CSE 102 **Programming Assignment**
W.A. Burkhard **Due: February 2, 2001**

B+tree Index Maintenance

We are to create an author index, using the B+tree structure implementation, for data in the "Current Index to Statistics Extended Database" (CIS). 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 database file is arranged according to journal name; we will be constructing a dense index. The CIS database is a licensed database product.

The syntax for the data records is presented in section 3 of the `USRGUIDE.TXT` file located within the course `public/Data` subdirectory. Our index will be built using author names. Accordingly, we first construct a summary file of fixed length records (60 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.

The executable `summary` creates a summary file for the data; the records are 60 bytes long. The summary files will be sorted so we can build the B+tree index in a bottom-up fashion. The executable `sort` will sort the summary file created previously.

Our next step is to implement the B+tree index facility. A good place to begin is set the external nodes to two records and internal nodes to five keys in size. We make these methods general to work for all sizes of data. We will need the following programs: `access`, `makeIndex`, and `display`. The `access` program has two parameters – an author's name and the index name. This program returns a file listing the author's publication records within the index; the file is empty if there are no such records. The `makeIndex` program is to build the B+tree index structure; you will need to include an header record within the index to allow the `access` and `display` methods to function correctly. The program has three parameters; the name of sorted data file, the order of the B+tree to create, and the size of the external sorted summary data nodes. These are to be command line parameters: `-m <num>` for the B+tree order, `-e <num>` for the number of records in an external node, and `-f <file>` for the file containing the sorted summary data. The `display` program is to show the logical and physical structure of the B+tree; the program has one parameter – the name of the index.

Here is an example output from the `display` program showing the index built for a small, sorted summary file – `tryitData` found in `public/HW1`. The index was built using an order three B+tree and the external nodes contain two records each.

```
B+tree order is 3   external nodes contain 2 records

logical position    0:   physical position 852   count is 1      level is 2
     child[0] = 292
        Joshi, P. C.
     child[1] = 712

logical position    1:   physical position 292   count is 1      level is 1
     child[0] = 12
        Deken, Joseph
```

```
     child[1] = 152

logical position    4:   physical position 12    count is 2      level is 0
     child[0] = 0
        Baddeley, A.
     child[1] = 120
        Chambers, John M.
     child[2] = 240

logical position    5:   physical position 152   count is 2      level is 0
     child[0] = 360
        Eberhardt, Keith R.
     child[1] = 480
        Geng, Shu
     child[2] = 600

logical position    2:   physical position 712   count is 1      level is 1
     child[0] = 432
        Trampisch, H. J.
     child[1] = 572

logical position    7:   physical position 432   count is 2      level is 0
     child[0] = 720
        Liddell, Douglas
     child[1] = 840
        O'Fallon, Judith Rich
     child[2] = 960

logical position    8:   physical position 572   count is 2      level is 0
     child[0] = 1080
        Walrand, J.
     child[1] = 1200
        de Leeuw, J.
     child[2] = 1320
```

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 too. The `README` should contain some concluding status remarks as well.

1) Your task is to create programs that will generate, access, and display the B+tree index. Your `makeIndex` program should name the index file executable `<file>.index`; here `<file>.sort` designates the sorted summary data. You should use the standard input/output library `stdio.h`; this will result in straightforward code. The `makeIndex` program has three inputs – the name of the sorted summary data file, the order of the B+tree and the size of the external nodes. A first task is to architect the manner in which you will accommodate both internal and external nodes of various and differing sizes. (If you write the program for fixed sized nodes, your task to convert the program allowing for variable sizes will be lengthy.

2) The `access` program requires two input values – the name of the index (without the `.index` suffix and the author's name. The results if any, are to appear in the file named `results`.

3) The `display` program requires one input parameter value – the name of the index. The output should be directed to standard output; your program should logically position nodes as follows: the root is at position 0 and the children of the node in logical position i reside at positions

$$m \times i + 1,\ m \times i + 2,\ \cdots,\ m \times (i + 1).$$

4) You must submit a `makefile` that is capable of creating these three executables with the command `make all`; `make clean` should remove all the executables as well as the dot-oh files.

5) You must submit documentation for your programs. The logic manual which describes how it works is especially important here; you should carefully document your `#defines`. The user manual documents client usage.

Extra Credit

a. Implement range query access.

b. Implement B+trees using top-down tree construction on unsorted data. How does the time for building an index in this fashion compare with the bottom-up approach we are using.

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