CSC 300 Spring 2014
Project 2 – Indexes (*PRE-WORK*)
0 Points
Due: N/A

File processing is an important operation that can be found in almost all programming environment. Files are normally stored in one of two basic formats (text or binary). Text files store everything as a character. The number 64 would be stored as the characters '6' and the character '4' (in reality their ASCII values would be stored, each using 1 byte). Binary files also store characters as their ASCII values but store numeric values using their binary representation. For example, the number 64 would be represented by the single byte [1000000]. A major advantage of the binary format is that the numeric data being read-in or written-out does not have to be translated back and forth to and from ASCII character format. It is also often the case that the binary version of the data will be much smaller than the text version. The one disadvantage is that users can not use text based utilities like word processors to view the data in the file.

Using the binary format also allows us to build file access methods that may be more efficient than the sequential access method required for text based files that must read files character-by character or line-by-line.

Random-Access Files

Working directly with data from secondary storage in an efficient manner is made easier through the use of **Random-Access File** processing. This method allows us to access items in a file much like we do in a single-dimensional array. The file will be logically looked at as being made up of homogeneous records of the same size (in bytes). We can then use a logical record number in the same way we use an index value in an array. By pre-calculating the starting location of a record in the file, we can directly access 1 record's worth of binary data. (***see random access handout***)

For this (pre-work) you/we will create a binary file of records that will be used in assignment #2 for implementing a random access file system. You will use a simple 'sequential' method to test the correctness of your file.

Program End-User requirements.

(1) This program will read the data to build the binary file from a text file called **data.txt**. The text file will contain lines of data delimited by commas, each representing data about an employee.

The format for each line will be: *field1,field2,field3<eoln>*

Field1 : the company ID # of the employee
Field2 : the name of the employee. Names are from 1 to 30 characters long. (can have spaces)
Field3 : the age of the employee

(*for testing purposes you can create this file using notepad etc..)*

(2) The program will create a binary file of records called **empl.dat**. Each record will hold the data about one employee and have the following format:

```
struct struct_name
{
    int   id_num;
    char  name[31];
    int   age;
}
```

Design Requirements

N/A

Documentation Requirements

N/A

Submission Guidelines

N/A

***Parsing** is a term used to describe the process of taking a string and splitting it into the different fields of data that are present as substrings within it. As part of the parsing, the fields may or may not also be converted from strings into the appropriate formats of the fields.*

Note: *I will eventually provide a testing version of **data.txt** for you so you can make sure your program works with the correct format. However, you should create your own test file to get started.*