The assignment should be done in groups of two students. You must turn in the source code of your program through Canvas. Each group must submit only one file that contains the full name, OSU email, and ONID of every member of the group.

1: Linear Hash Index (10 points)

The objective of this assignment is to learn how to implement index structures on data stored in the external memories.

Assume that we have a relation *Employee(id, name, bio, manager-id)*. The values of *id* and *manager-id* are integers each with the fixed sizes of 8 bytes. The values of *name* and *bio* are character strings and take at most 200 and 500 bytes, respectively. Note that as opposed to the values of *id* and *manager-id*, the sizes of the values of *name* and *bio* are not fixed and are between 1 to 200 (500) bytes. The size of each page is 4096 bytes (4KB). The size of each record is less than the size of a page. Using the provided skeleton code with this assignment, write a C or C++ program that creates a **hash index file** for relation *Employee* using attribute *id*. Your program must also enable users to search the created index by providing the *id* of a record.

- **The Input File:** The input relation is stored in a CSV file, i.e., each tuple is in a separate line and fields of each record are separated by commas. Your program must assume that the input CSV file is in the current working directory, i.e., the one from which your program is running, and its name is *Employee.csv*. We have included an input CSV file with this assignment as a sample test case for your program. Your program must create and search hash indexes correctly for other CSV files with the same fields as the sample file.
- **Index File Creation:** Your program must read the input Employee relation and build a **linear hash index** for the relation using attribute *id*. Your program must store the hash index in a file with the name *EmployeeIndex* on the current working directory. You may use one of the methods explained for storing variable-length records and the method described on storing pages of variable-length records in our lectures on storage management to store records and blocks in the index file. They are also explained in Sections 9.7.2 and 9.6.2 of Cow Book, respectively. You can reuse your code for Assignment 2 to store pages in the index file. **Your index file must be a binary data file and not a text or CSV file.**
- **Index Parameters:** You must use hash function $h = id mod 216$. Your program must increment the value of n if the average number of records per each page exceeds 70% of the page capacity.
- **Main Memory Limitation:** During the index creation, your program can keep up to three pages plus the directory of the hash index in main memory at any time. The submitted solutions that use more main memory will *not* get any points.
- **Searching the Index File:** After finishing the index creation, your program must accept an Employee *id* in its command line and search the index file for all records of the given id. Like index creation, your program may use up to three pages plus the directory of the hash index in main memory at any time. The submitted solutions that use more main memory will not get any points for implementing lookup operation. The user of your program may search for records of multiple ids, one id at a time.

- Each student has an account on hadoop-master.engr.oregonstate.edu server, which is a Linux machine. You must ensure that your program can be compiled and run on this machine. You can use the following bash command to connect to it:

  ```
  > ssh your_onid_username@hadoop-master.engr.oregonstate.edu
  ```

  Then it asks for your ONID password and probably one another question. To access this server, you must be on campus or connected to the Oregon State VPN.

- You can use the following commands to compile and run C++ code:

  ```
  > g++ -std=c++11 main.cpp -o main.out
  ```

  ```
  > main.out
  ```

- **Skeleton Code:** We have included the files that contain the skeleton code to generate the required *EmployeeIndex* file. You can make changes to these files adhering to the assignment requirements.

- **Grading Criteria:** The programs that implement the correct algorithm, return correct answers, and do not use more than allowed buffers will get the perfect score. The ones that use more buffer than allowed will not get any points. The ones that implement the right algorithm but return partially correct answers will get partial scores.