Dhruv Rana
Professor Robila
CSIT313_01FA23
November 28, 2023

Homework 5

Task 1a) The remaining “else” is associated with the second “if”. That is because when multiple “if-else” statements are used, the “else” statement is associated with the nearest “if”. My code supports my statement because the condition of the first “if”: if(x > 2) is true because the integer variable x is defined with a value of 3 and so its associated print statement is executed printing “Prints First IF”. The second “if” condition: if(x == 2) is false and so its print statement is not executed but the “else” condition is executed, printing “Prints Else”. Since both the print statements associated with the first “if” and the remaining “else” were executed, the “else” must be associated with the second “if” and not with the first “if”.

Task 1b) To associate the “else” statement with the other “if” statement, the first “if” statement I moved the “else” above the second “if” statement. This would work because the “else” statement is associated with the nearest “if” so by moving the “else” above the second “if”, the closest “if” to the “else” is the first “if”. My code supports my answer because when the code is executed, the first “if” statement condition: if(x == 2) is false because integer variable x is defined with a value of 3 so its associated print statement is not executed and the else statement will be triggered, printing “Else Statement”. This means that the “else” statement is associated with the first “if” statement. The second “if" statement condition: if(x > 2) is true so the associated print statement is executed printing “Prints Second IF”, meaning that the “else” statement is not associated with the second “if” statement.

Task 2a) Java supports short-circuit evaluation of boolean expressions. My code supports my answer because the condition of the first “if” has been changed to incorporate boolean expression evaluation. The condition is: if (check == true || x > 2), where check is the boolean variable that is defined to be true, and the integer variable x is defined with the value of 1. The “if” statement uses short-circuit evaluation, since the variable check is always true, the first condition of the “if” statement is true. The “if” statement uses a “OR” operator, so it is not necessary to evaluate the second condition if the first is already true, so the second condition is not evaluated. When the code is executed, the “if” statement's overall condition is determined as true and it prints the associated print statement “Prints First IF”.

Task 2b) Java supports short-circuit evaluation of boolean expressions and so the Java code is implemented to enforce full evaluation. My code supports full evaluation because the condition of the first "if" has been changed to incorporate full evaluation. The condition is: if (check == true && x > 2), where check is the boolean variable that is defined to be true, and the integer variable x is defined with the value of 1. The "if" statement uses full evaluation, since the variable check is always true, the first condition of the "if" statement is true, but the "if" statement uses a "AND" operator, so the second condition must be checked for if it is also true, so the second condition is assessed. The second condition is false as the variable x has a value of 1, so variable x is not greater than the value 2, this means that the overall condition of the "if" statement is false, resulting in the execution of the "else" statement, printing "Prints Else".

Task 3c) The java code and c code I wrote is able to find the first row of an n by n integer matrix named x that has all zero values without using goto. Instead of using goto, both languages use for-loops and if conditions to accomplish the task. In Java and in C, a 3 by 3 matrix is hard coded, with the third row containing all zeros, and then using for-loops traverse the matrix. A difference between Java and C in creating an array is that in Java you are not required to specify an array size when you initialize the array with its elements, as Java determines the array size based on the elements given. However, in C you must specify the array size at creation even when you initialize with its elements. A boolean variable "found" is defined with the value true within the first, outer for-loop, so it is assumed that the matrix contains at least one row with all zeros. A problem I ran into in C when creating the boolean variable is that C does not have a built-in boolean data type. You need a import a header file for booleans using, #include <stdbool.h>, which is different from Java which has boolean data type as a part of the language. As the for-loops traverse the matrix, when a 0 is not encountered, found is set to false within the second, inner for-loop, and break from the for-loop. Instead of goto, I used an if condition to access if a row with all zeros is found, if so, it prints the row. In the case of the hard coded matrix, the output for the Java code and the C code is, 'First all-zero row is: 3'. In both languages, you can use the same approach without the goto statement to find the first row of an n by n integer matrix with all zero values.