# CS 4080 (Yang) Homework #3 (20 points)

**Special note:** You must cite reference(s) if any part of your answer is not your original work. If answers are from textbook, simply state textbook section number (e.g. text 5.1.2). No need to cite references if from lecture slides or in-class demonstration.

1. (4 points) Given a Java-like if statement

if (n >= 0 && factorial (n) mod 3 == 0) … print “got it”; else … print “uhuh”;

factorial (n) is defined as:

if (n == 0 or n == 1) return 1; else return n \* factorial (n – 1);

1. Under short circuit evaluation, what will be printed out if n = 3?

**“got it”**

1. Under short circuit evaluation, what will be printed out if n = -3?

**“uhuh”**

1. Under strict evaluation, what will be printed out if n = 3?

**“got it”**

(d) Under strict evaluation, what will be printed out if n = -3?

**Error, because even though n is not > 0, the second condition still gets evaluated. Factorial function of a negative number creates a stack overflow error.**

1. (6 points) Given the following Java-like if statement:

if (score > 80) // Note: indentation doesn’t play a role in Java semantics.

if (penalty < 3) System.out.println(“Nice”);

else System.out.println(“Okay”);

System.out.println(“Excellent”);

1. Without executing the following code, based on Java semantic rules for if statement, decide what will be printed out by this statement for each of the following cases:

(1) score = 85, penalty = 5 (2) score = 85, penalty = 2 (3) score = 70, penalty = 5

**Case 1:**

**Okay**

**Excellent**

**Case 2:**

**Nice**

**Excellent**

**Case 3:**

**Excellent**

(b) Add an *else* to the last line, redo above (1) to (3), what will be printed out now?

**Case 1:**

**Okay**

**Case 2:**

**Nice**

**Case 3:**

**Excellent**

1. (4 points) Loops:
2. Compare the following Java and Python for loops. Do they have the same output? If yes, say so. If no, explain why.

for (int ct = 0; ct < 100; ct++) { //Java

if (ct % 2 == 0)

ct + = 1;

System.out.println(ct)

for ct in range (100) : #Python

if ct % 2 == 0 :

ct += 1

print(ct)

**No they do not produce the same output. In the python for loop, we iterate on a range() function. Changing ct within the loop does not change the following iteration because ct resets with each iteration of the range() function.**

1. Rewrite the following Java do-while loop into a Java while loop. Assume all variables and functions are properly defined/initialized.

do

offset = (start % 2 == 0) ? f() : g();

total += offset;

while (offset != 0);

**offset = (start % 2 == 0) ? f() : g();**

**while (offset != 0) {**

**total += offset;**

**offset = (start % 2 == 0) ? f() : g();**

**}**

1. (2 points) Selection.
2. Rewrite the following code segment using a multiple-selection switch statement in Java. Assume k is an integer.

if (k == 1) || k == 6) result = 2 \* (++k) + 1;

else if (k == 2 || k == 3 || k == 5) result = 3 \* k - 1;

else if (k == 4) result = 4 \* k – 1;

else if (k > 7 && k < 14) result = (--k) – 2

else result = k;

print(result);

**switch (k) {**

**case 1:**

**case 6:**

**result = 2 \* (++k) + 1;**

**break;**

**case 2:**

**case 3:**

**case 5:**

**result = 3 \* k - 1;**

**break;**

**case 4:**

**result = 4 \* k - 1;**

**break;**

**default:**

**if (k > 7 && k < 14) {**

**result = (--k) - 2;**

**} else {**

**result = k;**

**}**

**}**

**System.out.println(result);**

1. Using the above example to describe (at least) one weakness in Java’s switch statement design and suggest a strategy for improvement. Show what your code look like (you may introduce new syntax if needed) after your improvement.

**One weakness in Java’s switch statement design is that it does not support ranges or conditions easily. Handling k>7 && k<14 required using an if statement within the default case, and making the default case the else statement.**

**switch (k) {**

**case 1, 6:**

**result = 2 \* (++k) + 1;**

**break;**

**case 2, 3, 5:**

**result = 3 \* k - 1;**

**break;**

**case 4:**

**result = 4 \* k - 1;**

**break;**

**case k > 7 && k < 14:**

**result = (--k) - 2;**

**break;**

**default:**

**result = k;**

**}**

1. (4 points) In a paper published in *CACM (1987)*, the following code segment is used as evidence that the readability of some code with gotos is better than the equivalent code without gotos (nor any unconditional branching statements.) This code finds the first row of an *n* by *n* integer matrix named x that has nothing but zero values.

for (i = 1; i <= n; i++) {

for (j = 1; j <= n; j++)

if (x[i][j] != 0) #number of comparisons that we’d count

goto reject;

println ('First all-zero row is:', i);

break;

reject:

}

1. Rewrite this code in Java without using any unconditional branching statements (e.g. break, exit, return etc.) nor you could change a for loop control variable to exit the loop (i.e. for (int j=0; j< n; j++) {if (cond) j = n; //such usage not allowed. However, you may use Boolean expressions as you wish.} Please keep in mind that your rewritten code shouldn’t increase the number of comparisons for x[i][j], i.e. keep the original algorithm idea.

boolean done = false;

for( i=1, i <= n && !done, i++) {

boolean allZero = true;

for( int j = 1; j <= n; i++) {

if (x[i][j] != 0) {

allZero = false;

}

}

if (allZero) {

System.out.println(“First all zero-row is: “ + (i+1));

done = true

}

}

(2) Write a complete program that uses the above rewritten code for the following given x matrix and output how many comparisons have been made. Note: add a counter to your code to count how many comparisons have been made. Attach screenshot of your code and execution result.

Use the following 10 x 10 matrix as input.

…

A screen shot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated