# CS351 DATA ORGANIZATION AND MANAGEMENT

## Programming Assignment #1

Static Hashing in JAVA

**October 13, 2009**
**Due Date: Oct. 23, 2009, Friday; by 23:59**

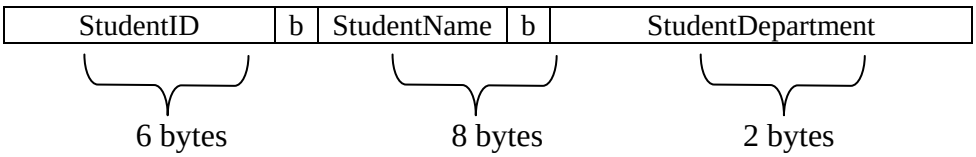
**Assignment:**

The purpose of this assignment is to give you a practical exercise with static hashing. For the sake of simplicity, you will implement record insertion from an input file **"Students.txt"**, not deletions or modifications. You will use "RandomAccessFile" class of Java which will help you during insertion of records to the output file.

As you know, in static hashing, there are two areas which contain records: "Prime" and "Overflow" areas. Each area will have specified number of buckets. Your program will ask user both the prime *area bucket number* and *overflow area bucket number*. For ex., when user enters 10 for prime area bucket number and 3 for prime area bucket number; then there will be 10 buckets in prime area and 3 buckets in overflow area. But you will choose **Bkfr** (Bucket Factor) as **1**, which means there will be only one record in each bucket both in overflow area and prime area.

**Input File: "Students.txt"**

The structure of the input file is as follows. 3 blank separated fields exist in each line of the input file. (b indicates a blank character)

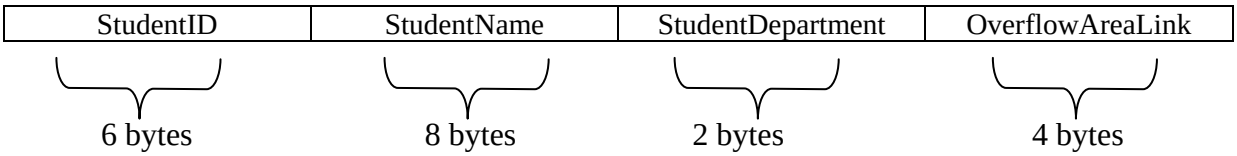



**Ex:** 204021 Emre CS
204123 Levent IE
203212 Dilek EE

The prime area bucket number taken from user will be used in our hash function, h(x), **x mod prime area bucket number** (x is the StudentID field of Students.txt). By using this hash function, you will identify the bucket in which that record will be inserted. Because of bucket factor is 1, if the bucket in the prime area has another record already inserted, then you will insert the record into another bucket in the overflow area. But you also have to check whether overflow area has any empty buckets, because overflow area will have specified number of buckets, not infinite. When inserting a record, *if the overflow area is full*, then you will give an *error message* about insertion and you will skip to insertion of other records in the input file.

**Output File: "HashFile.txt"**

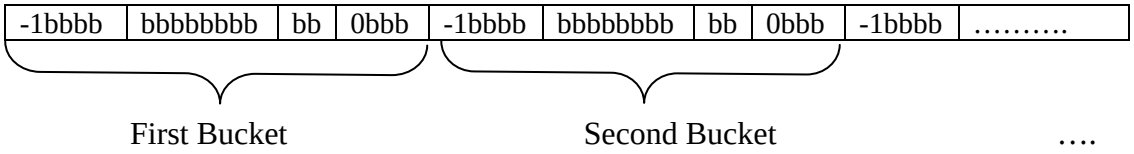
Your program will create an output file, "HashFile.txt" which will both prime area buckets and overflow area buckets. First of all, the output file will contain only **1 line**, which means it will be a *sequential file*: each bucket will come one after the other. This will ease your seek operation during insertions. Then, *each bucket* will cover **20 bytes** in the output file:




But before the insertions of records, you have to initialize this output file as follows:

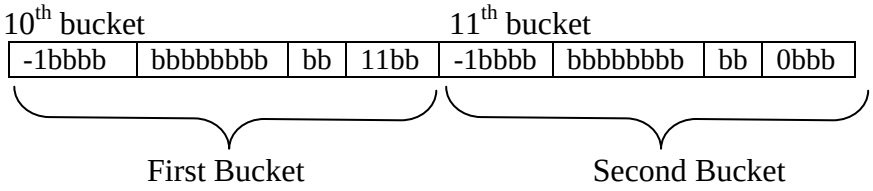
**For Prime Area Buckets:**

You will write **"-1"** into *StudentID* field of each bucket which will indicate that bucket is empty. And you will write "0" into OverflowAreaLink field of each bucket in prime area which will indicate that bucket has no link to another bucket in the overflow area. StudentName and StudentDepartment fields will remain empty. After initialization, your output file has to look like:




**For Overflow Area Buckets:**

You will write **"-1"** into *StudentID* field of each bucket which will indicate that bucket is empty. But OverflowAreaLink filed will be different than prime area buckets. There will be a linked-list between buckets in overflow area, which means that each bucket will have a link to following bucket in the overflow area except the last bucket in the overflow area. StudentName and StudentDepartment fields will remain empty. After initialization, your output file has to look like (if prime are bucket no: 10 and overflow area bucket no: 2; the first bucket be $0^{th}$ bucket):




*****Each **"b"** indicates **one blank character.**

*The first bucket of overflow area will come just after the last bucket of the prime area in the output file.

**Overflow Pointer:**

In this assignment, you will have an overflow pointer, which will point the beginning of the overflow area initially. When you insert any record into a bucket in the overflow area, this overflow pointer will point the first empty bucket in the overflow area. At the end of the program, you will write the value of this pointer in another output file **"Overflow.txt"**.

*This assignment does not include any deletions, but the second assignment will be about deletions, so this overflow pointer will be actually useful in the second assignment.

**Implementation:**

You will have only one java class, **"StaticHashing.java"** which will make these operations in order:

- ask user the prime area bucket number and overflow area bucket number
- initialize the output file as explained previously
- insert each record in "Students.txt" into "HashFile.txt" using hash function
- write the overflow pointer value into "Overflow.txt"

You will use **"RandomAccessFile"** class of Java in this assignment for read and write operations. So you will create the random access file in read/write mode ("rw"). There are a few types of read and write methods in this class. But, if you use **"writeBytes()"** method to write records into buckets, you can see the inserted records in the output file during your tests. For read operations, you can use **"readFully()"** method.

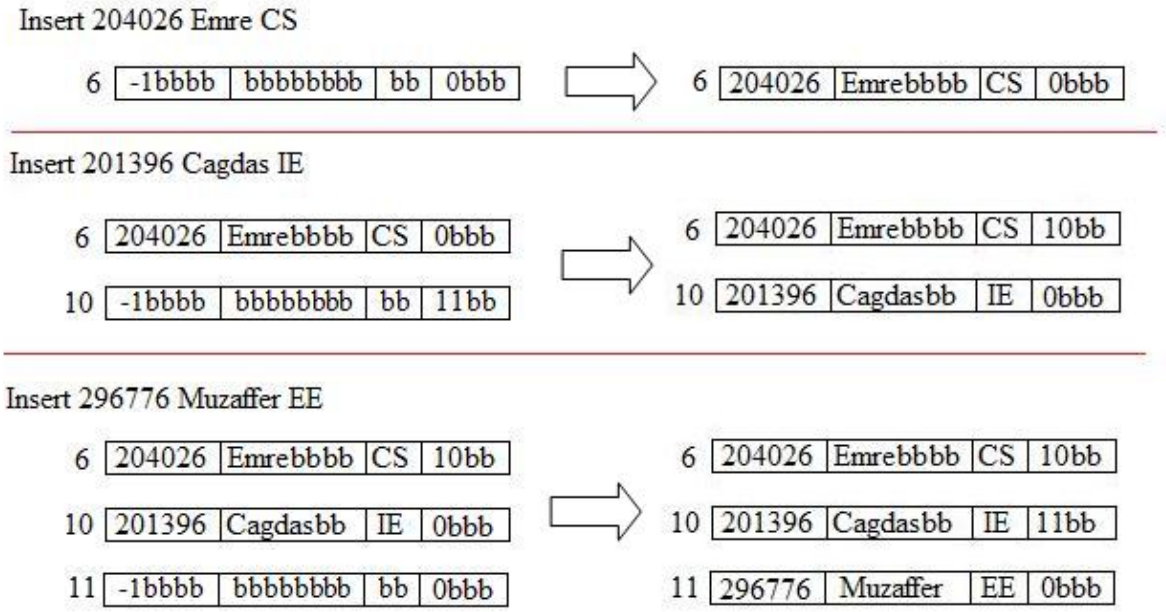
There is a kind of cursor called the ***file pointer*** of RandomAccessFile class. Input operations read bytes starting at the file pointer and advance the file pointer past the bytes read. If the random access file is created in read/write mode, then output operations are also available; output operations write bytes starting at the file pointer and advance the file pointer past the bytes written. The file pointer can be read by the **getFilePointer**() method and set by the **seek()** method.

You will use **"seek()"** method of RandomAccessFile to write the record into the bucket defined by hash function. For ex. We have a studentid 204029, and user entered 10 for prime area bucket number. Then hash function will return **9** as the result of 204029 mod 10. You will go to the $9^{th}$ bucket by using **seek(9*20)** because each bucket will be 20 bytes in this assignment. Then you will write your record in this bucket.

When you insert a record into an empty bucket, you will not change the OverflowAreaLink field of the bucket in the prime area which is initialized as "0". But when a bucket in the prime area is filled, you will insert that record into the first empty bucket in the overflow area which is pointed by file pointer. **If the OverflowAreaLink field of the bucket in the prime area is "0"**, you will insert the record into the first empty bucket which is pointed by overflow pointer and you will update the OverflowAreaLink field of the bucket in the prime area with the bucket number of the bucket in the overflow area. So the bucket in the prime area will have a link to the bucket in the prime area which indicates but records in these buckets have the same result in the hash function. **But if OverflowAreaLink field of the bucket in the prime area is not "0"**, that's, it has a bucket number, then you will again insert the new record in the first empty bucket in the overflow area pointed by overflow pointer. But

this time you will update the OverflowAreaLink field of the last bucket with the same result in the hash function. You have to recursively go through the links written in OverflowAreaLink field of each bucket and find the last bucket in the list, which will have "0" in OverflowAreaLink field. And you will update this field by the bucket number of bucket where the new record is inserted.

Ex: Prime area no: 10
Overflow area no: 2
You will see the contents of the buckets before and after insertion:




When inserting 296776, hash function will give us 6. When we look at the bucket 6, it has a record already inserted, and it has "10" in the OverflowAreaLink field, which means there is a bucket which is linked with this bucket. Then, when we look at bucket 10, it has a record already inserted, too. But it has "0" in the OverflowAreaLink field, so this bucket is the last bucket in the linked list. Then we will insert the record with studentid 296776 into the first empty bucket in the overflow area, which is the bucket 11. After insertion, we update the OverflowAreaLink field of bucket 10 with "11". But notice that, at the initialization of output file, buckets in the overflow area has already linked to each other. But during the insertion of 201396, we update the OverflowAreaLink field of bucket 10 with "0" to indicate that bucket is the last bucket in the linked list.

* **Notice that each write and read operation advances the file pointer**, so especially after read operations, don't forget to reposition file pointer to the beginning of the bucket.
* You will read strings from input file, so for studentid field, don't forget to parse it into integer before using them in the hash function.
* There is one blank between each field in the input file. That's, each field is separated by one blank.
* When you update the OverflowAreaLink field of a bucket, you will write the bucket number of the bucket where the new record is inserted, not the address of that bucket. (For ex. you will write 11 into OverflowAreaLink field not 220(11*20), but when you travelling through buckets by using these links, don't forget to multiply it with 20, which is the size of each bucket.)

* writeByte(“Emre”) costs 4 bytes, one byte for each character and it advances file pointer 4 bytes. And also during the readFully(x) advances the file pointer past the bytes read.

**Testing:**

A sample input file (http://www.cs.bilkent.edu.tr/~canf/CS351Fall2009/Fall09HWs/Students.txt) is posted to the website to help you test your program. This is not the input file that will be used in grading your program. While the test data provided should be useful, you should also do testing on your own test data to ensure that your program works correctly.

**Submission:**

You will submit your “StaticHashing.java” java class as win.rar file. The name of the winrar file will be *StudentID_Name_Surname.rar* (20491000_Emre_Celik.rar) Details about upload page will be announced in the course web page.