donair, the program asks for toppings which is wrong as it should ask for toppings when Pizza is selected.
    - The defect is caused due to a missing break statement in the switch statement. (changes being made is the java file attached)
- The formula to calculate the discount using coupon was wrong.
    - The original formula was just subtracting the coupon from price.
    - The corrected formula is used to calculate the discount using coupon.
- The String value of the toppings were not being compared properly causing wrong output
    - The "==" were replaced by .equalsIgnoreCase() method to compare the String with the specified Object.
- The price of the toppings was not being added to the initial price of the Pizza leading to wrong calculations.
    - The cause for wrong calculations was due to the initial price being overwritten by the toppings price giving wrong output.
    - It was corrected by adding the toppings price to the initial price. (changes being made is in the java file attached)

The approach to locate the defect:

- Firstly I went through the program to understand what it actually does and how should it work. (This is always my first step)
- Then after understanding what the program should actually do, I move on to see what are the things that are not working as expected by running various test cases.
    - Analysing the test cases gives a sense about what is going wrong in the program
    - Then isolate the logic for which the test cases failed & tried to write the correct logic and re-run the program for the failed test cases.
    - If they pass then rest of the test cases are run to see whether the entire program is running properly.

The amount of time it took:

- It took me 15 minutes of time to locate all the bugs that were causing the program to not function properly as needed and give erroneous output.
- The IntelliJ debugger helped a lot as I was able to minutely analyse each and every step along with all the values being stored and updated.

Ackermann.java:

Defect & The cause of them:

- The program was leading to a StackOverflowError.
    - The cause of the error was wrong logic in the 2nd recursive ackermann() function under the else statement.

(changes have been made in the java file attached)

The approach to locate the defect:

- Firstly, I went through the program to understand what it actually does and how should it work.
- Then I ran the program to see what output does it give
    - In this case the program gave StackOverflowError.
    - As this program implements recursive functions, I looked at the recursive function calls and the logic inside them i.e. the parameters that are being passed.
    - By looking at the recursive functions I was able to pin-point the cause of the error.
    - I even tried to debug by utilizing the debugging feature of IntelliJ, I put a breakpoint at the 1st recursive function under the else if and then under the 2nd recursive function under else statement.
    - By iterating step by step, I was able to find the cause of error.

The amount of time it took:

- It took me 2 minutes to locate the error as it is a program that uses recursive functions, I was able to find the error after running the program once.
- After running the code, it was giving StackOverflowError, so it clearly states that there is some missing base condition due to which the program is not able to come out of the recursive function calls.

LinkedListTest.java:

Defect & The cause of them:

- There is only one defect in the program & that is of Shallow copy instead of Deep copy done of the linked list
    - This causes the append() method to not function properly.
    - If changes made without resolving the problem of Shallow copying then it would result in an Infinite loop in the append() function.

The approach to locate the defect:

- Firstly, I went through the program to understand what it actually does and how should it work.
- As I know the working of Linked list, so it was easy to follow the steps & I ran the program to see what is the output.
- Judging on the basis of output, I backtracked the code form the point of error that is the append() method, leading to the copy() function call.
- I put the breakpoint at the copy() function call in the Main method and started iterating from there.
    - The copy function technically shows no problem in debugging as no error or exception is faced nor any values are wrongly placed, but with careful observation it shows that shallow copy is being made of the linked list.
    - Then moving on from the copy function, I move to the append() function and after iterating, few errors in the logic are identified.
    - Any changes made only to the append() function will cause the program to go into Infinite loop without the addLast() method to create a deep copy of the linked list.

The amount of time it took:

- It took me around half hour to debug this program. Fixed the problem by creating a new addLast() method to make deep copy of the linked list and also make changes to while loop on append() method and the assignment after the while loop.

Analysis:

What strategy or strategies for debugging are most effective for you?

- Go through the code to understand what it actually does. (also a strategy)
- Devise test cases based on the program and run them to see where exactly the program is failing.
- Based on the failed test cases I adopt different strategies to debug the code.
    - **Backtracking**:
        - I start from where the problem seems to be occurring causing the program to not run as expected.
        - Sometimes it could be a single method or a chunk of code causing the problem.
    - **Debuggers:**
        - I use debuggers mainly IntelliJ for setting breakpoints in the program, stepping into and over functions, watching program expressions & inspecting the memory contents.
        - Debuggers help to minutely follow the code & even keep track of the stack trace.
    - **Understanding flow of the Code:**
        - Sometimes just understanding the flow of the code even helps to pin-point the error in the code.
    - **Have I faced this before:**
        - I also try to recall, whether I have faced a similar problem before.
        - If yes, than what approach did I use to resolve it and can the same approach be used for the current one.
    - **Change Perspective:**
        - At times I try and change the approach with which I look at the bug, a fresh perspective helps a lot.
- After making the fix, I review the code by re-running the test cases to see whether the bug has been corrected or not and based on the review I move forward with debugging. It is an iterative process.
- Once the fix is working I look for similar errors within the code.

What makes them effective?

- There are several factors that make my debugging strategies effective to me;
    - It gives me a high-level understanding of the code
    - Test cases helps me to pin-point the problem within the code
    - Debugging tools makes it easy to track the variable values & the flow of the code
    - Saves good amount of time while debugging & resolving the problem.
    - Experience plays a huge role in bug/error solving.

For which conditions of the code will your strategies be effective or ineffective?

- Going through the code at times is a very ineffective due to below factors;
    - If the code is written by someone else and lacks readability
    - Majority of the times the code is very complex to go through it completely
    - This strategy is effective if I already have an understanding of the code & it is not that complex.
- Using Test cases to pin-point the bug causing part of the code;
    - This strategy is very effective and used majority of the times to locate the bug causing code.
    - Only thing to look for is the variety of test cases required to cover all possible scenarios
- Backtracking;
    - This is an effective technique, but will fail if the error is very far away from the symptom.
    - Otherwise it is an effective approach to locate & resolve the bug.

- Debuggers;
    - Using debuggers is always the best way to debug any code as it has several useful features like setting breakpoints in the program, stepping into and over functions.

How can a debugger support your strategies?

- Debugger helps me to analyse the code minutely by placing several breakpoints.
- It makes it easy to run test cases and see which part of the code is causing the error.
- Features like stepping into and over functions helps a lot in backtracking the code.
- Using debugger, it is easy to understand the flow of the code along with stack trace and variable values.
- Mainly debugger helps me with 4 main steps of debugging which are;
    - Identify
    - Isolate
    - Fix
    - Review
    - Look for similar bugs