Overview:

The overall system is Python version 3.5.2 based with BerkeleyDB backed database usage. The overall program initially takes an .XML chosen by the user in which Phase 1 will take as an input and output 4 .TXT files each containing different types of data. These being dates, terms, emails, and recs. The program in phase 1 acts as a file parser. In Phase 2 the files will be taken and by using linux command lines in terminal the files will first be sorted in alphabetical order. Th en from there will be loading into a .IDX file by using BerkeleyDB function DB_LOAD. This takes the data and loads into the .IDX where it will contain a row id allowing for easy retrieval. In phase 3, the program takes all the work performed by phase 2 and now uses the .IDX and creates a .DB file which will be used in order to query the data pending the user input which will determine the query necessary to perform the task desired by the user.

User Guide:

Phase 1:

In Phase 1, we wrote a program that reads records in XML from the standard input and produces 4 files as described next. The format of the input is as given in the samples with the first, second and the last input lines giving the XML tags and every other line giving a record. terms.txt: This file includes terms extracted from email subject and body; for our purpose, suppose a term is a consecutive sequence of alphanumeric, Emails.txt: This file includes one line for every email address that appears in a from, to, cc and bcc field. dates.txt: This file includes one line for each email record in the form of *d:l* where d is the date of the email, and *l* is the row id. recs.txt: This file includes one line for each email in the form of l:*rec* where l is the row id and rec is the full email record in XML.

Phase 2:

In order to use Phase 2, you need to have 4 files that are .txt and have the following names; terms.txt, dates.txt, recs.txt, and email.txt. Once having all four files in the correct format then the user must open the Phase 2 commands file where they must open a terminal window in the same folder that the 4 files are located. The copy paste the four sort functions into the terminal window. The functions will run then 4 new files will appear in the folder. These 4 files are the sorted versions of the files that were originally created. Once having the new sorted files named; terms-sorted.txt, dates-sorted.txt, recs-sorted.txt, and email-sorted.txt. The user must run the next set of unix commands located in the Phase 2 Commands.txt file. These commands will now take the 4 sorted files and create a hash index on recs.txt with row ids as keys and the full email record as data, a B+-tree index on terms.txt with terms such as keys and row ids as data, a B+-tree index on emails.txt with emails as keys and row ids as data, and a B+-tree index on dates.txt with dates as keys and row ids as data. The final output will be 4 index files named, re.idxm da.idx, te.idx, and em.idx which will be used in phase 3 in order to create database files.

Phase 3:

For the use of Phase 3, the user must have created 4 .idx with the names re.idx, te.idx, da.idx, and em.idx. These files are loaded into the program where the .db files are made. Once initiating the program the user is prompted to use decide to insert a query or quit. The user makes their choice and then must decide whether they want to either view the full or brief layout of the information being returned. The full output includes the entire records while the brief only returns the ROWID and the SUBJ of correspondence. Then the user we be prompted to enter their query, where the must follow the basic rules assigned. After the user enters their query, multiple for loops begin deducing what the user input was, where they search based on keywords such as subj, body, date, bcc, cc, to, from. The loops will correct the form of the user input and make sure it is formatted correctly to be used in the query portion of the program. The query portion takes the formatted user input and designates the the lists to specific .db files created by phase 2. The queries do their task based on the user input and retrieve any data that satisfies their conditions required.

Brief Description of our Algorithm:

Our Algorithm for evaluating user queries mainly consists of for loops where we try to determine the user input and clean it up in order to be used later down the line and be processed by our query function. The query function run by determining what the output from the for loops was and then splits the data based on the semicolon and then we run the query to detect all possible matches based on the database and then those matches that are found go through some condition handling and in the end we have a final output which is printed to the user.

Testing Strategy:

The testing strategy for our team consisted of everyone testing each phase and confirming outputs were the same across the board. We also tested with errors in the text file being loaded in order to make sure an issue in an initial phase doesn't break a later phase. We confirmed the outputs in phase 2 with others on the discussion board to insure everything was working properly for the final phase. The final phase we….

<u>Issues / Bugs Encountered</u>:
Python Versions: Each member of the team was using a different version of python and once coming together with all our code and tried to run it, certain functions wouldn't work due to the python version not accounting for specific things.

- print("f……) does not work in python 3.5.2 but it does on python 3.7.2

Phase 2 DB_LOAD: The commands initially created were not entirely correct and results were being chopped off. So the revisting of the phase 2 commands took time away from spending on daunting task of phase 3. In the end the commands where cleaned up and made more understandable while using the break.pl script provided.