CS 351 – DATA ORGANIZATION AND MANAGEMENT

FALL 2010 - Programming Project #1

(Modified Version of November 4)

Intersection of Sequential Files

Due: November 12, 2010 at 11:59 PM

1. The Objective

In this programming assignment you are asked to implement a JAVA console application which simulates finding the intersection of two piles (*F1* and *F2*) and writing the common records to an output text file (*F12*). The aim of the assignment is to make you understand how to handle this intersection procedure efficiently. You will also increase your coding skills regarding to how to manage r/w files in JAVA. **Please read the entire assignment before starting implementation.**

2. File Structures

Student ID	Student Name	Department

The format of a record in both input and output files are given in the above figure. The first column of a record includes student's id number, and the following columns are student's full name and his/her department code.

Consider the following remarks for file structures:

- There is no duplicate of a record in input files. A full record can be read into a String (i.e you do not have to know the size of a record in this assignment).
- You have to read/write a file like in Section 5 of this assignment.
- Examples of *F1*, *F2* and *F12* can be found in: http://cs.bilkent.edu.tr/~ctoraman/cs351fall10/project1/samples.rar

3. What to do?

You have to complete 4 parts in this assignment:

Part A: In this part, you follow these steps:

- Open the first input file *F1 and* the output file *F12* for writing.
- Search records of *F1* in *F2*:
    - Read a record *R1* from *F1*.
    - Search *R1* in *F2:*
        - Open the second input file *F2*
        - Search the second input file sequentially for *R1*.
        - If *R1* is found in *F2*, then write *R1* to *F12*.
        - Close F2.
    - Repeat this procedure until you process all records of *F1*.
- Close *F1* and *F12*.

Part B: In this part, you follow these steps:

- Open the first input file *F1* and the output file *F12* for writing.
- Suppose you have enough memory size to read a block of *n* records in one time:
    - Read a block *B1* of *n* records from *F1*.
        - Open the second input file *F2*.

- Read a record *R2* from *F2*.
- Search *R2* in *B1:*
  - Search B1 sequentially for *R2*.
  - If *R2* is found in *B1*, then write *R2* to *F12*.
- Repeat this for all records in F2.
- Close *F2*.

◦ Repeat this procedure until you process all blocks of *F1*.

◦ Close *F1* and *F12.*

Part C: This part is similar to Part B. Repeat the same procedure with Part B. However consider the following conditions as well:

- F1 is given as sorted according to Student ID field of records.
- When you search *R2* in *B1*, use Binary Search algorithm on Student ID field of records.

Part D: After you successfully implement Part A, B and C; then you will execute your implementations by following these steps:

- Execute Part A with the example files(**big samples**) given in Section 2 and record the execution time (explained in Section 6). Then execute the same input files in Part B with n = 100,000 and record execution times. Finally, execute the same inputs in Part C with n = 100,000. You have 3 cases to compare in total.
- Prepare one-page report that answers the following questions:

- Write execution times for Part A, B and C in terms of **ms (milliseconds)**. Order the execution times.

- Which case did perform the best? Did you expect these results? Explain.

- Which case did perform the worst? Did you expect these results? Explain.

4. General Rules:

- Your applications for part A, B and C must be named **FileIntersectionPartA**, **FileIntersectionPartB, FileIntersectionPartC** respectively.

- You are free how to test your implementation. We will use our own main method to grade your projects. Therefore, do not put any essential code segments into your main method(s). Also we will call only the constructors(of **FileIntersectionPartA**, **FileIntersectionPartB** and **FileIntersectionPartC**) while testing. FileIntersectionA gets no parameter whereas B and C need an integer variable representing n records to read each time.

- The input files will always be **stdlist1.txt** and **stdlist2.txt** for **Part A and B.** Their names will be **stdlist1sorted.txt** and **stdlist2.txt** for **Part C.** The output file will always be **outputA.txt, outputB.txt, outputC.txt** for **Part A, B and C** respectively**.** All the files must be placed in the same folder that your JAVA project folder exists.

- Your report must be named **StudentID.doc** (e.g 20504152.doc)

- Note that we will test your programs with another input files than the example files given in Section 2. Also a supportive tool that is able to detect plagiarism will be used.

5. Tutorial: How to R/W File in JAVA

In JAVA, there are various ways to handle r/w a file; but you must use the following way. Other implementations will **not** be accepted.

You have to use BufferedReader and BufferedWriter classes to read and write a file respectively. Consider using the following methods of BufferedReader (You can also see JAVA API for all methods):

- *constructor*: **BufferedReader(Reader in):** Opens the file to read. The argument is an instance of Reader class. See JAVA API for more details.
- **readLine()**: Reads a line from the file.
- **close()**: Closes the file.

Also use the following methods of BufferedWriter:

- *constructor*: **BufferedWriter(Writer out):** Opens the file to write. The argument is an instance of Writer class. See JAVA API for more details.
- **write(String str):** Writes a string to the file.
- **close():** Closes the file.

6. How to Time Your Execution?

Finding your execution time is easy in JAVA. The following code segment displays a sample execution of PartB. *time* variable calculates the execution time in **ms**. startTimer() and endTimer() are just before the program execution and just after the program execution.

```
long time;
reset();
startTimer();
intersectionSecondMethod(1);
endTimer();
time = duration();
```

The following code segment calculates execution time by finding the difference between two different time points.

```
static long startTime, endTime;


public static void startTimer(){
    startTime = System.currentTimeMillis();
}

public static void endTimer(){
    endTime = System.currentTimeMillis();
}

public static long duration(){
    return endTime - startTime;
}


public static void reset(){
    startTime=0;
    endTime=0;
}
```

7. Submission

I. Put all your files (**FileIntersectionPartA.java, FileIntersectionPartB.java, FileIntersectionPartC.java, StudentID.doc**) into a **rar** file. Do **not** put any other file.

II. Name your rar file as **StudentID_cs351p1.rar** (e.g 20504152_cs351p1.rar). Files with other formats will be **ignored** automatically.

III. Use the submission page (http://cs.bilkent.edu.tr/~ctoraman/cs351fall10/project1/submission/) to upload your rar file. Read the upload rules in the submission page as well. Late submissions will **not** be accepted!

IV. You can resubmit until the due date and the previous file will be replaced.

V. For questions, contact with ctoraman@cs.bilkent.edu.tr and hayrettin@cs.bilkent.edu.tr