University of Toronto
csc228S— File Structures and Data Management, Spring 1997

Midterm Test – $\boxed{\text{Sample Solutions!}}$

Duration: 50 minutes

Aids allowed: None

Family Name: ____________________ Given names: ____________________

Student #: ____________________ Tutor: ____________________

Make sure that your examination booklet has 6 pages (including this one).

For questions that involve writing C code, comments are not necessary. If you need to call a standard C function but can't remember the correct order of arguments, just indicate the meaning of each argument.

1.	________	/ 10
2.	________	/ 8
3.	________	/ 6
4.	________	/ 3
5.	________	/ 8
Total	________	/ 35

Question 1

[10 marks in total]
Assume that the following type has been declared:

```
 typedef struct nodetyp {
     int             data;
     struct nodetyp *next;
     } Node;
```

a. Write a C function that takes a linked list as an argument and removes its first element. Your function should also return, through an argument, the integer that was stored in that first element. The return value of your function should be -1 if the linked list given to it was empty, and 0 otherwise.

```
int removeFront (Node **fp, int *val)
{
   if (*fp == NULL)
        return -1;
   else {
        Node *temp = *fp;
        *fp = (*fp)->next;
        *val = temp->data;
        free(temp);
        return 0;
   }
}
```

b. Assume that the following declarations have been made:

```
Node  *myList;
int    num;
```

and that `myList` has been made to point to a linked list with several values in it. Call your function and then print out the integer that was removed from the front of `mylist`. You may ignore the return value of your function.

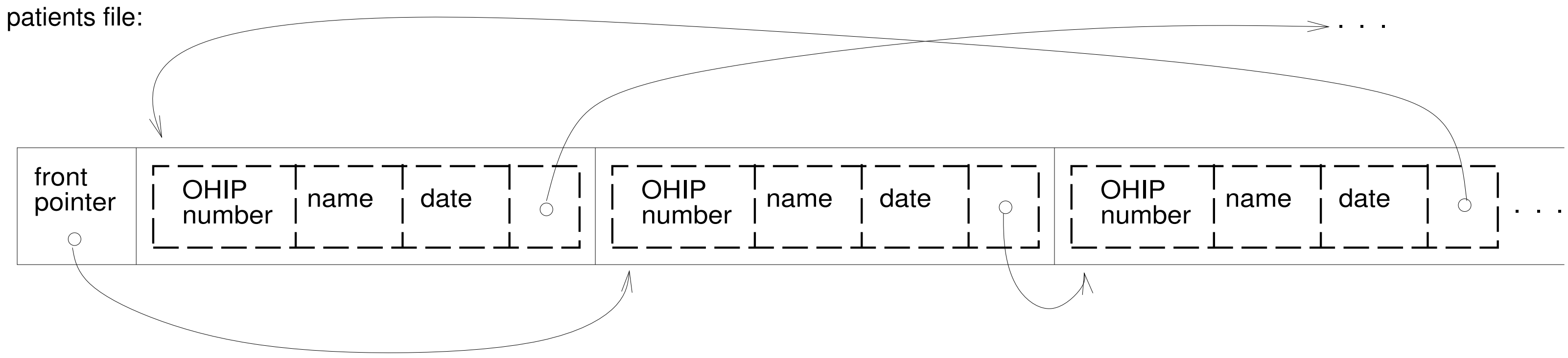
```
removeFront (&myList, &num);
printf ("%d", num);
```

Question 2

[8 marks in total]

A very simple record for a medical patient might contain an OHIP number, name, and the date when the patient checked into the hospital. Consider a file called `patients` which contains a "linked list" of such records, sorted by OHIP number. Assume that a "front pointer" is stored at the beginning of the file, in binary format.

a. In the box below, draw a picture of the beginning of the file. Show several records, the boundaries between records, and the pieces of the records. Use arrows to represent "pointers" in the file. Don't worry about making up actual values for the records; just show what goes where.



b. Declare a variable to store, in memory, the "front pointer".

```
long int front;
```

c. What C function would one use to read the "front pointer" into memory?

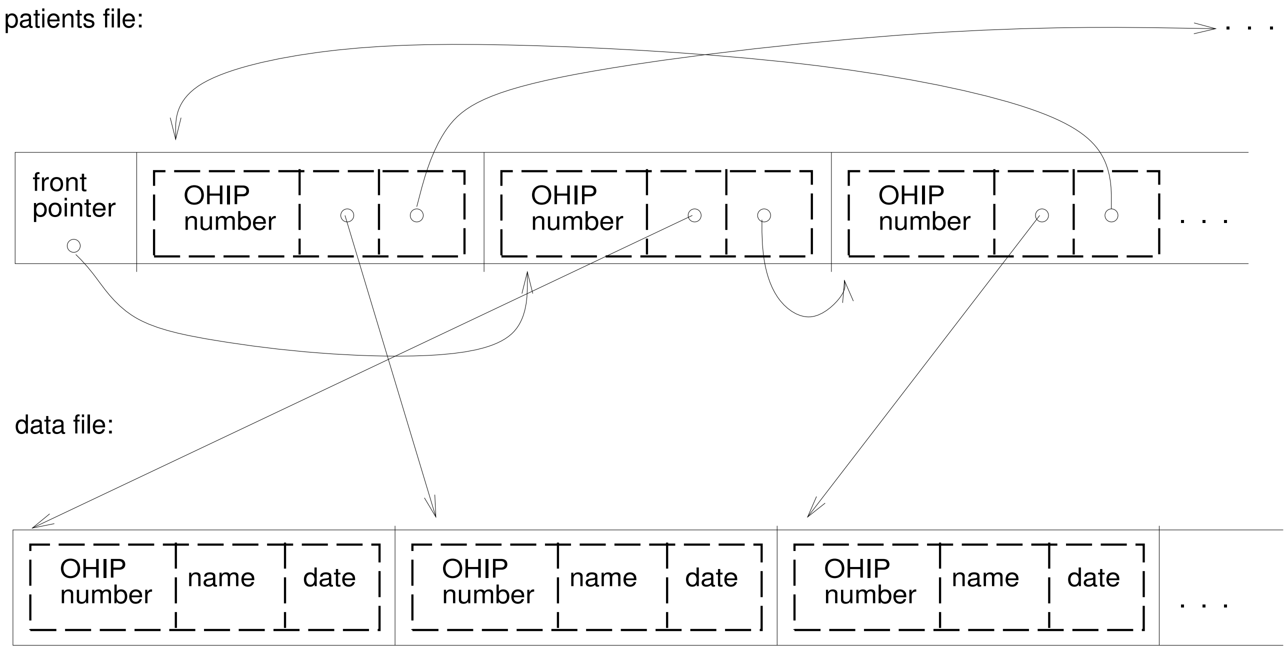
```
fread
```

d. What C function would one use to follow that pointer, and thus prepare to read the first record?

```
fseek
```

e. Imagine a different design, in which the records are stored in another file called `data`, and the `patients` file merely points to them. Thus, the `patients` file would be an index on the `data` file. Assume that the `patients file` still has the same linked list structure, sorted by OHIP number. Draw a picture of the two files, being sure to show any relationships between them.

`See next page for answer.`


patients file:
front pointer
OHIP number
OHIP number
OHIP number
. . .
data file:
OHIP number
name
date
OHIP number
name
date
OHIP number
name
date
. . .

Question 3

[6 marks in total]
Recall that cosequential processing occurs when we go through two or more files, each sequentially (*i.e.*, simply from top to bottom), and in tandem.

For each of the problems below, circle the appropriate response to indicate whether or not it can be accomplished using cosequential processing. If it can, briefly describe how the standard algorithm would be extended or modified to accomplish the task; if it cannot, explain why not.

a. **Given:** File 1 contains student records, sorted by student number. File 2 also contains student records, but stored in a linked list within the file, sorted by student number.
Problem: For each student in file 1, print out out the number of times that their surname occurs in file 2.

CANNOT be done using
cosquential processing

Extension/modification to the standard algorithm (if it can be done)
or explanation (if it cannot):

`To find (and thus count) all occurrences of a given surname within file 2 requires reading the entire file, because it is sorted by student number. So for each student in file 1, we must go through all of file 2, and that means going through the file many times overall. This is not cosequential processing. (In fact, even if file 2 was sorted by surname, we'd have to keep going back to the top of it for each new student in file 1, because file 1 is sorted by student number rather than surname.)`

`Another issue is` *how* `we go through file 2. If we traverse it by following the links in the linked list, this requires one seek per student record, whereas cosequential processing requires that we go through the file simply from top to bottom. It would be possible to ignore the links in file 2 and read it simply from top to bottom (out of order, but that would make no difference), but this still doesn't eliminate the problem mentioned above.`

b. **Given:** 5 files of student records, each sorted by student number.
Problem: Produce a new file containing all records from the 5 files, sorted by student number. That is, merge the 5 files.

CAN be done using
cosquential processing

Extension/modification to the standard algorithm (if it can be done)
or explanation (if it cannot):

`This fits the cosequential paradigm, because it can be accomplished by going through all five files sequentially, in tandem. We just need to have 5 variables for 5 current records -- one per file -- and inside the main loop we must find the smallest among all 5 of the current records.`

Question 4

[3 marks in total]

a. With a balanced binary search tree, searching is $O(\log n)$. This may acceptable if the tree is stored in memory, but may not be if the tree is stored in a file. What is the difference (between using a tree stored in memory and using a tree stored in a file) that can make $O(\log n)$ search time unacceptable in a file?

`Each time we go down a level in the tree, we only need to follow a pointer in memory if the tree is stored in memory, whereas we must perform a seek if the tree is stored in a file. This is much slower.`

b. Briefly describe the two most important differences between the structure of a B-tree and the structure of an ordinary binary search tree.

(1): `The branching factor of a B-tree must be greater than 2, so it is (potentially)` **bushier** `than a binary search tree.`

(2): `A B-tree must conform to rules that force it to be` **balanced**`, in a certain sense. This forces it to take advantage of the potential bushiness.`

Your answer should fit above this line.

Question 5

[8 marks in total]

Note: You may find the conversion tables at the bottom of this page helpful in answering this question.

To a UNIX system, text files and binary files look the same; they are both just sequences of bytes. The difference is in how the bytes are used to stand for things like words and numbers. For this question, assume that integers are represented by four bytes in memory. Specify any other assumptions that you need to make.

Assume that the following C code has been executed:

```
int num;
num = 41;
FILE *fp;
fopen (fp, "out", w);
```

a. Write a single statement in C that will write `num` to file `out` in binary.

```
fwrite (&num, sizeof(int), 1, fp);
```

b. Show (in binary) the sequence of bytes that would be written to the file as a result of your C statement from part *a*.

```
00000000   00000000  00000000  00101001
```

c. Write a single statement in C that will write `num` to file `out` as text.

```
fprintf (fp, "%d", num);
```

d. Show (in binary) the sequence of bytes that would be written to the file as a result of your C statement from part *c*.

```
00110100  00110001
```

character	ASCII value
1	49
4	52
\ n	10

decimal number	binary equivalent
1	00000001
4	00000100
10	00001010
41	00101001
49	00110001
52	00110100

END OF EXAM; TOTAL PAGES = 6