Homework Six

A201/A597/I210—Spring Semester 2005

Due in OnCourse by Friday, March 11, 11:59pm
(*No written report is due in class, but read below for details!*)

Abstract

Write programs that solve 5 of the problems below as described. It's up to you what 5 problems you choose but one of them must be Problem 3.9 (that is, the `Copy` problem). One of the problems (Problem 3.1) is solved already, so you can't choose that one. Also included in the set are 4 (four) bonus problems. Thus the total presented below is of 10 problems.

Programs should be submitted to OnCourse in the dropbox for Homework Six. One or two paragraphs must summarize your program at the top of your source code file (see example posted, reproduced below). You're encouraged to work in groups and discuss the problems but you need to write the programs and prepare the write-up all by yourself.

The Computer Science Department[1] and the School of Informatics[2] clearly specify the rules of academic honesty and academic integrity, so please read the documents and make sure you understand them and comply with them.

Posting solutions or major hints on the bulletin board is not allowed.

1 The Summary

This homework is about using everything that you learned so far (loops, tokenizers etc.) to read data from a file and process it. In the case of Problem 3.9 you also need to open a file and write data into it. All files we deal with are text files. In Java we could read and write many types of files (serialized data, XML, etc) but we only focus on text files in A201. Problems listed below are at the level of difficulty of Homework Five and they would look very similar if instead of a file they would be reading from the console, a sequence of lines terminated with `done`, as we have done in the past. In that context we would simulate the end of file (EOF) by typing its value: `Control-D` (press the `Ctrl` key and while you keep it pressed also press the key for the letter `D`). To test for end of file we would be asking in our program if the line that was read was empty (`null`).

[1] `http://www.cs.indiana.edu/Academics/integrity.html`
[2] `http://www.informatics.indiana.edu/courses/honesty.asp`

2 What, No Written Report?!

Starting with this assignment you will not have to turn in a written report in lecture any longer[3]. Instead you will have to describe the program(s) by writing Javadoc comments in the program, the way I did it below. Please follow the template of Solved Problem 3.1 when turning in the programs, with Javadoc comments about your program design, as appropriately.

3 The Problems

As indicated earlier, the first problem is shown with a solution. We hope you find this helpful. Please note the large Javadoc comments[4] which from now on will replace the written reports we used to turn in lecture.

3.1 Count the Number of Lines in a File (Solved)

Write a program that reads a file and reports the number of lines in the file. The program receives the name of the input file on the command line and prints the report back on the screen. Here's how my program works:

```
frilled.cs.indiana.edu%cat one.txt
this file
      is a test
                  for
    my program
it contains
    a                       quote
by soren kirkegaard
which
    goes like this:
       Life   can only be
     understood
              backwards;
                    but
it
   must be lived
  forwards.
frilled.cs.indiana.edu%java One
Exception in thread "main"
        java.lang.ArrayIndexOutOfBoundsException: 0
        at One.main(One.java:23)
frilled.cs.indiana.edu%java One one
Something went wrong:
        java.io.FileNotFoundException: one (No such file or directory)
There are 0 lines in file one
frilled.cs.indiana.edu%java One one.txt
There are 16 lines in file one.txt
frilled.cs.indiana.edu%
```

[3] This, of course, comes as a great relief to many of us

[4] This code is also posted on-line (in color)

Here's the code of the program, with comments as a model, for you.

```
import java.io.*;

/**
 * In this program I solve the problem of counting the number of lines
 * in a file. This is Problem One in Homework Six (due Fri 3/11 @11:59pm)
 *
 * @author Adrian German
 *
 */

public class One {

   /** The main method is only two lines long because a procedure (method)
    * is used to count the lines in a file. The name of the file is passed
    * on the command line and is available to the main method as the first
    * of the command line arguments, args[0]. args[0] is passed to the
    * One.countLines(String) method and the int returned by the method is
    * placed in the count variable (which is local to main). All of this is
    * described on one line. The next line prints count and the program ends.
    */

    public static void main(String[] args) {
        int count = One.countLines(args[0]);
        System.out.println("There are " + count + " lines in file " + args[0]);
    }

    /** The countLines method receives one argument, called name, of type String.
     * It sets a local variable, count, to zero, It then tries to create a new
     * FileInputStream, called cable, using the name of the file on the disk. On
     * top of the cable we put receiver, which is an InputStreamReader, and on top
     * of that one we create the handset (a BufferedReader) which will allow us to
     * read lines from the file with the name "name". So we start by reading the
     * first line from the handset and then we enter a loop: while the line is not
     * the end of file, increment the count by one, and read a new line, repeat.
     * If the try fails an error message is printed. Otherwise the file input stream
     * is closed. The program returns count which is the number of lines read from
     * the file. A return value of zero doesn't necessarily mean there was an error
     * since the file could have been empty.
     */

    public static int countLines(String name) {
        int count = 0;
        try {
            FileInputStream cable = new FileInputStream(name);
            InputStreamReader receiver = new InputStreamReader(cable);
            BufferedReader handset = new BufferedReader(receiver);
            String line = handset.readLine();
            while (line != null) {
```

```
                count = count + 1;
                // System.out.println("(" + line + ")");
                line = handset.readLine();
            }
            cable.close();
        } catch (Exception e) {
            System.out.println("Something went wrong: " + e.toString());
        }
        return count;
    }
}
```

3.2 Number of Words in a File

Write a program that reads a file and reports the number of words[5] in the file. The program receives the name of the input file on the command line and prints the report back to the screen.

```
frilled.cs.indiana.edu%java Two one.txt
The total number of tokens in the file is: 31
frilled.cs.indiana.edu%
```

3.3 Average Number of Words per Line

Write a program that reads a file and reports the average number of words[6] on a line, in the file. The program receives the name of the input file on the command line and prints the report back to the screen.

```
frilled.cs.indiana.edu%java Three one.txt
This file has 31/16 = 1.9375 words per line.
frilled.cs.indiana.edu%
```

3.4 Average Line Length (in Characters)

Write a program that reads a file and reports the average number of characters[7] on a line, in the file. The program receives the name of the input file on the command line and prints the report back to the screen.

```
frilled.cs.indiana.edu%java Four one.txt
This file has 136/16 = 8.5 characters per line.
frilled.cs.indiana.edu%
```

[5] We're interested in *tokens*, actually

[6] We're actually interested in *tokens*, again

[7] For the purpose of this exercise blank characters do not count.

3.5 Average Word Length (per File)

Write a program that reads a file and reports the average number of characters per word (that is, token) in the file. The program receives the name of the input file on the command line and prints the report back to the screen. The result is one number.

```
frilled.cs.indiana.edu%java Five one.txt
This file has 136/31 = 4.387096774193548 characters per word.
frilled.cs.indiana.edu%
```

3.6 Average Word Length (per Line)

Write a program that reads a file and reports the average number of characters per token per line. The program gets the name of the input file on the command line and prints the report back to the screen. The result is a sequence of average word lengths, one for each line in the file.

```
frilled.cs.indiana.edu%java Six one.txt
Line 1 has 8/2 = 4.0 characters per token, on average.
Line 2 has 7/3 = 2.3333333333333335 characters per token, on average.
Line 3 has 3/1 = 3.0 characters per token, on average.
Line 4 has 9/2 = 4.5 characters per token, on average.
Line 5 has 10/2 = 5.0 characters per token, on average.
Line 6 has 6/2 = 3.0 characters per token, on average.
Line 7 has 17/3 = 5.666666666666667 characters per token, on average.
Line 8 has 5/1 = 5.0 characters per token, on average.
Line 9 has 13/3 = 4.333333333333333 characters per token, on average.
Line 10 has 13/4 = 3.25 characters per token, on average.
Line 11 has 10/1 = 10.0 characters per token, on average.
Line 12 has 10/1 = 10.0 characters per token, on average.
Line 13 has 3/1 = 3.0 characters per token, on average.
Line 14 has 2/1 = 2.0 characters per token, on average.
Line 15 has 11/3 = 3.6666666666666665 characters per token, on average.
Line 16 has 9/1 = 9.0 characters per token, on average.
frilled.cs.indiana.edu%
```

3.7 Print the Longest Line in File

Write a program that reads a file and reports the longest line (the line with the most characters[8] in it). The program gets the name of the input file on the command line and prints the report back to the screen. The result is a line of text, surrounded by parentheses.

```
frilled.cs.indiana.edu%java Seven one.txt
The longest line in file one.txt is:
(this file                              )
frilled.cs.indiana.edu%
```

As you can realize, there were spaces in my file.

[8] Spaces are counted, this time.

3.8 Print the Line with Most Words

Write a program that reads a file and reports the line with the most tokens in it. Obviously, the blank spaces can't have an influence here (unlike in the previous problem). The program gets the name of the input file on the command line and prints the report back to the screen. The result is a line of text, surrounded by parentheses.

```
frilled.cs.indiana.edu%java Eight one.txt
The line with most words in file one.txt is:
(       Life   can only be )
frilled.cs.indiana.edu%
```

3.9 Make a Copy of a File

Write a program that makes a copy of the file. The program receives two arguments on the command line: the first one is the source file (to be copied) and the second one is the target file (the newly created copy).

```
frilled.cs.indiana.edu%ls -ld one.txt two.txt Nine.java
ls: two.txt: No such file or directory
-rw-------  1 dgerman faculty 744 Mar  5 23:46 Nine.java
-rw-------  1 dgerman faculty 319 Mar  5 17:41 one.txt
frilled.cs.indiana.edu%javac Nine.java
frilled.cs.indiana.edu%java Nine one.txt two.txt
frilled.cs.indiana.edu%ls -ld one.txt two.txt Nine.java
-rw-------  1 dgerman faculty 744 Mar  5 23:46 Nine.java
-rw-------  1 dgerman faculty 319 Mar  5 17:41 one.txt
-rw-------  1 dgerman faculty 319 Mar  5 23:47 two.txt
frilled.cs.indiana.edu%cat one.txt two.txt
this file
      is a test
                  for
    my program
it contains
    a                     quote
by soren kirkegaard
which
    goes like this:
       Life   can only be
     understood
              backwards;
                    but
it
   must be lived
  forwards.
this file
      is a test
                  for
    my program
it contains
```

```
    a                     quote
by soren kirkegaard
which
    goes like this:
       Life   can only be
     understood
              backwards;
                    but
it
   must be lived
  forwards.
frilled.cs.indiana.edu%
```

3.10 Search and Replace

Write a program that reads a file and two words and writes back the contents of the while with all the occurrences of the first word replaced by the second. The file name and the two words are to be specified on the command line, as `args[0]`, `args[1]` and `args[2]` respectively.

```
frilled.cs.indiana.edu%java Ten one.txt kirkegaard kierkegaard
this file
is a test
for
my program
it contains
a quote
by soren kierkegaard
which
goes like this:
Life can only be
understood
backwards;
but
it
must be lived
forwards.
frilled.cs.indiana.edu%
```

4 The Datafile Used

I used one and the same datafile (`one.txt`) throughout[9].

[9] Perhaps the single most memorable feature of its contents is that I misspelled Kierkegaard.