University of Toronto
csc228S - File Structures and Data Management, Spring 2001

Midterm Solutions — Morning Section

Friday 02 March 2001

Duration: 50 minutes

Aids allowed: None

Family Name: ______________ Given names: ______________

Student #: ______________ Tutor: ______________

- There are 6 pages, including this one. The test is out of 30 marks and the value of each question is provided; please use this information to manage your time effectively.
- For questions that involve writing code, comments are not necessary. If you need to call a standard function but can't remember the correct order of arguments, just indicate the meaning of each argument.

Part A:	______	/ 5
Part B:	______	/ 7
Part C:	______	/ 2
Part D:	______	/ 3
Part E:	______	/ 4
Part F:	______	/ 4
Part G:	______	/ 5
Total	______	/ 30

Part A [5 marks in total]

Consider the following program:

```
#include<fstream>
int main(void){

   unsigned char u = 17;
   char c = 'A';
   char *p = new char[10];
   p[0] = 'A';
   p[1] = 'B';

   fstream out;
   out.open("outfile", ios::out | ios::bin);
   out << c;
   out.write(&c, sizeof(char));
   out.write(&u, sizeof(unsigned char));
   out.write(&p, 2*sizeof(char));
   out.write(p, 2*sizeof(char));

   return 0;
}
```

For each of the following statements, show exactly what would be written to the file, byte by byte. Write your answer in octal, using 3 digits per byte. Assume `sizeof(char)` and `sizeof(unsigned char)` are 1.

If there is not enough information to answer, explain why.

Note: The ascii value of 'A', is 65 (base 10).

```
out << c;

   101


out.write(&u, sizeof(unsigned char));

   021


out.write(&c, sizeof(char));

   101


out.write(&p, 2*sizeof(unsigned char));

   Not enough information because p is a pointer and we don't know
   its actual value.


out.write(p, 2*sizeof(unsigned char));

   101 102
```

Part B [7 marks total]

For this question, assume we are talking about the kind of B-tree used in lecture and in our textbook. Also assume that the height of a tree is the number of nodes on the longest path from the root to a leaf (including the root and the leaf).

1. Suppose we have a B-tree of order 4 (*i.e.*, where $M = 4$).

   Give a small example that shows what might go wrong if we implemented B-tree deletion so that redistribution is never considered.

   `Answer: Any example where a node underflows and both adjacent siblings have too much in them -- so that the merge overflows`

2. Suppose we have a B-tree of order 7 (*i.e.*, where $M = 7$), and that the tree contains a total of 30 keys at the leaf level.

   What is the minimum possible height of this tree? ___2___

   What is the maximum possible height of this tree? ___2___

Part C [2 marks; 1-mark penalty for a wrong answer]

Suppose we have a file of records, and are using a free list to keep track of the pieces of the file that are no longer in use. Why don't we use the operating system's free list to keep track of the unused pieces of our file? Circle the one best answer.

1. This would cause a lot of internal fragmentation.
2. This would cause a lot of external fragmentation.
3. $\boxed{\text{We can't use the operating system's free list in this way.}}$
4. We *can* use the operating system's free list in this way, but we know more about how our program needs to use the free list and so can write code for it that is more efficient.

```
Answer explanation:
The operating system's free list keeps track of unused space that is outside of anyone's
file.  It doesn't keep track of unused space within our own files -- that's up to us.
Eg.  Suppse you write 100 integers to a file, in binary say, and then want to delete
one.  There is no way for you to "give back" its space to the OS. All you can do is
mark it deleted and remember to reuse the space.  You could shift the rest of the file
up to cover over the "hole" and be left with a hole that's at the end of the file,
but even then you cannot give that space within your file back to the OS. The only
thing you can do to relinquish that space is to create a brand new file and write to
it only the wanted parts of the old file, then delete the old file.  (This is very
inefficient, which is why we use free lists.)
```

Part D [3 marks in total]

What is the big-oh time complexity of merging `n` files of records, whose sizes are s_1, s_2, ... s_n? Assume that a good algorithm is to be used.

ANSWER: $O(s_1 + s_2 + \ldots + s_n)$

Suppose that no one file fits into memory, and that the algorithm minimizes the number of reads. How many records must it have in memory at any one time?

ANSWER: n

Part E [4 marks]

Let `i` be `sizeof(int)` and `c` be `sizeof(char)`. Suppose we have declared and initialized an integer `n`.

Suppose that writing `n` to a file as text would consume *less* file space than writing it in binary. What does this tell you about `n`?
ANSWER: `(The number of digits in n) * c is less than i.`

Suppose that writing `n` to a file as text would consume more file space than writing it in binary. What does this tell you about `n`?
ANSWER: `(The number of digits in n) * c is greater than i.`

Part F [4 marks in total]

You are designing an operating system similar to Unix but with some variations. You want to have a maximum file size of 2^{32} blocks. Your blocks are 2^{16} bytes and your pointers each take 4 bytes. You are considering an inode that has 12 direct pointers, 2 pointers each to a single-level index and 1 pointer to a 2-level index.

State the maximum number of blocks addressable by your inode. You may give this answer as an equation.

ANSWER: $\underline{12 + 2^{15} + 2^{28}}$ blocks

```
Answer explanation:
2^16 bytes per block and 4bytes pointers means 2^14 pointers per block

first 12 blocks are directly accessible
each 1-level index gives access to the next (2^14) blocks
2-level index gives access to the next (2^14)^2 = 2^28 blocks

total = 12 + 2*2^14 + 2^28
      = 12 + 2^15 + 2^28
```

Is this sufficient to address all the blocks in the file? (circle one) YES / $\boxed{\text{NO}}$

```
Answer explanation:  (a back-of-the-envelope calculation)
12 + 2^{15} < 2^{28}  so the total is < 2*2^{28} = 2^{29} << 2^{32}
```

You are considering changing the maximum file size but it must be a power of 2. What is the largest n for which a file of 2^n blocks can be addressed by your inode structure?

ANSWER: 28, since $2^{28} <$ total $< 2^{29}$

Part G [5 marks in total]

Consider a file which contains only four different characters. The relative frequencies of the characters are shown in the table below.

character	frequency
A	20%
B	75%
C	4%
D	1%

If the file contained x characters in total, and you were to use fixed length codes, what is the minimum number of bytes it would take to encode the file?

ANSWER: `2x bits = x/4 bytes`

If we were to use variable length codes what is the minimum number of bytes needed to encode the file?

ANSWER: `1.3x bits = (1.3/8)x bytes`

Construct a set of valid Huffman codes for the file and show the encoding it gives for the letter C. There are a number of correct answers.

ENCODING FOR C: `any 3-digit binary number is a correct answer.`

End of Examination