CIS 110 Exam 1, Spring 2021

Q1. True/False

1. `int x = 4.0;` causes an error.

2. `double x = '4';` causes an error.

3. The following code compiles:

```java
int i = 0;
if (3 < 21) {
    i++;
} else {
    i = i - 2;
} else if (10 > 3) {
    i = 1 / 9;
}
```

4. The following code compiles:

```java
int i = 0;
if (3 < 21) {
    i++;
} else (10 < 3) {
    i = i - 2;
}
if (10 > 3) {
    i = 1 / 9;
}
```

5. The following code would compile if sequence has type String but NOT if sequence has type String[]:

```java
for (int i = sequence.length(); i >= 0; i--) {
    System.out.println(sequence.charAt(i));
}
```

6. Running the following code causes a white rectangle to be visibly drawn over a black background:

```java
PennDraw.setPenColor(PennDraw.WHITE);
PennDraw.filledRectangle(0.5, 0.5, 0.1, 0.3);
PennDraw.clear(PennDraw.BLACK);
```
7. A loop control variable must be updated with every iteration of a while loop.

8. "c" + "d" evaluates to "cd".

9. For a given array, \( arr, arr[\text{arr.length}] \) evaluates to the last element in the array.

10. If a test case for a unit of code passes, that means that there are definitely no bugs in that unit of code.

Q2. Short Fill in the Blanks

1. What is the return type of this function \( \text{returnsBoolean} \)?

```java
public static double returnsBoolean(int character) {
    int x = 3 - character;
    int y = 4;
    int str = x + y;
    return str;
}
```

2. The _____ in a recursive function, unlike the recursive case, does not make any more recursive calls and instead allows Java to start clearing activation frames off of the call stack.

3. Look at the following code snippet:

```java
char b = ______;
if ('q' < b) {
    if (b > 's') {
        System.out.print("R");
    } else {
        System.out.print("C");
    }
    if ('s' > b) {
        System.out.print("U");
    } else if ('s' == b) {
        System.out.print("T");
    }
} else if (b > 'q') {
    System.out.print("Z");
}
```

What value of \( b \) would cause the program to print "RT"? If there is none, type none.

-----

What value of \( b \) would cause the program to print "CU"? If there is none, type none.

-----

What value of \( b \) would cause the program to print "CT"? If there is none, type none.
What value of `b` would cause the program to print "$z"? If there is none, type none.

Q3. Long Fill in the Blanks

SEAS just decided that they want to fix their servers but broke everything even worse by wiping the entire mainframe! They’ve asked you to help them write a function to reset the servers before all their data is lost!

This function is supposed to take a single command line argument of an integer, add 7 and divide it by 2. Then this function will determine if the calculated number is a multiple of 3. If it is, it will print “Success”. Otherwise, it will print “Try again”. Fill in the blanks so that this program compiles and runs as specified:

```java
public static void main(String[] args) {
    // Convert first command line argument to an int
    int x = ____1____; //#1
    int y = ____2____; //#2
    // Figure out what message to print
    ____3____ message; //#3
    if (y ____4____ 3 == ____5____) { //#4, #5
        message = "Success";
    }
    message = "Try again";
    System.out.println(message);
}
```

Q4. Niven Numbers

A niven number is a number that is divisible by the sum of its digits. Write a function that recursively returns the minimum niven number given an integer array, or returns `Integer.MAX_VALUE`.

You are given a function, `sumDigits`, which takes an integer input `n` and returns the sum of the digits in `n`. For instance, `sumDigits(12) = 3`, `sumDigits(35) = 8`, `sumDigits(2) = 2`.

Now, please write the following two functions:

```java
public static boolean isNiven(int n) which, using the above sumDigits function, returns a boolean if a number is a Niven number: isNiven(7) = true, isNiven(35) = false [35 is not divisible by 8], isNiven(36) = true [36 is divisible by 9]
```

```java
public static int minNivenNum(int[] arr, int index) which, using recursion, returns the smallest Niven number, or the value Integer.MAX_VALUE if the array contains no niven numbers. For instance, minNivenNum({15, 7, 10}, 0) = 7, minNivenNum({35, 36, 51}, 0) = 36.
```
Q5. Debugging

Oh no! The University’s Covid Test Registration System got a virus and is no longer sending a message to students if they don’t get their Covid tests!!!! Help Penn protect us by fixing the bugs in the below function (and by socially distancing, wearing masks, staying inside whenever possible)!

The below function is meant to take in a boolean array where each index is true if person \( i \) has taken their Covid test. It returns a string containing all of the people who haven’t taken their test (they will all get a strongly worded email by Amy Gutmann very soon). If there are more than 5 people who haven’t taken their Covid test, we want to print to the console saying “AHHHHHHH”.

There are seven lines containing bugs in the following function; however, a line may include more than one bug. For each bug, identify the line number, provide a very brief explanation of the bug, and rewrite the line to fix the bug.

```java
1. public static String covidTests(boolean people) {
2.     int count = 0;
3.     String nonTesters;
4.     int len = people.length()
5.     for (int i = 0, i < len, i++) {
6.         if (people[i]) {
7.             count++;
8.             nonTesters = nonTesters + i + ", ";
9.         }
10.     }  
11.     if (count > 5) {
12.         System.out.println(AHHHHHHHHHH);
13.     }
14.     return count;
15. }
```

Please include all errors in the following format, with each error on its own line: Line Number / Description of error / Correct code (Example: Line 42 / Doesn’t parse string x for integer / `Integer.parseInt(x)`)

Q6. Long Coding

Dinosaur Game is a small browser game built into the Google Chrome web browser. In the game, you play as a tiny T-Rex running along a desert landscape, bravely jumping over perilous obstacles. The game has an avid following among our TAs, and so they'd like to start making their own custom versions. You've been tasked with using your programming skills to help them build the courses that the dinosaur has to run along.
To build Dinosaur Game, we'll implement the courses as arrays of doubles. Each position in the array indicates the height of an obstacle inside that position on the course. For example, we might have the following example of an array:

```java
double[] course = {0, 0, 0, 0, 0, 4.0, 0, 0, 0, 0, 0, 0, 0, 2.6, 2.8, 2.4, 0, 0, 0};
```

If we were to visualize this course, we'd see mostly flat ground with two major obstacles. **We define an obstacle as a consecutive sequence of cells with non-zero heights.** The first obstacle starts and ends at course[6]. The next obstacle is wider, ranging from course[14] to course[16].

```java
1. public static double[] pullSliceFromCourse(int start, int length, double[] course)
```
To generate a never-ending track for the dinosaur to run on, we'll implement a function `pullSliceFromCourse` that returns a copy of a sequence (called a `slice`) from an array of doubles `course` starting at index `start` with a length of `length`. Keep in mind, a slice can be as small as one item.

`pullSliceFromCourse` should *wrap around* the input array `course` if necessary. Wrapping around means that when `start + length >= course.length`, then we start pulling indices from the beginning of the array after we hit the end of it.

For example, if `double[] course = {0, 0.3, 0.4, 0, 1.2}`:

- `pullSliceFromCourse(0, 1, course)` should return the array `{0}`
- `pullSliceFromCourse(2, 3, course)` should return the array `{0.4, 0, 1.2}`
- `pullSliceFromCourse(2, 5, course)` should return the array `{0.4, 0, 1.2, 0, 0.3}` *(this is a wraparound case, and the sequence of elements we pull is `{course[2], course[3], course[4], course[0], course[1]}`)*
- `pullSliceFromCourse(0, course.length, course)` should return an array identical to `course`.

You can assume that `length > 0, 0 <= start < course.length, and course.length > 0`.

2. public static boolean checkValidSlice(double[] slice)

We also need some way to check to make sure that a particular slice input is valid. If any obstacle is too long or too tall, then the dinosaur will never be able to jump over them! We say that an obstacle is *too long* if it takes up more than three consecutive indices. For example:

- if `double[] slice = {0, 0.3, 0.4, 0, 1.2, 0, 2.0, 2.0, 2.0}`, `checkValidSlice(slice)` returns `true`. There are three obstacles, and they take up only two, one, and three consecutive positions in the array.
- if `double[] slice = {0, 0.3, 0.4, 0.9, 1.2, 0, 2.0, 2.0, 2.0}`, `checkValidSlice(slice)` returns `false`. There are only two obstacles this time, but the first takes up four consecutive positions in the array.

We say that a slice contains an obstacle that's *too tall* if any of the heights in `slice` are greater than 4.0. For example:

- if `double[] slice = {0.4, 0, 1.2}`, `checkValidSlice(slice)` returns `true`.
- if `double[] slice = {4.0, 4.1, 0, 1.0, 1.1}`, `checkValidSlice(slice)` returns `false` because `slice[1]` is 4.1.

You can assume that `slice.length > 0`.

Hint: Consider using a boolean variable to track whether or not you're currently in an obstacle.

3. public static double[] generateNextPortion(int length, double[] course)

Now that you've written the previous two functions, you can complete the following function `generateNextPortion(int length, double[] course)` to generate the next randomized sequence of the course for the T-Rex to run across. Specifically, `generateNextPortion` should call `pullSliceFromCourse` to choose a slice of course of length `length` starting at a random starting index chosen uniformly between 0 and `course.length - 1`, inclusive. The slice should be checked for its validity using `checkValidSlice`. If the slice is a valid one, return it. Otherwise, if the slice is not valid, print
"default" and return an array of length length containing just 0s. Make sure to use the functions you already wrote to do this: you will lose points if you do not.

Hints:

- You don't need to have solved parts 1-2 correctly to get points here!
- Use Math.random() to generate a random double between 0 and 0.999.... Use this result to choose a random starting index between 0 and course.length - 1.