CIS 1068: Practice Problems 11

Some practice with basic loop algorithms: step-by-step loops, sentinel loops, accumulation loops, and nested loops.

1. Basic Step-by-Step Loops
A basic step-by-step loop has an int variable with a start value, an end value, and an update value. The algorithm for this kind of loop looks like this:

```java
int currentStep = START_VALUE;
while(currentStep < END_VALUE) {
    // do something, which may involve the value of currentStep
    currentStep += UPDATE_VALUE;
}
```

(or, equivalently)

```java
for( int currentStep = START_VALUE;
    currentStep < END_VALUE;
    currentStep += UPDATE_VALUE)
{
    // do something, which may involve the value of currentStep
}
```

a. Definite-length loops
In definite-length loops, the problem will tell you exactly how many times the loop should execute.

1. Print “I will not use a variable without declaring and initializing it first” 100 times on the screen.

2. Create an array of 20 ints. Set the value of the first element to 1, the next element equal to 2, etc., so that the last element equals 20.

3. Create an array of 20 Point objects. Initialize each Point object so that its x coordinate is a random number between 1 and 10, and its y coordinate is a random number between -5 and 5.

b. Indefinite-length loops

1. Read in an int N from the keyboard. Print all of the numbers between 1 and N.

2. Read in an int N from the keyboard. Create an array of N ints, and store the numbers between 1 and N in the array.
3. Generate a random number N between 1 and 100. Print every multiple of 7 between 1 and N (eg., if N = 31, it should print 7, 14, 21, and 28).

4. Generate a random number N between 1 and 100. Read a character C from the keyboard. Create a char array of length N, and initialize every element to the character stored in variable C. Construct a String from the char array.

2. Sentinel loops
A sentinel loop keeps track of a boolean variable, and keeps going until that boolean variable becomes false. Here is the algorithm:

```java
boolean sentinel = true;
while(sentinel) {
    // do some stuff
    if(<test>) {
        sentinel = false;
    }
}
```

(or, an alternate version)

```java
while(true) {
    // do some stuff
    if(<test>) {
        break;
    }
}
```

a. Read Strings from the keyboard and print their length, until the user enters the word “STOP!” (all caps).

b. Generate a random number between 1 and 10. Ask the user to guess the number, and keep going until they get it right.

3. Accumulation loops
Accumulation loops keep track of and update information each time through the loop. Usually, if a problem says find the “total” or “average” or “maximum” or “biggest” or “anything-est”, then you need an accumulation loop.

Accumulation loops are add-ons to either step-by-step or sentinel loops. You need to start with either a step-by-step or sentinel loop, and then create an extra variable to accumulate extra information.
Here is an algorithm for a basic accumulation step-by-step loop.

```java
int currentStep = START_VALUE;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep < END_VALUE) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += UPDATE_VALUE;
}
```

The data type of the accumulator depends on what kind of information you’re trying to accumulate. Same for the current_value, and the way in which the current_value combines with the existing accumulator value to create the new accumulator value.

For instance, if you’re trying to find the total of an int array, then accumulator is type int, current_value = array[currentStep], and accumulator = accumulator + current_value;

If you’re trying to find the longest String in a String array, then accumulator is type String, current_value = array[currentStep], and accumulator = current_value.length() if current_value.length() is longer than accumulator.length().

a. Read in 10 numbers from the keyboard and compute their average (the accumulator should be the total, then divide by 10 after the loop to get the average).

b. Read in 10 Strings from the keyboard and compute the total length of all of them.

c. Read in Strings from the keyboard until the user types “STOP!”. Compute the total length of all of them. (Note: this is an accumulation loop with a sentinel loop, not an accumulation loop with a step-by-step loop.)

d. Read in 10 numbers from the keyboard and count how many are odd (or also try counting how many are divisible by 7, or how many are not divisible by either 12 or 19).

e. Read in x and y coordinates of Point objects. Find the Point that is farthest from the origin (or new Point(0,0)). Stop when the user enters 0 for x and 0 for y.

f. Create an array of 100 doubles, and initialize the array with random Gaussian values (use the nextGaussian() method from the Random class, which returns a double). Compute the mean (average) and standard deviation of the array. If the mean is M, then you can compute the standard deviation by computing \( S = \sum_i (x[i]-M)^2 \), and then taking the square root of S.

g. Create an array of 100 Strings, and initialize the array by generating a random integer between 0 and 1 million for each element, and converting the integer to a String. Find the longest String in the array.
**Nested Loops**

Nested loops are the toughest ones we have seen. They combine an inner loop (which repeats some stuff X times) with an outer loop (which repeats the whole inner loop Y times). As a result, the inner loop’s stuff gets repeated \( X \times Y \) times.

A nested loop can combine an inner step-by-step loop with an outer sentinel loop, or an inner step-by-step loop with an outer step-by-step loop, or any other combination. It depends on the problem.

Some common versions:
- Displaying two-dimensional figures (ASCII art) requires inner and outer step-by-step loops.
- Computing with 2D arrays normally requires inner and outer step-by-step loops, where the inner loop repeats as many times as the number of columns and the outer loop repeats as many times as the number of rows.
- File processing often involves nested sentinel loops (we’ll get to those later).

It’s also possible to combine accumulation loops with nested loops.

a. Create a 10 x 10 2D array of Point objects, initialized randomly.

b. Create a 8 x 7 2D array, and initialize it randomly (use either the nextGaussian() or nextInt() method from the Random class, you choose). Then compute the sum, average, largest, and smallest elements in the array. (Initialization is a nested step-by-step loop, the rest is a nested step-by-step loop plus accumulation).

c. Create an array of 10 Strings, each initialized with random integers between 0 and 1 million (converted to Strings). Count how many 0’s appear in all of the Strings. (Initialization is a single step-by-step loop. Counting is a nested step-by-step loop with accumulation.)

d. Create a 4 x 2 array A, a 2 x 3 array B, and a 2 x 2 array C. Initialize A and B to random values (either nextGaussian() or nextInt(), you can choose). Then store the result of A * B in C, where A* B is matrix multiplication. That means, for element C[row][column] you add up \( \sum_j A[\text{row}][j] \times B[j][\text{col}] \). (Double-nested step-by-step loops with accumulation)