Problem 1

1) **2** arguments

1 mark for correct answer

2) See code below

```
int rangeSum(int x, int y) {
   if (x == y) {
       return 0;
   } else {
       return x + rangeSum(x + 1, y);
   }
}
```

Obviously, in student answers, the function name and parameter names will vary.

6 marks total:

- **1 mark** for correct type signature *(any int, long, signed/unsigned etc type is fine)*
- **1 mark** for returning 0 when x == y *(note: award this mark even if they get the structure of the code entirely wrong but it still returns 0 when x == y)*
- **1 mark** for using an if statement with correct condition
- **1 mark** for recursively calling the function
- **1 mark** for passing the correct arguments to the recursively called function
    - 0.5 marks if they increment the wrong parameter *(e.g. because they mixed up the order of arguments on the stack)*
- **1 mark** for adding x (the first parameter) to the return value of the recursive function
    - 0.5 marks if they add y (the second parameter) *(e.g. because they mixed up the order of arguments on the stack)*

3) The function computes the sum of all integers between x (inclusive) and y (exclusive).

1 mark for reasonable explanation

1 bonus mark if they identify that the function does not terminate if x > y

Problem 2

Part 1

Question 1 [3 marks]
Physical addresses are 13 bits wide

- 2 bits are needed for the byte offset within each cache line
- 3 bits are needed for the set index
- Therefore 13 - 2 - 3 = **8 bits** are needed for the tag

1 mark for showing correct calculation of no. of bits in the tag (13 - 2 - 3 is enough working)

- 0.5 marks if correct value is used later in the question, but no working is shown

Per cache line, we need:

8 bits (tag) + 1 bit (valid bit) + 32 bits (data bits) = 41 bits

For the entire cache, we need:

16 cache lines * 41 bits per line = **656 bits = 82 bytes**

1 mark for recognising that the index is not part of the cache

- 0 marks if the student gives answers for both with and without the index and does not clearly indicate which one is the actual answer

1 mark for correct calculation with working

- We will accept solutions expressed in bits or bytes
- 0.5 marks if correct answer is given with no working shown

Question 2 [2 marks]

- 2 bits for the byte offset within each cache line
- 0 bits for index (fully associative cache)
- 13-2-0 = **11 bits** for tag

1 mark for recognising 0 bits needed for index

- If the student misread the question and changed the line count/cache size/some other value that resulted in index bits being needed, don't award this mark
- If the student gives the correct answer but does not explicitly mention that no index bits are needed, still award this mark

1 mark for correct no. tag bits

- If the student lost the previous mark due to misunderstanding of the question, award this mark as long as their explanation/working is correct and makes sense

Part 2 [3 marks]

Physical address: **0100011110001**

0.5 marks if correct, **0.25 marks** if correct but a 16-bit answer is given

Byte offset: **0x1**
Cache index: **0x4**
Cache tag: **0x47**
Cache hit: **N**

Cache byte returned: **-**

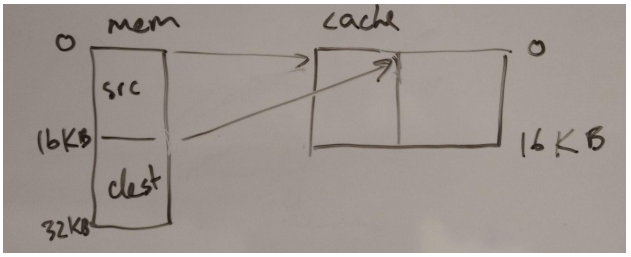
0.5 marks per correct answer (2.5 marks total)

Problem 3

Part 1

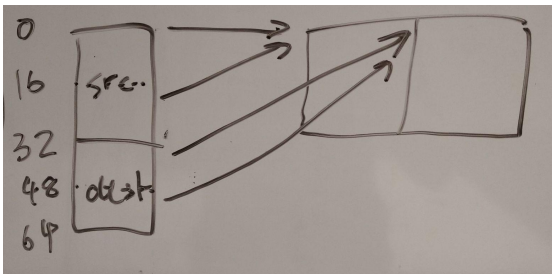
Question 1 [2 marks]

- Each array is 64 * 64 * 4 = 16,384 = 16KB in size
- Cache is 32KB in size, but is 2-way set associative, so each 16KB block of memory will map to the same cache sets (see diagram below).
- However, when we read src[i][j] and dest[i][j], even though they will map to the same cache set, they will be loaded into separate cache lines (due to cache associativity), and therefore nothing will be evicted.
- Each cache line can hold 4 integers. Since we are accessing each array in the order that it is stored in memory, we will have 1 miss (to bring the line into cache) followed by 3 hits.
    - (students may describe the access pattern as 2 hits followed by 6 misses, depending on how they phrase it. This is OK.)
- Therefore the miss rate is **25%.**



Question 2 [2 marks]

- Each array is 64 * 128 * 4 = 32,786 = 32KB in size
- Since this is a multiple of 16KB, the cache behaviour will be very similar to the previous question
- When we are half way through iterating through each array, we will have to start evicting items from cache to make room for new data
- However, this will not impact the miss rate because we were never going to use that cached data again anyway
- Miss rate is **25%**



For each question (Q1 and Q2):

1 mark for the correct miss rate

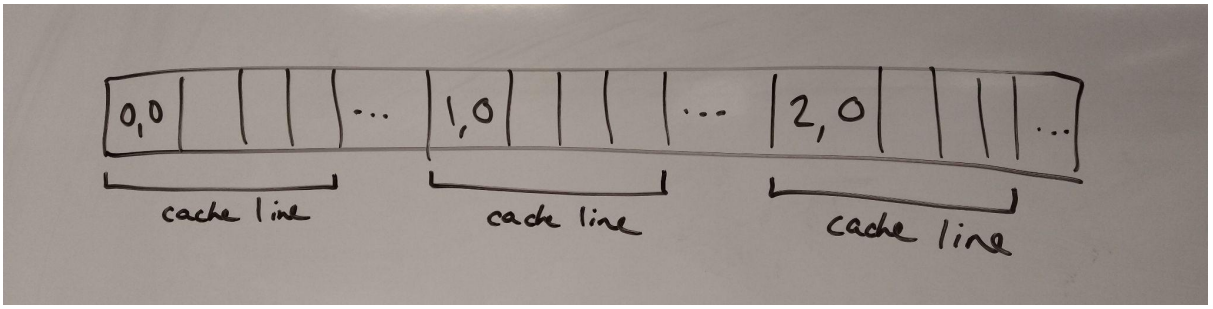
- If the student writes 75% and it is clear from their working that they accidentally wrote the hit rate instead of the miss rate, award this mark

1 mark for a reasonable explanation

- Two key components: how the addresses map to cache lines, and the hit/miss pattern. Award 0.5 mark if only one of these is present.
- Be liberal in what explanations you accept as long as they are reasonable
- Diagrams are completely acceptable (check whether they have included one)

- In Q2, students may refer back to Q1 if they want to make their explanation shorter

Part 2 [3 marks]



- Since the arrays are stored in row-major order, iterating row-by-row will access non-contiguous chunks of memory.
- Therefore, *initially*, each memory access will result in a cache miss, because a different cache line is accessed (see diagram above).
- However, as shown in Part 1 Question 1, the entire working dataset fits in the cache, and no more than 2 lines in memory ever map to the same cache set, so nothing is evicted
- Therefore, after the entire dataset has been loaded into cache, subsequent accesses will be hits, regardless of the (usually slow) access pattern.
- Therefore the miss rate is also **25%**

(we didn't intend for this to be a trick question when designing the exam, forgot about the size of the dataset… rip students. Therefore we have decided to award most of the marks even if they get the wrong answer)

If the student gets the correct miss rate, they can receive up to 3 marks

- **1 mark** for writing the correct miss rate (25%)
    - If the student writes 0% and it is clear from their working that they accidentally wrote the hit rate instead of the miss rate, award this mark
- **2 marks** for identifying/explaining how the entire working dataset fits in cache
    - They do not need to explicitly talk about the (bad) memory access pattern, because it is unimportant in this particular case. It suffices to explain why a cache line is never evicted after it has been loaded for the first time
    - They can refer back to Part 1 Question 1 if they wish
    - Award partial marks as appropriate

If the student gives 25% as their answer, they can receive up to 2 marks

- **0.5 marks** for writing the incorrect miss rate (25%)
    - If the student writes 75% and it is clear from their working that they accidentally wrote the hit rate instead of the miss rate, award this half mark

- **1.5 marks** for explaining how this memory access pattern (usually) results in a lot of cache misses

Note if the student gives an answer other than 25% or 100% (or 75% or 0% if they clearly accidentally swapped miss/hit rates), then they are not eligible for any marks (except for partial marks for their explanation if they were close).

Problem 4

If the student assumes that memory is word-addressed (and therefore each page consists of 2^10 addresses), then mark using the same marking criteria as below, but refer to the alternate answers in green. Cap the mark to 14/16.

Part 1 [1,1,2,2,2 = 8 marks]

1. 20-bit virtual addresses, 4096 byte pages (12 bits needed for page offset), therefore 20-12=8 bits for virtual page number, total of **2^8 = 256 entries** in the page table
   10 bits needed for page offset, therefore **2^10 = 1024 entries** in the page table
   1 mark for correct answer (no working required), **0.5 marks** for incorrect answer if mostly correct working provided but has a minor error
2. 16-bit physical addresses, 12 bits used for page offset, therefore **2^4 = 16 physical pages** (max)
   10 bits used for page offset, 16-10=6, **2^6 = 64 physical pages**
   1 mark for correct answer (no working required), **0.5 marks** for incorrect answer if mostly correct working provided but has a minor error
3. Yes, it is possible for multiple virtual pages to be mapped to the same physical page. Examples include:
   - Shared libraries being shared across processes
   - 2 processes running the same binary?
   - The same file mapped twice using mmap()
   - …?

   1 mark for identifying the correct answer (yes)
   1 mark for providing an example
4. Bits **13-12** **(11-10)** are used for the TLB index, and bits **19-14** **(19-12)** are used for the TLB tag. The lower-order bits of the virtual page number are used for the TLB index because we want adjacent pages to index into different TLB sets. This will give high-locality accesses higher performance. All remaining bits are used for the tag.
   0.5 marks for identifying the TLB index bits
   0.5 marks for identifying the TLB tag bits
   - Note that students may use different bit numbering (e.g. zero or one indexed, highest or lowest order bit equals zero) - look at their working and mark as correct if their answer makes sense for their counting scheme

   1 mark for giving a reasonable explanation (incl. calculation)
5. It would dramatically increase the metadata overhead required and complicate address translation if virtual and physical pages are different sizes.
   For instance, if virtual pages were smaller than physical pages, it would be necessary to translate the virtual page offset into a physical page offset. If virtual pages were larger than physical pages, then it would be necessary to lookup both the virtual page number AND the virtual page offset to determine which physical page to access.
   2 marks for some reasonable explanation (be liberal in what you accept, award partial marks at your discretion)

Part 2

Question 1 (0x12A58) [4 marks]

Virtual address: 0001 0010 1010 0101 1000

VPN	0x12 (0x04A)
TLB Index	0x2 (0x2)
TLB Tag	0x04 (0x12)
TLB Hit	N (N)
Page Fault	Y (unclear)
PPN	- (not in table)

If the student made the page addressing error earlier, then the VPN does not appear in the page table given in the exam. Award marks for “Page Fault” if the answer they have given is justified by their answer to PPN.

Physical address: - (-)

0.5 marks per correct answer (4 marks total)

Question 2 (0x8147C) [4 marks]

Virtual address: 1000 0001 0100 0111 1100

VPN	0x81 (0x205)
TLB Index	0x1 (0x1)
TLB Tag	0x20 (0x81)
TLB Hit	Y (N)
Page Fault	N (unclear)
PPN	0x4 (not in table)

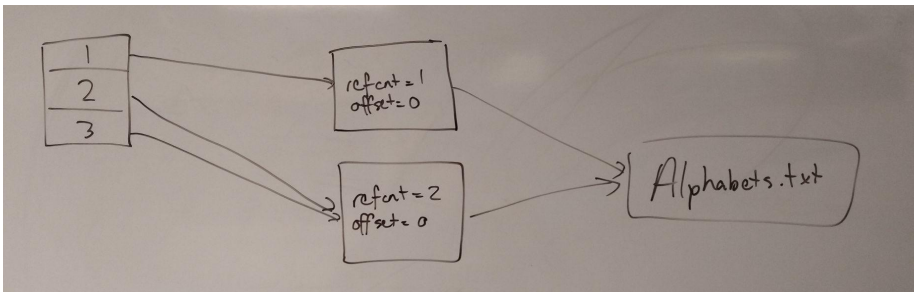
Physical address: 0100 0100 0111 1100 (not in table)

Once again, award marks for page fault, PPN, physical address if their answer matches their justification.
0.5 marks per correct answer (4 marks total)

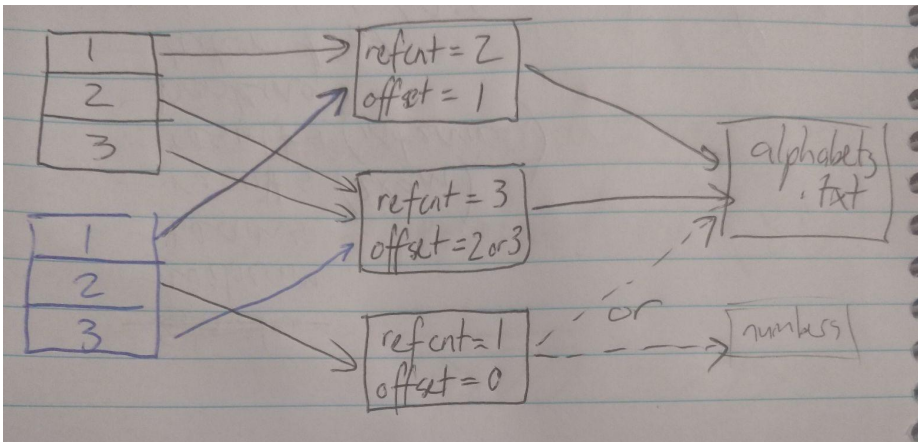
Problem 5

Part 1

Line A



Line B



For each diagram:
1 mark for rough structure
2 marks for correct structure
1 mark for correct metadata (filepos only)
1 bonus mark overall if stdin/stdout/stderr included

Part 2

If child is scheduled before parent: **[2 marks]**
aa|bac|debf
If parent is scheduled before child and manages to print: **[2 marks]**
aa|b|cad|ebf
(Bonus marks) If parent is scheduled before child, reads, but does not print: **[1 mark]**
aa||cad|bebf
1 mark for writing 2 invalid traces
If there is a minor error (e.g. they miss the c/d at the end of the child process), **-0.5marks**
For each additional invalid trace added, **-1 mark** (min of 2/4 for this part if they found both correct traces)

Problem 6

Answer: A, B, C, and D are valid outcomes. E is invalid.

Example traces are given below: **note these are not the only possible traces**

A (valid)	**B (valid)**	**C (valid)**	**D (valid)**
a=2 print(b,c) b=2 print(a,c) c=2 print(a,b)	a=2 b=2 print(a,c) print(b,c) c=2 print(a,b)	c=2 print(a,b) b=2 print(a,c) a=2 print(b,c)	b=2 c=2 print(a,c) print(a,b) a=2 print(b,c)
a=2 print(b,c) c=2 print(a,b) b=2 print(a,c)	b=2 a=2 print(b,c) print(a,c) c=2 print(a,b)	b=2 print(a,c) c=2 print(a,b) a=2 print(b,c)	c=2 b=2 print(a,b) print(a,c) a=2 print(b,c)

E (invalid):
In order for 22 to be printed first, two of the variables must have been previously set to 2. However, in order for 00 to be printed at the end, at least two of the variables must still be 0. This is not possible without violating sequential consistency.

For each output:

1 mark for correctly identifying it as Valid/Invalid
1 mark for providing either:

- A trace of instructions that would give the output (if it is valid), or
- An explanation of why the output is invalid

If the student has an invalid trace but gets the right answer, then 0/2 for that part.

Problem 7

2 marks if they answer that the argument of free() is invalid, but explain why, and say that it's because bit 0 indicates the block is free, we give them **2**

or

1 mark per correct answer (6 total) for each of the memory locations below:

(1) 0x00000031
(2) 0x00000012 or 0x00000010
(3) 0x00000021
(4) 0x00000013
(5) 0x00000021
(6) 0x00000031

Problem 8

Part 1 (12 points)

Perfect solution (7284920 is an example):

- Three opened files and three mmap() calls.
- Correct logic for copying keys and values to the key.txt and values.txt files.
- Correct offsets use when copying (keysize = 32 bytes, int size = 4 bytes)
- Both memcpy and char-by-char copying are equally acceptable
- The size of the offset is ambiguous. Accept both 4 and 8 bytes.

Exceptions:
Some misunderstood the question and only did mmap() for the database file and used read/write syscalls for the other two files. We should make an exception and give a perfect score. Example is 7316096.

Other Solutions:

- Only one mmap() of the database file and using file descriptors as pointers for the other two files. And copying logic is wrong. (4 points)
- Three mmap() calls but copying logic is wrong (6 points)
- No mmap() call (0 points)
- If they store the absolute address instead of the offset (minus 2)

Part 2 (6 points)

If 1MB = 1024*1024B = 1048567B, then:
3 platters * 2 surfaces * 20000 cylinders * 500 sectors * 512 bytes per sector / (1024 * 1024)
= 29296.875 ~= 29,296 = 14,648.4375 (2 mb)

If 1MB = 1000 * 1000B = 1000000B, then:
3 platters * 2 surfaces * 20000 cylinders * 500 sectors * 512 bytes per sector / 1000000
= 30,720 (div 2) = 15,360

3 marks for correct answer
3 marks for correct working

SHOULD BE around 15,000, need to divide

Up to **4 marks total** If they forgot that each platter has 2 surfaces
Up to **5 marks total** if they made some minor calculation error but were otherwise correct