CSCB09 2026 Summer Assignment 2

Due: July 5 11:59PM

In this assignment, you will practice stdio.h file I/O in C.

As usual, you should aim for reasonably efficient algorithms and reasonably well-organized, well-factored, comprehensible code.

Q0: eof and errors (0 marks)

These subquestions are not marked but you should still do them. They help you with the real question later. And some of them may appear again on tests and exams!

Write programs and create files to find out:

(a) Open a non-empty file in `"r"` mode, then write something anyway. Is it success, eof, error?

(b) The previous part, then read something. Should be a success, but what does `ferror()` still say?

(c) Find a way to cause both `feof()` and `ferror()` to return true.

(d) Open a file in `"r"` mode, then read so many bytes that eof must occur, and check that `feof()` returns 1.

Now `fseek()` to anywhere, even the end of file (`fseek(f, 0, SEEK_END)`). What does `feof()` say now? (This is already foretold by the `fseek()` man page.)

(e) Open a file that has only n bytes in `"r"` mode, then `fseek()` to $n + 10$ anyway. Does its return value say success or error?

(f) The previous part, then try to read. What do you get—success, eof, error?

(g) Open a file that has only n bytes in `"r+"` mode, then `fseek()` to $n + 10$ anyway, then write something. Is the writing success, eof, error? If success, what happens to the file now? (Be sure to `fclose()` so you are really seeing final file content.)

(h) `fseek()` to -10. Clearly an error. But what do `feof()` and `ferror()` say?

(Surprised me too. In retrospect, I guess the man page foretold it.)

Binary Search Tree on Disk

The Janus Social Network Company stores user rankings in a binary search tree in a file. That may be impractical (there are better choices such as B-trees, or even just let sqlite3 do it for you), but we are stuck with it! The file format is:

0. At the beginning there are 8 bytes that you can treat as one `long` value. It tells you the file position of the root node, or -1 if none.
1. The rest of the file has 0 or more nodes, each node being this struct (bst.h):

```
typedef struct node {
  char user[28];        // username, NUL-terminated C string
  unsigned rank;
  long left;            // file position of left child; -1 if none
  long right;           // file position of right child; -1 if none
} node;
```

2. There may be unused areas. This should not affect you if you stay focused on binary search tree traversal.

Q1: Dump (1 mark)

This question encourages the good habit of writing additional code and tools that help you check results and debug. It is worth only token marks because you should do it even if it were worth 0 marks.

Write and hand in bst-dump.c to print out the nodes in a given file. The pathname is given as the 1st command-line argument. The nodes should be printed in in-order. The output format is this for each node:

`pos=%ld user=%s rank=%u left=%ld right=%ld\n`

The value for `pos=` should be the file position of the node itself.

A sample tree file is provided (sample-tree.bin). A sample output file for that tree is provided (sample-dump.txt).

Automarking will use only valid files and existing pathnames. But you should still try to catch and report common errors for your own sanity. (E.g., in your own testing, if you have a typo in the filename, who else will remind you?) Please use stderr for error messages and your own debugging messages.

Q2: Update (9 marks)

Once in a while the company wants to increase a user's ranking by 1. Implement the following function for that in bump.c.

```
int bump(FILE *f, const char *user, int *e);
```

You may assume:

- `f` is non-null and the file is open.
- `feof(f)` and `ferror(f)` are both false before calling your function.

The return value and `e` are for reporting problems to the caller:

- If you encounter an I/O error: Set `*e` to `errno`, unless `e` is `NULL`. Return 1.
- If you encounter eof while reading (this would mean the parent gaslighted you with a bad `left` or `right` field!), return -1. You can ignore `e`.
- If the user is not found, return -1. You can ignore `e`.

The above conditions are "whichever happens first".

If everything works, return 0. You can ignore `e`.

A very basic test driver test-bump.c is provided.

Debugging And Error Messages

If you like to print debugging or error messages for your own sake, please send them to stderr only.