CSE 351 Summer 2025, Final Exam
August 22, 2025

Full Name:	
UW NetID:	@uw.edu
Seat Number:	
I certify that all work is my own. I had no prior knowledge of exam contents nor will I share the contents with any student in CSE 351 who has not yet taken the exam. Violation of these terms may result in a failing grade. **Signature:**	

Instructions:

- **Do not turn the page until you are instructed to begin.**
- This exam is closed book (no smartphones, calculators, or other electronic devices). Turn off any mobile devices, remove hats, headphones, and smartwatches, and put them away.
- You are allowed one page (U.S. letter, double-sided) of *handwritten* notes. **Write your name and NetID on your notes page and turn it in with your exam.**
- This exam contains 5 problems and 1 bonus question spread across 7 pages. The last page is a reference sheet. You may detach it from the rest of the exam.
- When a box or line is provided, please write your answer in the box or on the line.
- If a question involves bubbling in a ◯, please fill the shape in completely.
- You have 60 minutes to complete this exam. **Please stop writing when the clock stops.**

Advice:

- Read the questions carefully before starting. Make sure you understand what they're asking before you start working on them!
- **This is a long exam!** There are plenty of opportunities to show what you've learned. Don't spend too much time on any one problem, skip around, and make sure you have the chance to attempt all the questions.
- **Relax! You are here to learn :)**

Question	**1**	**2**	**3**	**4**	**5**	**6**	**Total**
Points	6	11	10	13	10	1	**51**
Page	2	3	4	5	6	7	

Name: _____ UW NetID: _____

1. (6 points) Processes

Consider the C function `foo()`:

```
void foo() {
  int x = 2;
  int y = 0;
  int* p = malloc(sizeof(int));
  *p = 0;

  while (y < x) {
    if (fork()) {
      y = 100;
    }
    printf("x%d ", x);
    *p += x;
    x--;
  }

  printf("p%d ", *p);
  free(p);
}
```

(a) (2 points) How many processes are created when we run `foo()`, including the original (parent) process?

3

(b) (4 points) Is it possible for `foo()` to produce the output `"x2 x2 p2 p3 x1 p3 x1"`? Explain why or why not in 1 – 2 sentences.

◯ Yes (Possible) ◯ No (Impossible)

Explain:

No, `foo()` can never produce this output. Because all our processes must end by printing `p*` for some value of `*`, we can never see an output that ends with `x1`. In addition, any `p3` printed must be preceded by an `x1`, and the first `p3` in this output is not.

Name: _____ UW NetID: _____

2. (11 points) Caching

We run the C code below on a system with the following parameters:

- 8 KiB physical address space
- 512 B cache size
- 2-way set associativity
- 16 B cache blocks
- Write-back, write-allocate

```
struct AB {
  long a;
  char b;
};

struct AB X[16]; // X starts at address 0x0800

// ...initialize X...

for (int i = 0; i < 16; i += 2) {
  X[i].a = X[i].a + X[i].b;
  X[i].b = X[i+1].b + 1;
}
```

(a) (4 points) Give the address size and TIO breakdown for this system:

Address bits: _____13_____ **Tag bits:** _____5_____

Index bits: _____4_____ **Offset bits:** _____4_____

(b) (4 points) What is the miss rate of the code above? Ignore any code not shown. Assume that the cache starts cold (empty) and `i` and `j` are stored in registers. For partial credit, you may *optionally* provide the access pattern of this code.

Miss rate: _____2/5 (40%)_____

***Optional* access pattern for partial credit** (you may not need all boxes)**:**

Address:	**R 0x800**	**R 0x808**	**W 0x800**	**R 0x818**	**W 0x808**	**(i=0)**
Hit/Miss:	**Miss**	**Hit**	**Hit**	**Miss**	**Hit**	
Address:	**R 0x820**	**R 0x828**	**W 0x820**	**R 0x838**	**W 0x828**	**(i=1)**
Hit/Miss:	**Miss**	**Hit**	**Hit**	**Miss**	**Hit**	

(Question continues on next page.)

(c) (3 points) For each of the following changes, would the miss rate increase, decrease, or stay the same? For each change, assume all other parameters stay the same.

(i) Change the cache associativity to **direct-mapped**

◯ Increase ◯ Decrease ◉ Stay the same

(ii) Increase the block size to **32 B**

◯ Increase ◉ Decrease ◯ Stay the same

(iii) Change the write-miss policy to **no-write-allocate**

◯ Increase ◯ Decrease ◉ Stay the same

3. (10 points) Java & C

(a) (6 points) For each of C and Java, indicate whether each operation below is **Possible**; **Impossible**, meaning you could never compile a program that attempts the operation; or an **Error**, meaning that the operation can be attempted, but will always result in a runtime error.

(i) Index past the bounds of an array.

C: ◉ Pos. ◯ Imp. ◯ Err. **Java**: ◯ Pos. ◯ Imp. ◉ Err.

(ii) Dereference the null pointer.

C: ◯ Pos. ◯ Imp. ◉ Err. **Java**: ◯ Pos. ◯ Imp. ◉ Err.

(iii) Get the address of an `int` field in an instance of a C struct/Java Object.

C: ◉ Pos. ◯ Imp. ◯ Err. **Java**: ◯ Pos. ◉ Imp. ◯ Err.

(b) (4 points) You've been hired to consult on the login module for a new Linux app. The module will involve reading user input, comparing it against a stored password, and maintaining the user's login state in dynamic memory.

Your employer wants to know whether they should use C or Java to write the module. Which language do you tell them they should pick? Explain your answer by referring to the tradeoffs between C and Java, and give one disadvantage of this choice.

Either C or Java accepted with an appropriate explanation and disadvantage.

Example: The employer should pick Java because Java is not vulnerable to buffer overflow attacks or memory (mis)management bugs the way C is. This property is particularly important for security-critical applications like login authentication. One disadvantage of this choice is that the module will be slower and take up more memory than if it were implemented in C.

Name: _____ UW NetID: _____

4. (13 points) Virtual Memory & the Memory Hierarchy

Consider a system with the following parameters:

- 64 KiB virtual address space
- 11-bit physical addresses
- 256 B pages
- A 4-entry direct-mapped TLB with LRU replacement

(a) (4 points) Fill in the following values:

Page offset bits: _____8_____ **VPN bits: _____8_____**

TLB index bits: _____2_____ **TLB tag bits: _____6_____**

(b) (9 points) Assume that the TLB and page table begin in the state shown below:

Set	TLBT	Valid	PPN	Set	TLBT	Valid	PPN
0	0b10 1010	0	0x4000	**2**	0b11 1111	1	0x10AA
1	0b00 1000	1	0x5008	**3**	0b00 0110	0	0x3801

Partial page table:

VPN	PPN	Valid	VPN	PPN	Valid
0x21	0x5008	1	0xA8	0x4000	0
0x50	0x12CC	1	0xC1	0x3FF0	1

Fill in the table below with the results of each virtual memory access in the sequence. Each access occurs after the access(es) before it. If a given box cannot be determined, write "ND".

Access	TLB hit? (Hit/Miss)	Page Fault? (Y/N)	PPN (in hex)
Read 0xA800	**Miss**	**Y**	**ND**
Read 0xC140	**Miss**	**N**	**0x3FF0**
Read 0x21BB	**Miss [conflict]**	**N**	**0x5008**

Name: _____ UW NetID: _____

5. (10 points) Dynamic Memory Allocation & Memory Bugs

Consider the following C code:

```
1     long* ptrs[6];

2     for (int i = 0; i < 6; i++) {
3       if (i%2 != 0) { // i is odd
4         ptrs[i] = malloc(sizeof(long)*i);

5       } else {
6         ptrs[i] = ptrs[i - 1];
7         *(ptrs[i]) = i;
8       }
9     }

10    for (int j = 0; j < 6; j++) {
11      free(ptrs[j]);
12    }
```

(a) (4 points) The code above contains <u>at least two</u> memory-related bugs! Identify <u>two</u> of these bugs, making sure to reference the <u>line(s) of code</u> where they occur.

You do not need to use the exact wording from lecture for each type of bug, but it should be clear what problem you're describing. The order in which you give the bugs does not matter. You also do <u>not</u> need to say how to fix each bug.

First bug:

Dereferencing a non-pointer on line 7 (when i = 0)
OR Freeing a non-pointer on line 11 (when j = 0)
OR Double-free on line 11 (when j is even and > 0)
Also accepted (though not allocation-related): indexing out of bounds on line 6 (when i=0)
NOT a bug: wrong allocation size. We correctly allocate enough space for 1-5 longs, and store the resulting long* (implicitly cast) in an array of long*'s.

Second bug:

A different bug from the same list.

(Question continues on next page.)

(b) (6 points) Suppose that our heap allocator uses an explicit free list like the one from Lab 5: 8-byte alignment, a one-word header, and a one-word footer that may be overwritten in allocated blocks. Assume also that the heap is initially empty, and that it uses a first-fit allocation strategy.

The C code on the previous page calls `malloc()` three times. For each call, how large is the allocated heap block, and how much of that block is internal fragmentation (all in bytes)?

	Heap block size	Internal fragmentation
First `malloc()`:	**32 [min block size]**	**24**
Second `malloc()`:	**32**	**8**
Third `malloc()`:	**48**	**8**

6. (1 point) What has been your favorite topic in CSE 351 this summer?

All answers accepted.

This page left intentionally blank.