# CS6360.004 Fall 2019

# Assignment 5: Create a Single-Level Index

This program creates a simple index on a text file. It will read a text file containing data and build the index, treating the first n columns as the key, **where n will be specified by the user**.  Writing this program will require knowledge of binary I/O in the language of your choice.  The only allowable language is C/C++. You will be provided two data sets for testing, the same ones the TA will use. The first one (CS6360Asg5TestDataA.txt) should be indexed on the first 15 bytes. The second one (CS6360Asg5TestDataB.txt) should be indexed on the first 4 bytes.

If there are duplicates, which might be the case if you indexed the larger of the two test files on just one column, you should put them into your index with their correct record pointers.

This program must run from the command line, which takes four parameters:

1. The first parameter is one of two strings: The string –c indicates that you should create an index. The string –l (lower-case L) indicates you should list the file using the index.
2. The name of an input text file.
3. The name of the output file.
4. The number of characters, starting with the first one, that comprise the key.

For example, (call your program INDEX) the following command line would create an index on a file called DATA.TXT, produce an output file DATA.IDX, with a key length of 4:

INDEX -c DATA.TXT DATA.IDX 4

The output file must contain the key and the 8-byte **binary** offset of the record within the data file. Thus records in your output file are fixed-length, containing the number of bytes in the key plus 8 bytes of pointer, or offset. In the example, each index record would be 12 bytes long. The output file must be sorted. While a real system would use an external sort, you may create an array of objects and sort that in memory. Thus if your input file is this:

AAAATest data 1

ZZZZAnother test record

CCCCThird test record

Your output file would look like this, with the brackets meaning that what is in them is a binary number, not decimal digits. That is, there are no actual brackets in the file. Each record, in this case, is exactly 12 bytes long, with 4 for the characters of the key and 8 for a long integer.

AAAA[0]

CCCC[39]

ZZZZ[15]

Thus the first data file record is in position 0 in the input file, the second one with key CCCC is in position 39, and the third one with key ZZZZ, the second input record, is in position 15.

The second way to run this program, to list the contents of a file using the index, takes the following parameters:

1. The string –l (lower-case L) for “list”.
2. The name of the associated data file.
3. The name of an index file.
4. The length of the key.

This usage must list the records in order using the data from the index. Thus you will read a record from the index file, use its “pointer” to read a record from the data file, and display it. Since the data records are variable length, you should read in 1000 bytes, knowing where the record starts, and look for a newline character. Then display the record.

This command lists all of the records in alphabetical order:

INDEX -l DATA.TXT DATA.IDX 4

Given the data above, you would see:

AAAATest data 1

CCCCThird test record

ZZZZAnother test record

|  |  |
| --- | --- |
| **Grading Criteria** | |
| Program takes any number between 1 and 24 as the key length | 10 |
| Program correctly computes offsets of records | 10 |
| Program produces a correct output file with a mix of text and binary data | 30 |
| Program correctly reads the index and lists all of the records | 50 |

**Possible errors:**

The output file is text and editable with Notepad: -30

The output file is not sorted: -20

The program crashes for valid input: -30

Program ignores the key length from the user and hard-codes it. -30.

Duplicate records are not handled correctly. -10

Listing the file (the -l option) does not produce a sorted list of all records. -30