CSE 102 **Programming Assignment**
W.A. Burkhard **Due: February 5, 1999**

Linear Hashing Index Maintenance

We are to create an author index using linear hashing for data in the "Current Index to Statistics Extended Database" (CIS) which is a licensed database. The database is line oriented making it rather easy to peruse using a text editor; there is one record per line. Each record contains one bibliographic citation. The lines do not have the same length.

The syntax for these records is presented in section 3 of the `USRGUIDE.TXT` file located within the course `public/Data` subdirectory. The third field usually will contain the author names associated with the article. The names will be separated by semicolons. Each line begins with a `#` mark and ends with a newline mark. Each field ends with a `#` mark as well. Here is a typical entry designating an article in Advanced Applied Probability:

```
#83AdvAppPr 15 444- 459 J#On the GA/G/$\infty$ queue#Kuczek, Thomas####PV09;94m#
```

There can be other interesting marks within entries; in this instance, the `$\infty$` phrase is TeX for the infinity symbol but is of no significance for us. The pound sign delimiters and newline terminator are the only certain entries on each line. Adjacent pound sign marks signify an empty field. This line is shown here since it will fit easily on this page; many of the lines are considerably longer.

Our index will be built using author names. Accordingly, we will first construct a summary file of fixed length records (62 bytes) that contain two values; the author's name and the location of the record within the CIS database. There will be one such record for each author associated with an article. Some names in the third field contain extra TeX symbols to designate diacritical marks; in these cases the fifth field is supposed to contain the authors names with all diacritical marks stripped out. Since we will not normally input `P\'olya` but `Polya` if we would like to search for Pólya, we use the stripped name. The third field may also contain a corporate or institutional author such as the `U.S. Census Bureau`; we will not list these articles within our summary file. When there is more than one stripped spelling of a name, we are to create one summary record for each. When there is more than one stripped spelling per author, we create one summary record for each. There is one other feature of author names – there may be concluded by a *role* abbreviation. This designates among other possibilities that the person is indeed the author, an editor, a reviewer, or a translator. The role designators are `(Auth)`, `(Ed)`, `(Rev)`, or `(Transl)` respectively. We should remove these as well from our summary; almost everyone would input `Jones, J.R.` rather than `Jones, J.R. (Ed)`. The location is to be the byte offset within the database file of the associated record.

First we are to create create a summary file for our data. The records are to be 62 bytes long. These summary files will eventually be input for our linear hashing index facility. This portion of the exercise is very short; but it is a necessary step.

Our next step is to implement the linear hashing index facility. You should use `#define` statements to set compile-time parameters such as the primary and overflow block sizes; a good place to begin is to use 512 byte primary and overflow block sizes. We can store 8 records as well as leaving space for a pointer and a Bloom filter signature. While it would be nice to make these methods perfectly general and work for all sizes of data etc. we will strive for expediency here. We will need the following "public" method: `access`.

You must create "private" methods such as `split`, `insert`, `makeIndex` etc. The `access` method has two parameters – an author's name and the index name. It is to return a list of offsets within the data file of the records associated with the author; the list is empty if there are no such records. The `insert` method is used by the index provider to create the structure. the function has one parameter – an summary file record. It returns true if it is successful and false otherwise. The `makeIndex` has two parameters – the summary file name as well as the data file name. The summary file is opened and repeated calls to `insert` are made, one for each summary record. The data file name is stored for future reference. The `split` method signature is to have a signature of your design.

What to turnin

0) Groups need only turn in one copy of the work. Your `README` file should contain the names and logins for the partnership. All your files should contain this information at the beginning. The `README` should contain some concluding status remarks as well.

1) Your first task is to create a program that will generate the summary files. Let's call this program `summary.c` and the executable `summary`. I suggest that you use the standard input/output library (`stdio.h`); this will result in straightforward code. This program has one input – the name of the data file to summarize; it is to create one file with the same name prefix and `.sum` suffix.

2) Your second task is to create the linear hashing methods described above. Let's put this program in two files `lhAccess.c` with executable `access` and `lhIndex.c` with executable `lhIndex`.

3) You must submit a `makefile` that is capable of creating these three executables with commands such as `make lhIndex`, `make summary` or `access`.

4) You must submit documentation for both programs. The logic manual which describes how it works is especially important here. The user manual documents client usage.

Extra Credit

a. Make a performance study with "random" data counting the numbers of disk accesses for `access` and `makeIndex.`

b. Implement the ibim access method as well.