CodeU Coding Exercises - Session 1

Arrays and Matrices

Exercise 1 (DO THIS): Array “Zip” or “Join”:

Use the following class “Zip” as your starting point, and provide an implementation for “join”.

```
class Zip {
  // Fill in the method "join". It returns a boolean array. The ith
  // value is that array (i.e., array[i]) should be true if the ith
  // value in the first argument to join is divisible by the ith value
  // in the second argument to join. The returned boolean array should
  // be exactly as long as the shorter of the two arguments.
  //
  // Reminders:
  //
  // 1. An integer p is said to be "divisible by" an integer q when there
  // is some integer k such that q*k = p. This is the same as saying
  // "the remainder of p when divided by q is 0". The remainder
  // operator is Java is written with a percent sign: "a % b" is the
  // remainder of a when divided by b.
  //
  // 2. The length of an array bar is stored in bar.length.
  //
  // 3. New arrays are declared with the syntax:
  //        float[] foo = new float[8];
  //
  static boolean[] join(int[] y, int[] z) {
    // STUDENTS: WRITE YOUR CODE HERE!
  }

  public static void main(String[] args) {
    //
    // Expected output:
```

```
    //  false
    //  false
    //  false
    //  false
    //  true
    //  false
    //  true
    //
    // STUDENTS, ADD ADDITIONAL TEST CASES BELOW
    //
    int euler[] = {2, 7, 18, 28, 18, 28, 45, 90, 45};
    int jenny[] = {8, 6, 7, 5, 3, 0, 9};
    boolean divisibles[] = join(euler, jenny);
    for (int i = 0; i < divisibles.length; ++i) {
      System.out.println(divisibles[i]);
    }
  }
```

Exercise 2 (DO THIS): Array Filtering

Use the following class “Filter” as your starting point, and provide an implementation for “evens”.

```
public class Filter {
  // Write a function named "evens" that takes as input an array of
  // ints and returns a different array of ints containing
  // only the even elements of the input.
  public static int[] evens(int[] input) {
    // Here are some reminders:
    //
    // You can find input's length using input.length.
    //
    // You can find the remainder of a division using %. For instance,
    //   11 % 3 ⇒ 2
    //   25 % 4 ⇒ 1
    //
    // You can declare a new array of integers with the syntax:
    //   int[] var_name = new int[n];
    //
    // For example:
    //   int[] clown = new int[10];
    // creates an array named clown of 10 integers (clown[0] through clown[9])
    //

    // STUDENTS, WRITE CODE HERE.
  }
```

```
  public static void main(String[] args) {
    //
    // Expected output:
    //  test1 results:
    //  8
    //  6
    //  0
    //  test2 results:
    //  2
    //  18
    //  28
    //  18
    //  28
    //  90
    //
    // STUDENTS, ADD ADDITIONAL TEST CASES BELOW
    //
    int[] test1 = {8,6,7,5,3,0,9};
    int[] ans = evens(test1);
    System.out.println("test1 results:");
    for (int i = 0; i < ans.length; ++i) {
      System.out.println(ans[i]);
    }
    int [] test2 = {2,7,18,28,18,28,45,90,45};
    ans = evens(test2);
    System.out.println("test2 results:");
    for (int i = 0; i < ans.length; ++i) {
      System.out.println(ans[i]);
    }
  }
}
```

Exercise 3 (OPTIONAL): Collatz Sequences

```
class Collatz {
  // Consider a sequence of positive integers starting with x. If x is
  // even, the next integer in the sequence is x/2. If x is odd, the
  // next integer in the sequence is 3*x+1. The sequence stops when it
  // reaches 1.
  //
  // For example, if x is 7, the sequence is
  //
  // 7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
  //
  // Fill in the function loopCount so that it returns the length of
```

```
  // the sequence starting from x.
  static int loopCount(int x) {
    // STUDENTS: FILL IN CODE HERE!
  }

  // Using loopCount, fill in the function maxLoop so that it returns
  // the maximum sequence length for any sequence that starts with a
  // number greater than or equal to x and less than y.
  static int maxLoop(int x, int y) {
    // STUDENTS: FILL IN CODE HERE!
  }

  public static void main(String[] args) {
    System.out.println(maxLoop(1,100000));
  }
}
```

Exercise 4 (OPTIONAL): Sparse Matrix Output

In some applications of 2-dimensional arrays or matrices, the matrix may be very large, but the data may be very *sparse* (i.e., most of the values are zero). In such cases it may be far more efficient to input or output only the non-zero values.

Write a function that accepts an integer matrix as input and generates a line of output for each non-zero value in the matrix. Each line should have the following format:

```
[row_number, column_number]: value
```

For example, for the following matrix:

```
| 0 0 0 0 |
| 0 6 0 0 |
| 8 0 0 4 |
```

the expected output would be:

```
[1, 1]: 6
[2, 0]: 8
[2, 3]: 4
```

Also write a main routine that creates several test matrices and calls your function to output them.

Please Help - Share your Feedback:

[Session 1 Feedback Form]