University of Toronto
CSC228F— File Structures and Data Management, Fall 1994

Midterm examination — Solutions!

Duration: 50 minutes

Aids allowed: One $8\frac{1}{2} \times 11$ inch page.

Make sure that your examination booklet has 5 pages (including this one).

Write your answers in the spaces provided. You will be rewarded for concise, well-thought-out answers, rather than long rambling ones. Write legibly.

Name: ______________________
(Circle your surname)

Student #: ______________________

Tutor: ______________________

1.	________	/ 8
2.	________	/ 4
3.	________	/ 4
4.	________	/ 8
5.	________	/ 6
Total	________	/ 30

1. (8 marks)
   Fill in the blanks appropriately in the following C program segments, so that in each case, the function will reset the variable as indicated in the comment.

   (a)
```
#include <stdio.h>
void resetInt ( int * n)
{
    *n  = 0;
}

int main (void)
{
    int num = 938;
    /* Reset num to zero. */
    resetInt ( &num );
    return 0;
}
```

   (b)
```
#include <stdio.h>
typedef struct nodetyp {
    int             data;
    struct nodetyp *next;
    } Node;

void resetLinkedList ( Node ** p)
{
    *p  = NULL;
}

int main (void)
{
    Node *front = (Node *) malloc(sizeof(Node));
    front -> data = 149;
    /* Reset the linked list to be empty. */
    resetLinkedList ( &front );
    return 0;
}
```

   (c) If you have a pointer to some data structure (for example, a linked list, as in (b) above) and you want to pass it to a function, how do you decide whether to pass the pointer itself, or a pointer to the pointer?

```
It depends on whether or not you might have to change the pointer itself within
the function.
If you might, then you must pass a pointer to the pointer; if you definitely
won't, then you can just pass the pointer itself.
(In the latter case, you could pass a pointer to the pointer, but that would
be misleading).
```

2. (4 marks)
   There are many ways arrange the storage of files on disk. If files are to be stored in equal-sized pieces, linked together, a size must be chosen for the pieces.

   (a) Describe a disadvantage that arises if the pieces are very small.

   ```
   The smaller the pieces, the more pieces there will have to be --- and hence
   the more pointers.
   Storing more pointers takes more space.
   More pointers also means more pointers have to be followed to get to any given
   spot in the file, and this takes more time.
   ```

   (b) Describe a disadvantage that arises if the pieces are very large.

   ```
   The last piece is likely to be half empty on average, and since the pieces
   are large, that's a lot of wasted space.
   ```

3. (4 marks)

   (a) Why would an operating system ever remove something from the cache?

   ```
   If the cache is already full and something new has to be brought in to the
   cache (because access to data not already in the cache is requested), then
   something has to be removed from it first to make room.
   ```

   (b) There are many possible criteria that the operating system might use to decide *which* information in the cache to remove. Describe the overall goal that these criteria attempt to satisfy.

   ```
   There are many ways to say this, including:
    • As much as possible, we want future data requests to refer to data that
      already happens to be in the cache.
    • We want the cache to contain data that is most likely to be requested again.
    • We want the data we remove from the cache to be the data that is least
      likely to be requested again.
   In any case, we are hoping to save time.
   ```

4. (8 marks)
Consider your program for assignment 2. The initial transaction file, the "calls log file", is sorted according to when the calls were made. However, the master-file update algorithm needs to access the calls in order by phone number (and secondarily, by when the calls were made). Below I describe four ways to deal with this problem. For each one, say whether or not it is a good approach, and explain why or why not.

(a) Create a linked list in memory that contains the information from the calls log file but is sorted in the order that the master-file update algorithm requires.

```
Bad idea.
The amount of data we are able to handle will be limited by the size of memory,
because we will need O(n) memory to store the linked list (where n is the
number of records in the file).
```

(b) Create a data structure in memory that contains not the call information itself, but file position indicators that point into the calls log file in such a way that if you traverse the data structure, the file pointers will direct you to the calls in the order required by the master-file update algorithm.

```
Bad idea.
While this approach takes less memory than (a) above, it still requires O(n)
memory.
So the amount of data we are able to handle is still limited by the size of
memory.
```

(c) Use a "sort-merge" method, as described in assignment 1, to sort the calls log file into the order required by the master-file update algorithm, and put the resulting sorted information into a new file.

```
This is fine.
We can choose a partition size that guarantees our petition will fit in memory,
and external merging isn't affected by memory limits.  Also this method isn't
significantly more time-inefficient than other possible methods such as that
used in assignment 2b.
```

(d) Do not reorganize the calls log file or create a reorganized copy. Instead, update the master file using a different algorithm; one that doesn't need the call information to be in any particular order.

```
This would work, and isn't limited by the size of memory.
But the master-file update algorithm would have to traverse the calls log
file p times (where p is the number of phone numbers), each time looking through
up to c records (where c is the number of phone calls).  If we could access
the calls in the proper order we'd only have to look through them once.
On the other hand, to put the calls in the proper order requires some work.
One would have to compare the overall time-complexity of the two approaches
to see which is more efficient.
```

5. (6 marks)
With some file structures, we *compute* a record's location in a file from its key. There are many methods for doing this.

   (a) With direct mapping, we simply use the key itself as the record's location. This method can waste a great deal of space. There is one circumstance under which it does not. Explain.

   `Since we need room in the file for every` *possible* `key,`
   `space isn't wasted if the number of actual keys in use is equal to or close to the number of possible keys.`

   (b) Hashing is another method. With hashing, collisions can occur. What is a collision?

   `A collision occurs when two different keys hash to the same spot. In other words, when` $h(k1) = h(k2)$ `and` $(k1 \neq k2)$.

   (c) Imagine that you are designing an information system for student records at the University of Toronto, and 9 digit student numbers are the key by which operations are to be performed. Is it possible to design a hashing scheme in which collisions never occur? If it is possible, describe such a scheme; if it is not possible, explain why not.

```
We can only be guaranteed that collisions will not occur if there is a one-to-one
mapping between keys and possible hash locations.
(One way to achieve this is with direct mapping, but we could also achieve
it by some method that "jumbles" the keys around.)
If we want to call this a trivial case of hashing, then yes, it is possble;
if we don't want to call this hashing, then no it's not.
```

END OF EXAM