CSCI 3901 Lab 2: Debugging

Name : Jay Sanjaybhai Patel
Dalhousie ID : B00982253

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

- ➔ For the calendar app, I didn't use any debugging because at the very first time I ran the code, I found two errors just by looking at the code, as it is pretty easy. For linked list, I make a strategy that first let me make the flow in the mind of how all functions work, and then I try different input into the linked list. When I add null to the linked list, I find one error.

What makes them effective?

- ➔ I always try to first read the entire code and then go for debug tools. By doing this, we already have some idea about that function, so we have to just keep an eye on all the variable values during step into and step over.

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

- ➔ When we have a long codebase, it is very difficult to see and remember all the functions working in our mind. Instead, use debugging mode and follow the path of the code, and we automatically get the idea of that entire codebase. For example this strategy is useful in calender but not that much in linkedlist program.

How can a debugger support your strategies?

- ➔ Before this lab_2, I never used this debugging mode, breakpoints, and all. In fact, I always used print statements and it took a lot of time, and many times I just forgot to remove those print statements. In linkedlist I use this concepts and it was very easy for me to navigate through program, as well as finding those bugs.

How, if at all, do you feel that AI-generated content can contribute to debugging?

- ➔ AI tools only help in those errors that can be easily related to that function and we forgot to add that, or we did some mistakes in those part. For example, if we have to move in a 2D array, we make some mistakes in incrementing or decrementing i and j variables; it can easily be found by AI tools. But some runtime errors like index out of bound exception and many others can't be predicted by those tools because it is logic-based.

Analysis

The cause of the defects that you found.

- Calender:
  - ➔ First Bug - The date in every month starts from 0, but it must start from 1. It happened because the for loop to print every month's dates starts from 0.

    Before : for (i = 0; i <= days_to_print; i++)
    After : for (i = 1; i <= days_to_print; i++)

  - ➔ Second Bug - We have an array in which we store the number of days for the months. While we start our program for the first month, we get the first indexed value from that array, but the array is a zero-indexed data structure, so we have to fetch data after minus 1 for that month.

    Before : days_to_print = month_days[month];
    After : days_to_print = month_days[month - 1];

- LinkedList:
  - ➔ First Bug - When we create the first node in our linked list, we have one constructor in which we have to pass a word, but here it uses the two-parameter constructor in which we have to pass a word and nextNode. That function is used when we already have at least one element in our linked list as we have to add the address of the new linked list into our previous node. If it is the first node, just set nextNode to be null.

    Before : head = new Node(word , new Node ());
    After : head = new Node(word);

  - ➔ Second Bug - It is in the while loop of the find() function. As the user can also put a null value in the linked list, getWord() function returns the word of that current node, so when it returns null and compares the value with our word that we have to look for using .equals() method, it got an error and resulted in crashing the program.

Before :

```
location = head;
while (!location.getWord().equals(word) && !location.isLast()) {
    location = location.next();
}

if (location.getWord().equals(word)) {
    found = true;
}
```

After :

```
location = head;
while (!location.isLast()) {

    if((location.getWord() == null && word == null) || (location.getWord() != null && location.getWord().equals(word))){
        found = true;
        break;
    }
    else{
        location = location.next();
    }
}

if(location.getWord() == null && word == null){
    found = true;
}
else{
    if (location.getWord() != null && location.getWord().equals(word)) {
        found = true;
    }
}
```

The approach that let you locate the defect.

➔ For Calendar, I didn't use any debug method as it is clear in the very first go, and for LinkedList, I use debug mode in VS Code (breakpoints, Pause, StepInto, StepOver, StepOut, and restart).

The approximate amount of time it took you to locate the defect.

➔ In Calendar, it just took me approximately 15 minutes to find both the above bugs. Moreover, for LinkedList, it took me one and a half hours.