UIL Computer Science
Programming Contest
Using the Judging Environment; Includes Judging Guidelines

Adapted from the UIL Hands-on README.

Introduction: These instructions are for using the judging environment for the UIL Computer Science programming contest. You are not required to use the provided judging environment. It is provided as one option for contest directors to choose from, and can be a good solution for directors and schools that are new to judging a computer programming contest. These instructions assume the judging environment has already been installed. **It is <u>strongly suggested</u> that contest directors practice using the judging program <u>prior</u> to the day of the contest. You may encounter difficulties installing and running the `judge.bat` files depending on the security measures in place on the computer you are using.** Please consult your campus computer / information technology expert if you encounter problems. For testing you may use the dry run solution or solutions to actual problems, which are included on the judging CD.

Location of Contestants' Submissions: Create a folder called UILSolutions at the top level of the C drive on your judging station. If you wish to judge solutions from a different directory, you'll need to alter the `judge.bat` file. The `judge.bat` file and the judging environment installation instructions both contain explanations of how to set this value.

Judging Program Overview. The judging script performs the following basic actions:

1. Archives the contestant program
2. Copies the contestant program to a clean directory
3. Compiles the contestant program
4. Executes the contestant program
5. Compares the contestant program output to the expected output

During the 5th step, the Windows program CSDiff is invoked to compare the output from the contestant program to the expected output. In the best case, CSDiff will display a dialog box with the message:
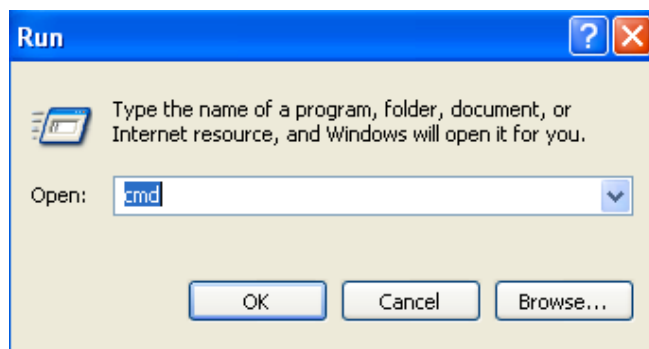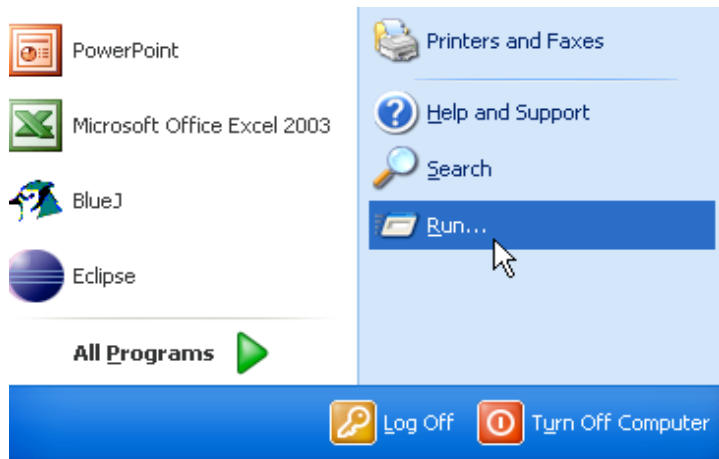
"Revisions are identical or differ by white space only!"

This means that the output was correct and that no inspection is required by a judge.
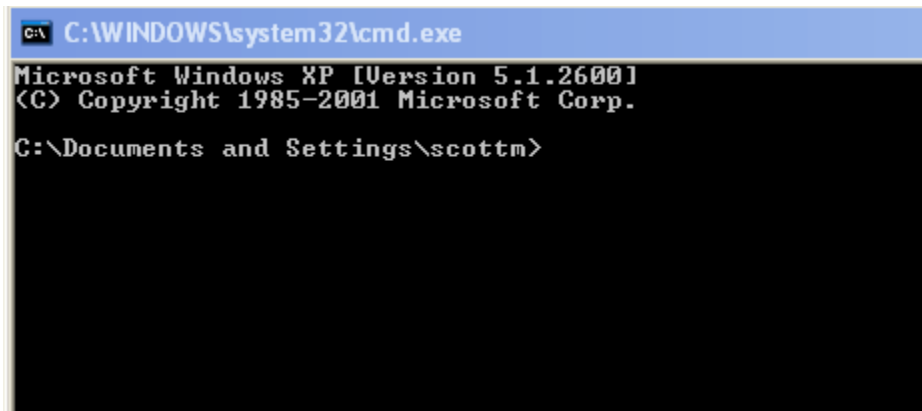
If there are disparities between actual and expected output, CSDiff will instead display a window which highlights the differences between the output produced by the contestant program and the expected output. This display can be confusing, so I urge all judges to experiment with the interface during the dry run.
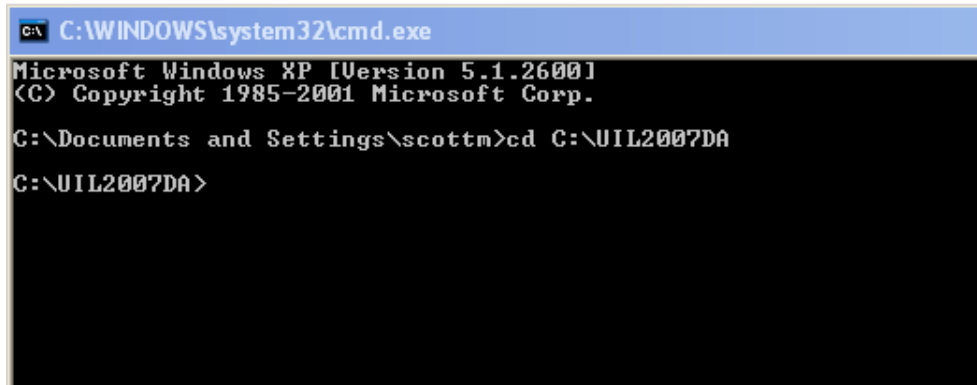
Example of Using the Judging Program:

1. Start the judging program by connecting to the `C:\UILProgramming` directory **(note that this directory name replaces the `UIL2007DA` directory name indicated in some of the screen captures).**

2. You can do this by running the `cmd` command from the Windows start menu.
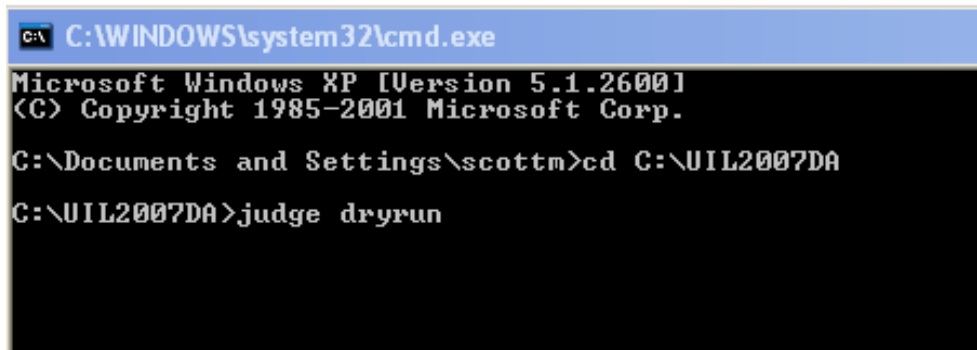

3. This opens a command window.

4. Move to the directory that contains the judging program by typing the commands
`cd C:\UILProgramming`.

```
C:\WINDOWS\system32\cmd.exe

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\scottm>cd C:\UIL2007DA

C:\UIL2007DA>
```
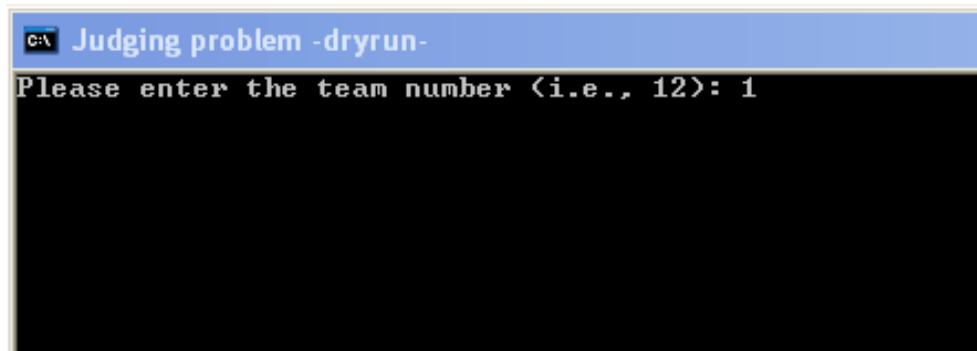
5. After moving to the correct directory, run the judging program by typing the command `judge [problem]` where `[problem]` is the name of the problem being judged. In this example I am judging the dry run problem. The dry run problem explanation is included as the last page of these instructions. To judge the dry run problem I type the command `judge dryrun`.

```
C:\WINDOWS\system32\cmd.exe

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\scottm>cd C:\UIL2007DA

C:\UIL2007DA>judge dryrun
```
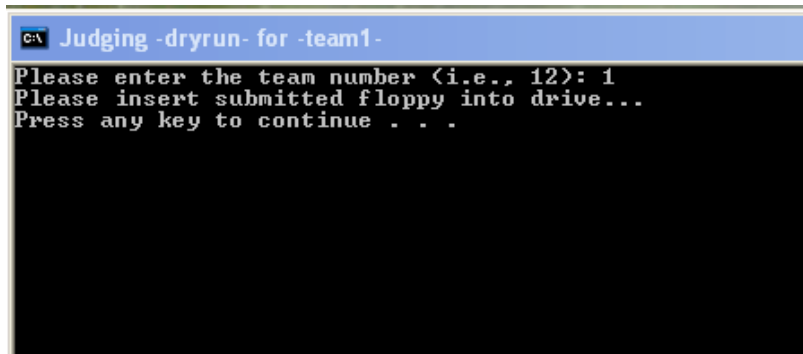
6. The judging program will now run through its steps. The first step asks for the team number. This is to archive the contestants' solution.
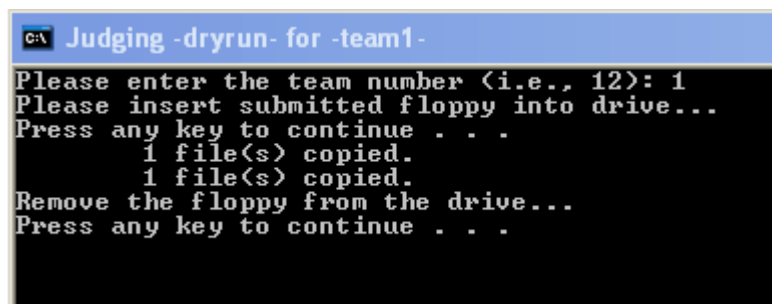
```
Judging problem -dryrun-
Please enter the team number (i.e., 12): 1
```

7. The judging program now asks you to insert the floppy into the drive. (Since you're judging from a defined location on the C drive, you can just continue.)
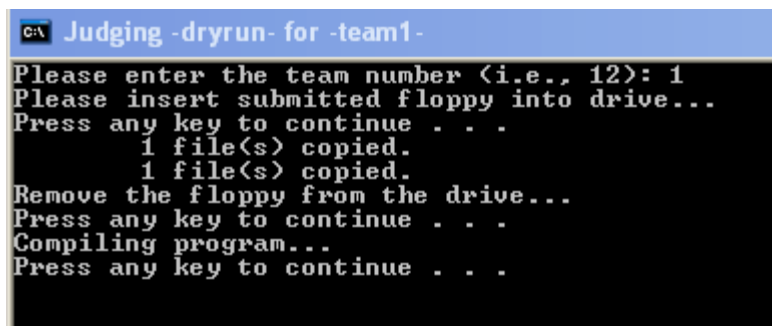
```
■ Judging -dryrun- for -team1-
Please enter the team number (i.e., 12): 1
Please insert submitted floppy into drive...
Press any key to continue . . .
```

8. Press any key. The contestant's solution is archived and the judging program asks you to remove the floppy. (If reading from the hard drive itself there is no action required at this point.)

```
■ Judging -dryrun- for -team1-
Please enter the team number (i.e., 12): 1
Please insert submitted floppy into drive...
Press any key to continue . . .
        1 file(s) copied.
        1 file(s) copied.
Remove the floppy from the drive...
Press any key to continue . . .
```

9. The judging program will now compile the program. Contestants are to submit the source code for their programs, the .java file, not the .class file. If the file compiles the following appears:

```
■ Judging -dryrun- for -team1-
Please enter the team number (i.e., 12): 1
Please insert submitted floppy into drive...
Press any key to continue . . .
        1 file(s) copied.
        1 file(s) copied.
Remove the floppy from the drive...
Press any key to continue . . .
Compiling program...
Press any key to continue . . .
```

If the program contained a compile error then you will see an error message and the judging program will stop. In this case mark "Does Not Compile" on the Run Sheet and return the attempt. This counts as an incorrect attempt.

Example of a compile error:

10. If the contestant's solution compiles with no errors press any key and the judging program will run the contestants' solution against the judging data. If the contestant's solution runs with no runtime errors the judging program will record the output and you will see the following:

If the contestant's program suffers a run time error you will see an error message and the judging program will stop.

Example of a program that suffers a runtime error.


```
C:\ Judging -dryrun- for -team1-
Please enter the team number (i.e., 12): 1
Please insert submitted floppy into drive...
Press any key to continue . . .
        1 file(s) copied.
        1 file(s) copied.
Remove the floppy from the drive...
Press any key to continue . . .
Compiling program...
Press any key to continue . . .
Running program...
        1 file(s) copied.
Exception in thread "main" java.io.FileNotFoundException: C:\dryrun.in
em cannot find the file specified)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(FileInputStream.java:106)
        at java.util.Scanner.<init>(Scanner.java:621)
        at dryrun.main(dryrun.java:6)
Judgement - RUNTIME ERROR
Terminating...
Press any key to continue . . .
```

If the program contained a runtime error then you will see an error message and the judging program will stop. In this case mark "Run-time Error" on the Run Sheet and return the attempt. This counts as an incorrect attempt.

11. If the contestant's solution did not suffer a runtime error the judging program will now compare the output of the contestant's program to the expected output using a Windows program named CSDiff.
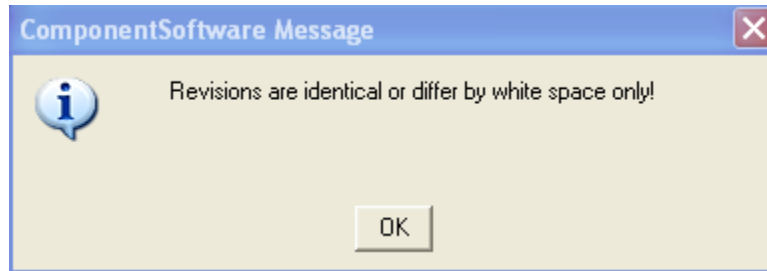


```
C:\ Judging -dryrun- for -team1-
Please enter the team number (i.e., 12): 1
Please insert submitted floppy into drive...
Press any key to continue . . .
        1 file(s) copied.
        1 file(s) copied.
Remove the floppy from the drive...
Press any key to continue . . .
Compiling program...
Press any key to continue . . .
Running program...
        1 file(s) copied.
Press any key to continue . . .
Diffing output...
```

If the contestant's output matches the expected output you will see the following message box:

This indicates the contestant's solution is correct! No inspection of the contestant's output is required by the judge. Mark accept on the Run Sheet and the contestants get credit for this problem.

You must click on the OK button of this window to go back to the judging program.
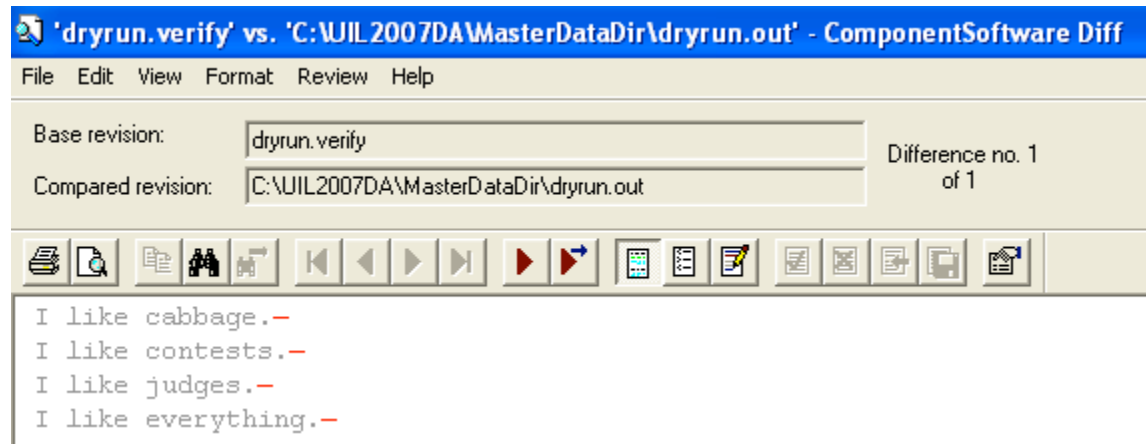
12. The judging program has now completed and you can press any key to end the program. You will now be ready to type in the command to judge the next problem.

13. If there are disparities between actual and expected output, CSDiff will instead display a window which highlights the differences between the output produced by the contestant program and the expected output. This display can be confusing, so I urge all judges to experiment with the interface before the day of the contest.

    The judge will have to make the final determination for the submission. Here are the basic guidelines for judges:

    a. Whitespace differences at the end of
    b. lines or after the last line of output are never significant.

    c. If the differences do not seem material to the problem solved, err on the side of accepting the solution. For instance, if a problem is about performing complex calculations, be flexible with output formatting. On the other hand, if the problem is all about formatting, then be a stickler.

    d. Above all, be consistent in your judging. With this goal in mind, it is usually best for each problem to have a designated judge or judging team to help ensure judging consistency for that problem.
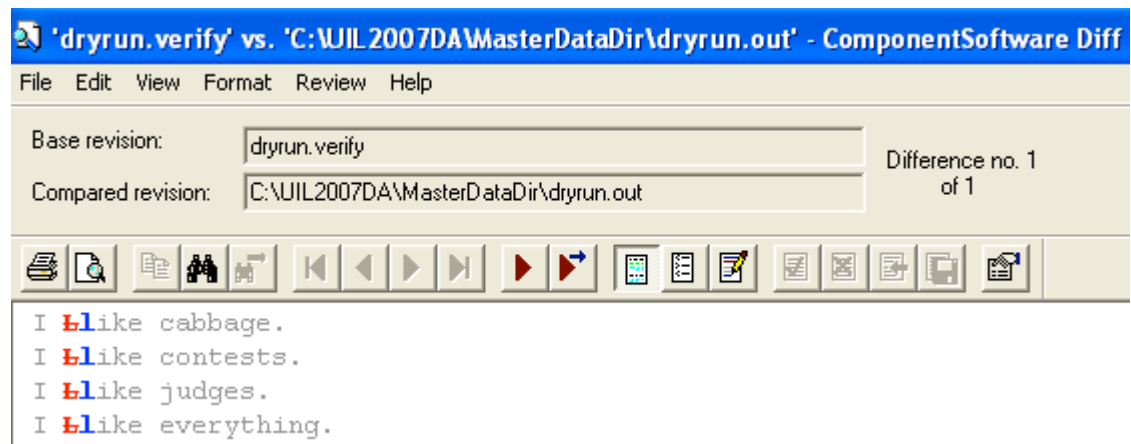
14. Here are some examples of differences between contestant and expected output.

<u>In this example of the dry run problem there is an extra space at the end of each line.</u>

The red line at the end of each line indicates a difference in the contestant's output. This would be an example of "Whitespace differences at the end of the lines and so they are not significant". **The contestant's solution is considered correct!**

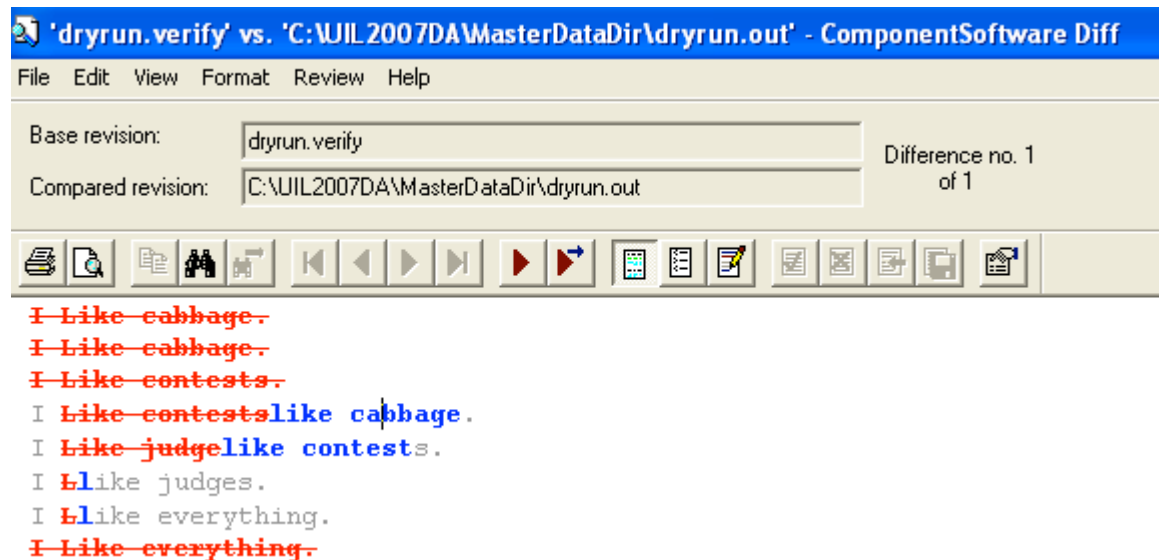<u>In this example of the dry run problem the solution capitalizes the word "Like".</u>

The red indicates the contestant's solution has a capital L while the blue indicates the expected output is a lowercase l. This would be an example of "If the differences do not seem material to the problem solved, err on the side of accepting the solution." **The contestant's solution is considered correct!**

In this example of the dry run problem the solution prints the required output twice instead of once.

This is a significant error and the contestant's solution would be rejected.

15. To reiterate, when judging the contest keep in mind "Above all, be consistent in your judging. With this goal in mind, it is usually best for each problem to have a designated judge or judging team to help ensure judging consistency for that problem."

16. After the contest the judging program and environment may be removed by simply deleting the entire `C:\UILProgramming` folder from the judging computers.

17. The dry run problem and the various versions of the dryrun.java program I used in these example are on the following pages.

Problem #0: Dry Run

Program Name: DryRun.java   Input File: dryrun.dat

Write a program that reads a list of items from the input file and outputs a message for each.

Input
The first line contains an integer, *n*, that indicates how many items are in the input file. The next *n* lines contain a single word. Each word represents an item that you like.

Output
For each item in the input, output a line stating, "`I like <item>.`". For example, if the item were cabbage, the program would output the line, "`I like cabbage.`"

Example Input File
```
4
cabbage
contests
judges
everything
```

Example Output To Screen
```
I like cabbage.
I like contests.
I like judges.
I like everything.
```

Correct solution to the dryrun problem. This code would be in a file named dryrun.java.

```java
import java.util.*;
import java.io.*;

public class dryrun{
     public static void main(String[] args) throws IOException{
          Scanner s = new Scanner( new File("dryrun.dat") );
          int num = s.nextInt();
          s.nextLine();
          String like;
          for(int i = 0; i < num; i++){
               like = s.nextLine();
               System.out.println("I like " + like + ".");
          }
     }
}
```

Attempted solution to the dryrun problem that contains a syntax error. The Scanner is not created correctly.

```java
import java.util.*;
import java.io.*;

public class dryrun{
     public static void main(String[] args) throws IOException{

          // the following line has a syntax error
          Scanner s = new Scanner( dryrun.dat );
          int num = s.nextInt();
          s.nextLine();
          String like;
          for(int i = 0; i < num; i++){
               like = s.nextLine();
               System.out.println("I like " + like + ".");
          }
     }
}
```

Attempted solution to the dryrun problem that results in a runtime error. The Scanner directed to a path hard coded off the C drive instead of being in the current directory.

```java
import java.util.*;
import java.io.*;

public class dryrun{
    public static void main(String[] args) throws IOException{
        Scanner s = new Scanner( new File("C:\\dryrun.dat" ));
        int num = s.nextInt();
        s.nextLine();
        String like;
        for(int i = 0; i < num; i++){
            like = s.nextLine();
            System.out.println("I like " + like + ".");
        }
    }
}
```

Attempted solution to the dryrun problem with an extra space at the end of lines. This solution would be considered correct.

```java
import java.util.*;
import java.io.*;

public class dryrun{
    public static void main(String[] args) throws IOException{
        Scanner s = new Scanner( new File("dryrun.dat") );
        int num = s.nextInt();
        s.nextLine();
        String like;
        for(int i = 0; i < num; i++){
            like = s.nextLine();

            //extra space output at end of line
            System.out.println("I like " + like + ". ");
        }
    }
}
```

Attempted solution to the dryrun problem with the word like capitalized. This solution would be considered correct.

```java
import java.util.*;
import java.io.*;

public class dryrun{
    public static void main(String[] args) throws IOException{
        Scanner s = new Scanner( new File("dryrun.dat") );
        int num = s.nextInt();
        s.nextLine();
        String like;
        for(int i = 0; i < num; i++){
            like = s.nextLine();

            //like is capitalized
            System.out.println("I Like " + like + ".");
        }
    }
}
```

Attempted solution to the dryrun problem with two lines of output instead of 1. This solution would be considered incorrect.

```java
import java.util.*;
import java.io.*;

public class dryrun{
    public static void main(String[] args) throws IOException{
        Scanner s = new Scanner( new File("dryrun.dat") );
        int num = s.nextInt();
        s.nextLine();
        String like;
        for(int i = 0; i < num; i++){
            like = s.nextLine();

            //two lines of output
            System.out.println("I like " + like + ".");
            System.out.println("I like " + like + ".");
        }
    }
}
```