Name:

YOU MAY READ THIS PAGE BEFORE THE EXAM BEGINS

1. You have 75 minutes to create and submit the 2 programs in this exam.
2. When instructed to start, download and extract the Dummy IntelliJ Project under the `~/workspace` folder if you do not have it there already.

   `https://www.cs.umb.edu/~siyer/teaching/cs210/dummy_project.zip`

3. Open the Dummy Project in IntelliJ. To create a program, right-click on the `src` folder under `dummy_project` in the top-left window and then select the *New* → *File* menu. Enter the name of the program in the pop-up window. Note that the name is case-sensitive and must match the suggested name exactly.
4. You may use the text, your notes, your code from the projects, and the code on the CS210 course website. No form of communication is permitted (eg, talking, texting, etc.) during the exam, except with the course staff.
5. Submit your work on Gradescope under the assignment named `"Sample Programming Exam 1"`.
6. Return this exam sheet to the course staff with your name written at the top. Failing to do so will void your exam submission on Gradescope.
7. You are *not* allowed to leave the exam hall before the official end time even if you are done early.
8. Your programs will be graded based on correctness, clarity (including comments), design, and efficiency.
9. Discussing the exam contents with anyone who has not taken the exam is a violation of the academic honesty code.

DO NOT READ FURTHER OR DOWNLOAD THE DUMMY PROJECT UNTIL SO INSTRUCTED

Problem 1. (18 Points) Implement a comparable, iterable data type called `Genome` that represents a genome sequence (a string of letters A, T, G, or C denoting nucleotides), and supports the following API:

`Genome implements Comparable<Genome>, Iterable<Character>`

`Genome(String s)`	constructs a `Genome` object from the genome sequence `s`
`double gcContent()`	returns $\frac{G+C}{A+T+G+C} \times 100$ for this genome; for example, GC content of the genome sequence "ACTGCG" is 67%
`String toString()`	returns a string representation of this genome in `"<length>:<sequence>"` format; for example, the genome sequence "ACTGCG" is represented as `"6:ACTGCG"`
`int compareTo(Genome other)`	returns a comparison of this and `other` genomes based on their lengths
`static Comparator gcOrder()`	returns a comparator for comparing genomes based on their GC content
`Iterator iterator()`	returns an interator for iterating over this genome in *reverse* order

Enter the following starter code in the program and replace the ellipsis (...) with your code.

Genome.java

```
import stdlib.StdOut;

import java.util.Comparator;
import java.util.Iterator;

public class Genome implements Comparable<Genome>, Iterable<Character> {
    private String s; // the genome sequence

    public Genome(String s)  { ... }

    public double gcContent() { ... }

    public String toString() { ... }

    public int compareTo(Genome other) { ... }

    public static Comparator<Genome> gcOrder() { ... }

    public Iterator<Character> iterator() { ... }

    private static class GCOrder implements Comparator<Genome> {
        public int compare(Genome g1, Genome g2) { ... }
    }


    private class ReverseIterator implements Iterator {
        private int i; // index of current letter

        public ReverseIterator() { ... }

        public boolean hasNext() { ... }

        public Character next() { ... }
    }

    // Unit tests the data type.
    public static void main(String[] args) {
        Genome g1 = new Genome(args[0]);
        Genome g2 = new Genome(args[1]);
        StdOut.println(g1);
        StdOut.println(g2);
        StdOut.println(g1.gcContent());
        StdOut.println(g2.gcContent());
        StdOut.println(g1.compareTo(g2));
        StdOut.println(Genome.gcOrder().compare(g1, g2));
        for (char c : g1) {
            StdOut.print(c);
        }
        StdOut.println();
    }
}
```

~/workspace/dummy_project

```
$ javac -d out src/Genome.java
$ java Genome ACTGCG GAACTTAGC
6:ACTGCG
```

```
9:GAACTTAGC
66.66666666666667
44.44444444444444
-1
1
GCGTCA
```

Problem 2. (7 Points) Implement the function `private static void closestPair(int[] a)` in `CloestPair.java` such that it prints the closest pair of integers in `a`, separated by a space. Enter the following starter code in the program and replace the ellipses (...) with your code.

ClosestPair.java

```
import java.util.Arrays;

import stdlib.StdIn;
import stdlib.StdOut;

public class ClosestPair {
    // Entry point.
    public static void main(String[] args) {
        int[] a = StdIn.readAllInts();
        closestPair(a);
    }

    private static void closestPair(int[] a) { ... }
}
```

~/workspace/dummy_project

```
$ javac -d out src/ClosestPair.java
$ java ClosestPair
4 9 3 -1 6
<ctrl-d>
3 4
```

Files to Submit

1. `Genome.java`
2. `ClosestPair.java`

Answers

Problem 1.

Genome.java

```
import stdlib.StdOut;

import java.util.Comparator;
import java.util.Iterator;

// This comparable, iterable data type represents a genome sequence (a string of letters A, T, G,
// or C denoting nucleotides).
public class Genome implements Comparable<Genome>, Iterable<Character> {
    private String s; // the genome sequence

    // Constructs a Genome object from the genome sequence s.
    public Genome(String s) {
        this.s = s;
    }

    // Returns (G + C) / (A + T + G + C) * 100 for this genome. For example, the GC content of
    // the genome sequence "ACTGCG" is 67%.
    public double gcContent() {
        int gc = 0;
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == 'G' || s.charAt(i) == 'C') {
                gc++;
            }
        }
        return 100.0 * gc / s.length();
    }

    // Returns a string representation of this genome.
    public String toString() {
        return s.length() + ":" + s;
    }

    // Returns a comparison of this and other genomes based on their lengths.
    public int compareTo(Genome other) {
        int x = this.s.length();
        int y = other.s.length();
        if (x < y) {
            return -1;
        } else if (x == y) {
            return 0;
        } else {
            return 1;
        }
    }

    // Returns a comparator for comparing genomes based on their GC content.
    public static Comparator<Genome> gcOrder() {
        return new GCOrder();
    }

    // Returns an iterator for iterating over this genome in reverse order.
    public Iterator<Character> iterator() {
        return new ReverseIterator();
    }

    // A comparator for comparing genomes based on their GC content.
    private static class GCOrder implements Comparator<Genome> {
        // Returns a comparison of genomes g1 and g2 based on their GC content.
        public int compare(Genome g1, Genome g2) {
            double x = g1.gcContent();
            double y = g2.gcContent();
            if (x < y) {
                return -1;
            } else if (x == y) {
                return 0;
            } else {
                return 1;
            }
        }
    }


    // An iterator for iterating over a genome in reverse order.
```

```
    private class ReverseIterator implements Iterator {
        private int i; // index of current letter

        // Constructs an interator.
        public ReverseIterator() {
            this.i = s.length() - 1;
        }

        // Returns true if there are more letters in the genome, and false otherwise.
        public boolean hasNext() {
            return i >= 0;
        }

        // Returns the next letter in the genome.
        public Character next() {
            return s.charAt(i--);
        }
    }

    // Unit tests the data type.
    public static void main(String[] args) {
        Genome g1 = new Genome(args[0]);
        Genome g2 = new Genome(args[1]);
        StdOut.println(g1);
        StdOut.println(g2);
        StdOut.println(g1.gcContent());
        StdOut.println(g2.gcContent());
        StdOut.println(g1.compareTo(g2));
        StdOut.println(Genome.gcOrder().compare(g1, g2));
        for (char c : g1) {
            StdOut.print(c);
        }
        StdOut.println();
    }
}
```

Problem 2.

ClosestPair.java

```
import java.util.Arrays;

import stdlib.StdIn;
import stdlib.StdOut;

public class ClosestPair {
    // Entry point.
    public static void main(String[] args) {
        int[] a = StdIn.readAllInts();
        closestPair(a);
    }

    // Prints the closest pair of integers in a, separated by a space.
    private static void closestPair(int[] a) {
        Arrays.sort(a);
        int minDist = Integer.MAX_VALUE;
        int u = -1, v = -1;
        for (int i = 1; i < a.length; i++) {
            int dist = (int) Math.abs(a[i - 1] - a[i]);
            if (dist < minDist) {
                minDist = dist;
                u = a[i - 1];
                v = a[i];
            }
        }
        StdOut.println(u + " " + v);
    }
}
```