Devang Patel
PLC Final

Programming Language Concepts Final

Github repo: : https://github.com/TreadKing/PLC/tree/master/Final

1. Final_1.java in github repo/zip file
2. Final_2.txt in github repo/zip file
3. Final_3.c in github repo /zipfile
4.
    a. 4 criteria to prove loop correctness and proof
        i. P -> I: The loop invariant must true initially
        ii. {I and B} S{I}: The loop invariant is not changing by executing the of the loop
        iii. (I and (not B)) -> Q: If I is true and B is false then Q is true
        iv. The loop terminates
    b. Proof
        i. The invariant in this problem is the B > 0. This is Ture initially.
        ii. After the loop executes, B > 0
        iii. After the loop b > 0, so the loop invariant is still true. After the loop, a = x^n, which is the same as Q.
        iv. When b <= n becomes false, thus the loop terminates.
5. Final_5.java: in github repo/zip file
   The readability of my code is about the same as original code. Both uses similar logic to iterate through the matrix. The difference is how it determines rejects. Instead of looking for looking for not zeros it looks for zeros and increments though the row till it hits a non-zero. One it hits a non-zero it breaks out of the row and moves to the next. If it increments to where j is equal to the length of the array, then it breaks out of incrementing though the matrix because then it knows it has pass through a row of all zeros. It then outputs “i” which is what row the program is on. This is less intuitive than reding though a go to statement because you have to run though the loop and see what break leaks where to determine what the output would be. The original code and the new code has the same time and space complexity because both could have to run though the whole matrix resulting in a complexity of n^2. The original code is slightly more compact and only has a minor advantage on readability. This means that it is up to the preference of the programmer on which to use.
6. Final_6_nested.java and Final_6_no_nested in github repo /zip file
   In the case of non-nested and nested both are reliable, but the nested version has higher reliability. This increases with scaling the input with more than 3 integers. This works similar to a branch and bound algorithm. The problem does not have to go down the whole decision tree in direction it knowns are already incorrect. This it is able to greatly reduce the time complexity, especially as the input size increases. The non-nested version has a worst case time complexity of n!, because it could be the last case checked. This is less efficient than nesting. The non-nested version has a readability advantage. It is very easy to see which conation al is being tested, while in the nested version it would take more time tracing the code.

Devang Patel
PLC Final

7. Tombstones have a higher time a space complexity, because tombstones are never deallocated from memory and reclaimed. This creates ineffective memory usage. Accessing a heap dynamic variable though this method requires an additional cycle to access. This introduces another level of time complexity. From a safety perspective it does prevent access to deallocated memory as the pointer will point to unchanging tombstone. The lock-and-key method has pointers represented as ordered pairs, keys and address. Heap dynamic variables are represented as the storage for the variable plus a cell that stores the "lock" integer value. When a Heap Dynamic variable is allocated a lock value is also created in the heap dynamic variable, the lock cell, and the key cell of the pointer. Accessing the dereferenced pointer compares the key value to the lock value. Matching values are allowed access. In the event of mismatched values, it returns a run-time error. When a heap dynamic variable is deallocated its lock value is set to an illegal value. When it's pointer dereferenced, tis address value will still be not change, but the key value will not work anymore, so access is denied. This prevents memory leaks and dangling pointers. However it does has time and space complexity overhead, all be it less than tombstones. Each block and pointer in the heap requires an extra word of memory space. Also, the comparing lock and key values has extra time complexity associated with it. The lock and key method has higher security associated as without the proper key it is impossible to access the locked memory. While with tombstone there is nothing preventing access to deallocated memory areas. This is easily done if you have the address of the memory segment you are looking for.

8.

    a. See Final_8_a.c in github repo/zip file
    b. See Final_8_b.c in github repo/zip file
    c. See Final_8_c.c in github repo/zip file
    d. See Final_8_d.txt in github repo/zip file
       j is initialized as -1, then it enters the loop. Then the program enters the switch stamen. This compares the value of j + 2 to different outcomes. If j +2 == 2 or 3 the switch statement executes j. If j + 2 == 0 then j+=2 is executed. If none of these are met, then j is set to 0. Then the program breaks out of the loop, then if j is less than or equal to 0 j = 3-i. then the next iteration of the loop executes.

9. I will be looking at swift for this question. Swift a C based language similar to Java. Both swift and java handle keywords similarly. Swift differs from Java in that Swift allows for keywords to be escaped out with backticks so it can be used as an identifier. This can reduce reliability by a small about, because keeping tack of which keywords are used as variables and which are not. This can cause mix-ups if the programmer does not keep proper track of the information. Because c is closely replanted to C and thus java, because it is based on objective C so it includes, he same data types as all of these languages. Swift does add tuples which allows for better reliability and readability. Both Swift and Java a type-safe language which prevents passing of the wrong variable type. This helps with reliability because it makes sure that the right data type is passed. Both

languages will inform you when you have type mismatching, so it is easy to fix. In relation to data storage Swift is closer to python, because when you declare a variable you do not have to specify a type to that variable. However, Swift used let instead of nothing. This can reduce reliability if someone assigns a variable wrong, but this can be easily fixed, when an error is presented. This also helps writability. Along with sharing many data type similarities with Java Swift, because of the common C relation, also shares seminaries in regard to control structures. Swift does have one distance change for control structures. Swift does not have implicit fall through in switch cases. After the fist case is met, the switch case ends on its own. This means explicit break statements are not required. This increases reliability if switch cases and makes them safer to use. Swift has four kinds of expressions: prefix, postfix, binary, and primary. Prefix and binary expression allow for operators to be applied to small expressions. Primary expression is the simplest expression and allows for accessing values. Postfix expressions allow for building of more complex expression using postfix. An example of this is function calls and memory access. The order operations for Swift is simplified compared to that of Java. Swift only has 6 division in its order of operations, while Java has 16. This greatly improves readability and writability. This could reduce reliability, because as a programmer can assume order of operations incorrectly and make a mistake. Swift a very powerful language and it improves on the readability and writability of other C based languages like Java. Both Java and Swift are related to C they will have very similar syntax. However, unlike C and Java swift does not require semicolons after each statement. However, it will allow it if there are multiple statements on one line. Also due to Swift having type interface, swift requires less type declarations. Like C, Swift has type aliasing, something that java does not have. Swift semantics are very strict, on some levels stricter than C. Immutable data types cannot be altered in any way. It will throw an error when this is attempted. So, a variable or constant object can modify all variable properties, but both constants and variable objects cannot modify constant properties. This also makes Swift a very powerful language and it improves on the readability and writability of other C based languages like Java.