University of Washington — Computer Science and Engineering
Summer 2026 - Instructor: Naama Amiel - 2026-08-20

CSE 351 FINAL (PART 1)

Last Name:	Perfect	
First Name:	Perry	
UW netID:	pperfect@uw.edu	
Name of person to your Left | Right:	Assembly Anne	Dubs the Husky
All work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CSE351 who haven't taken it yet. Violation of these terms could result in a failing grade and a report to CSSC. (please sign)		

<u>Do not turn the page</u> until instructed to do so by a staff member.

Instructions

- This exam contains 6 pages. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- The last page is a reference sheet. Please detach it from the rest of the exam.
- The exam is closed book (no laptops, tablets, wearable devices). You are allowed two pages (US letter/A4, double-sided) of handwritten notes. Scientific calculators are allowed.
- Please silence and put away all cell phones and other mobile or noise-making devices.
- Remove all hats, headphones, and watches.
- You have 60 minutes to complete this exam.

Advice

- Read questions carefully before starting. Skip questions that are taking a long time.
- Read all questions first and start where you feel the most confident.
- Relax. You are here to learn. You can do it! 🍀

Question 1: Number Representation

Suppose we want to create a representation scheme to store the **start times of classes throughout the week**. A class can start at any ten minute interval throughout the day (e.g. 7:30, 10:40, or 5:50). There are 7 days in a week, 24 hours in a day, and 6 10-minute intervals per hour.

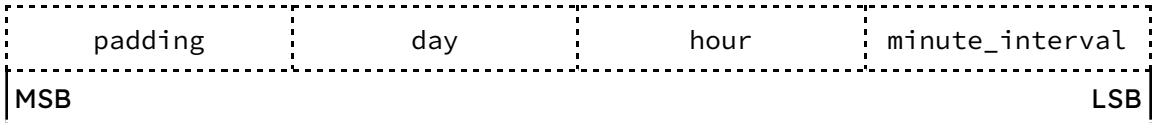
Sierra proposes the following representation, which uses three fields:

- `day`, which ranges from 0-6 (Sunday - Saturday)
- `hour`, which ranges from 0-23 (12:00 AM - 11:00 PM)
- `minute_interval`, which ranges from 0-5 (XX:00 - XX:50)

A) What is the **minimum number of bits** required for each field?

`day`: 3 bits `hour`: 5 bits `minute_interval`: 3 bits

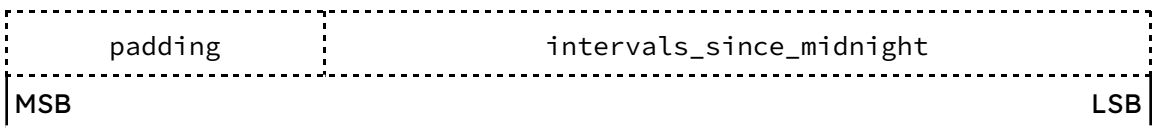
B) Assume the fields are stored contiguously in a C `short` with unused `padding` in the most significant bits, followed by `day`, then `hour`, then `minute_interval` in the least significant bits:



Write an expression which, given an instance of this representation (stored as a `short` variable named `time`), evaluates to the value of `time`'s `hour` field.

Expression: (time & 0xF8) >> 3 **OR** (time >> 3) & 0x1F

Anthony proposes a *different* representation, also stored in a C `short`, which instead counts the **number of 10-minute intervals** since midnight at the start of the week (meaning there are 1,008 possible intervals) as an unsigned integer value:



[Continued on next page]

C) Imagine you are in charge of scheduling at University of Washington. As part of this, you must frequently:

- **Postpone** or **move up** a class by 20-30 minutes,
- Determine the **difference in start times** (in minutes) between two classes.

Which representation would you use? CIRCLE ONE and explain your answer.

Sierra's Representation (3 fields)	Anthony's Representation (intervals)
Explanation **(1-2 sentences):** Since Anthony's representation is only one field, we can perform arithmetic on it directly. This means that postponing or moving up a class can be easily done by adding or subtracting the number of 10-minute intervals. Similarly, two class times in Anthony's representation can be subtracted directly to get the difference (in number of 10-minute intervals, which can then be multiplied by 10).	

D) Now imagine you are a University of Washington commuter student trying to schedule classes for yourself. Your goals for your schedule are as follows:

- Have classes on **as few weekdays** as possible,
- **Avoid** classes that **start too late** in the day.

Which representation would you use? CIRCLE ONE and explain your answer.

Sierra's Representation (3 fields)	Anthony's Representation (intervals)
Explanation **(1-2 sentences):** Sierra's representation allows us to isolate individual fields and directly read the day field (to identify the weekday a class lands on) and the hour field (to identify how late in the day a class is). With Anthony's representation, we would need to use arithmetic to calculate the day/hour instead.	

Question 2: Pointers and Memory

For this problem, we are using a 64-bit x86-64 machine (little endian). A partial view of the current state of memory (values stored in hex) is shown below:

```
// kitty is shown in memory
// &idx = 0x120
short  kitty = 0xCA7;
short  vals[8];
char*  idx[3];
```

Word Addr	+0	+1	+2	+3	+4	+5	+6	+7
0x100	33	35	31	00	43	53	45	00
0x108	21	21	00	79	61	79	21	00
0x110	2C	01	0C	A7	FF	00	A7	0C
0x118	07	DE	00	00	00	00	00	00
0x120	04	01	00	00	00	00	00	00
0x128	00	01	00	00	00	00	00	00
0x130	0B	01	00	00	00	00	00	00
0x138	28	01	00	00	00	00	00	00

A) **How many bytes** are allocated by the code snippet above?

Optional (for partial credit):

Bytes allocated by **short** `kitty = 0xCA7;`: 2

Bytes allocated by **short** `vals[8];`: 16 (2 * 8)

Bytes allocated by **char*** `idx[3];`: 24 (8 * 3)

42 bytes

Also consider this snippet of the ASCII table:

'a'	'b'	'y'	'z'	'B'	'C'	'D'	'E'	'S'	'1'	'2'	'3'	'4'	'5'	'!'
0x61	0x62	0x79	0x7A	0x42	0x43	0x44	0x45	0x53	0x31	0x32	0x33	0x34	0x35	0x21

B) What is printed by the following loop?

```
for(int i = 0; i < 3; i++)
    printf("%s ", idx[i]);    // %s prints a value as a string
```

Output: CSE 351 yay!

C) Give the **C type** and **value** (omit leading zeros) of the following expressions:

Expression	C Type	Value
`&kitty + 2`	short*	0x11A
`idx[-1] + 1`	char*	0xDE08

Question 3: C & Assembly

Consider the following x86-64 assembly for a function `mystery`:

```
mystery:
        movl    $0, %ecx                # Line 1
        movl    $0, %eax                # Line 2
        jmp     .L2                     # Line 3
.L3:    addl    $1, %ecx                # Line 4
.L2:    cmpl    %edx, %ecx              # Line 5
        jge     .L5                     # Line 6
        movslq  %ecx, %r8               # Line 7
        movl    (%rdi,%r8,4), %r8d      # Line 8
        testb   $1, %r8b                # Line 9
        jne     .L3                     # Line 10
        addl    %r8d, %eax              # Line 11
        jmp     .L3                     # Line 12
.L5:    imull   %esi, %eax              # Line 13
        ret                             # Line 14
```

A) What is the **C type** of each **argument** in `mystery`? *Not all arguments may be used. Leave the C types of arguments that are not part of the function call blank.*

Argument	arg1	arg2	arg3	arg4	arg5
C Type	`int*`	`int`	`int`		

B) `mystery` contains a for-loop and an if-statement. Fill in the C code snippet of the function body of `mystery` below to match the behavior of `mystery`'s assembly.

```
eax = 0;

for (__ecx = 0__; _ecx < edx__; ___ecx++___) {

    if (_____rdi[ecx] % 2 == 0_____) {

        ____eax += rdi[ecx]____;

    }
}

return ___eax * esi___;
```

Use register names as variable names (e.g., `rdi` for the register `%rdi`).

Other correct options for the condition (`& 1`, `!= 1`, etc.) were accepted. `*()` instead of `[]` also accepted.

Question 4: Procedures & The Stack

The function `power` raises a given base to a given exponent. Below is `power`'s C code and its corresponding disassembly:

```
int power(int base, int exp) {
    if (exp == 0)
        return 1;
    return base * power(base, exp - 1);
}
```

```
0000000000401126 <power>:
  401126:  85 f6                test   %esi,%esi
  401128:  75 06                jne    401130 <power+0xa>
  40112a:  b8 01 00 00 00       mov    $0x1,%eax
  40112f:  c3                   ret
  401130:  53                   push   %rbx
  401131:  89 fb                mov    %edi,%ebx
  401133:  83 ee 01             sub    $0x1,%esi
  401136:  e8 eb ff ff ff       call   401126 <power>
  40113b:  0f af c3             imul   %ebx,%eax
  40113e:  5b                   pop    %rbx
  40113f:  c3                   ret
```

A) What value is pushed onto the stack when `power` recursively calls itself?

`0x40113b`

B) The **4-byte immediate** `0x1` in the instruction `mov $0x1, %eax` (highlighted) can be found in the **disassembly**. What is the address of this immediate?

`0x0x40112b`

C) One of the arguments to `power` could cause **stack overflow**. CIRCLE the name of the argument and provide a value it would have that causes stack overflow.

Argument: `base` `exp` Value: -1 (or a very large positive #)

D) Suppose `main` calls `power(2, 3)`. Fill in the given memory diagram of the stack at the moment the stack is at its **largest**. Include a brief description of each entry as well as its value. If the value is unknown, write "unknown".
You may not need all the lines provided.

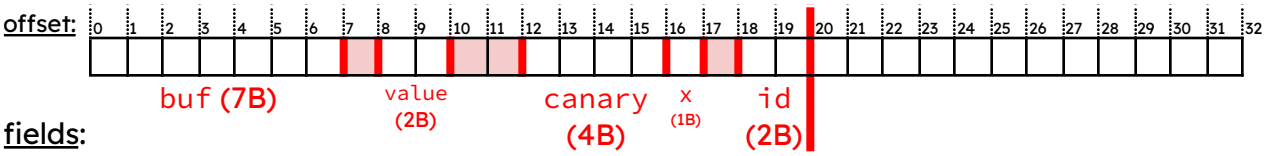
Address	Description	Value
0x7fffffffe028	Return address to `main`	unknown
0x7fffffffe020	`main`'s `%rbx` value	unknown
0x7fffffffe018	Return address to `power`	0x40113b
0x7fffffffe010	Old `%rbx` value	2
0x7fffffffe008	Return address to `power`	0x40113b
0x7fffffffe000	Old `%rbx` value	2
0x7fffffffdff8	Return address to `power`	0x40113b
0x7fffffffdff0		
0x7fffffffdfe8		
0x7fffffffdfe0		
0x7fffffffdfd8		

Question 5: Arrays and Structs

Consider the following definition for a struct:

```
struct s {
    char buf[7];
    short value;
    unsigned int canary;
    char x;
    short id;
};
```

A) Use the boxes below to draw the layout of an instance of `struct s` (1 box = 1B). Clearly label each field and its size (in bytes). Clearly **shade** any fragmentation. Draw a **clear**, **labeled** line indicating the end of the struct. Some provided boxes may be unnecessary.



B) How many bytes of internal and external fragmentation are in the struct, and what is the overall struct size?

Internal Frag: 4 Bytes | External Frag: 0 Bytes | Struct size: 20 Bytes

C) Consider writing into the `buf` field of a `struct s` instance using `gets()`:

```
struct s inst;
gets(inst.buf);
```

How many bytes would you need to write to `stdin` to **completely overwrite the `value` field** in `inst` with the value `0xE41C`?

10 bytes

D) Assume that multiple instances of `struct s` are stored in an array:

```
struct s arr[12];
```

Which of the following suggestions is better for **protecting** the `value` field of ***any*** `struct s` instances from **undetected buffer overflows of the `buf` field**?

Suggestion A: Move the `canary` field before the `value` field; if the value of `canary` has changed, alert that tampering has occurred:

```
struct s{
    char buf[7];
    unsigned int canary;
    short value;
    char x;
    short id;
};
```

Suggestion B: Remove the `canary` field; instead, rearrange the struct fields to move sensitive fields before the buffer:

```
struct s {
    short value;
    char x;
    short id;
    char buf[7];
};
```

CIRCLE the best suggestion and justify your answer in 2-3 sentences.

Suggestion A	Suggestion B

Explanation **(2-3 sentences):** Using Suggestion A, a changed `canary` field will correctly indicate buffer overflow in at least one element of the array. While Suggestion B protects against overflows of the `value` field in the same array element, it is still possible to overflow the `buf` field of an earlier array element to change the `value` field of the next element.

END OF EXAM

University of Washington — Computer Science and Engineering
Summer 2026 - Instructor: Naama Amiel - 2026-08-21

CSE 351 FINAL (PART 2)

Last Name:	Perfect	
First Name:	Perry	
UW netID:	pperfect@uw.edu	
Name of person to your Left | Right:	Assembly Anne	Dubs the Husky
All work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CSE351 who haven't taken it yet. Violation of these terms could result in a failing grade and a report to CSSC. (please sign)		

<u>Do not turn the page</u> until instructed to do so by a staff member.

Instructions

- This exam contains 5 pages. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- The last page is a reference sheet. Please detach it from the rest of the exam.
- The exam is closed book (no laptops, tablets, wearable devices). You are allowed two pages (US letter/A4, double-sided) of handwritten notes. Scientific calculators are allowed.
- Please silence and put away all cell phones and other mobile or noise-making devices.
- Remove all hats, headphones, and watches.
- You have 60 minutes to complete this exam.

Advice

- Read questions carefully before starting. Skip questions that are taking a long time.
- Read all questions first and start where you feel the most confident.
- Relax. You are here to learn. You can do it! 🍀

Question 1: Caching

On a system with **32-bit addresses**, we have a cache with the following parameters:

- 64B cache size
- 16B blocks
- Direct-mapped
- Write-back, write-allocate

A) Calculate the TIO address breakdown (in bits) for the cache described above:

tag: 26 bits index: 2 bits offset: 4 bits

For the following parts, also consider the following code snippet:

```
#define N 8
int A[N];     // &A = 0x100
int B[N];     // &B = 0x140

void copy(){
      for(int i = 0; i < N; i++){
            A[i] = B[i];
      }
}
```

B) Which **set** do `A[0]` and `B[0]` map to in the cache, and what is each element's **block offset**?

`A[0]`: set: `0b00` offset: `0b0000`

`B[0]`: set: `0b00` offset: `0b0000`

C) What is the **miss rate** of the code snippet? Why do we observe this miss rate?

Miss rate *(you may leave your answer as a fraction)*: 100%

Explanation **(1-2 sentences)**: Since the cache is direct-mapped and `A` and `B` are stored exactly one cache size apart, the same elements of `A` and `B` evict each other from the cache in every loop iteration.

D) Three *independent* changes are listed below. For each change, CIRCLE the option that indicates the effect of the change on the **miss rate** from part C:

i) Move `B` to address `0x150`	Increase	No effect	Decrease
ii) Change the block size to 32B	Increase	No effect	Decrease
iii) Make the cache 2-way set associative	Increase	No effect	Decrease

Question 2: Memory Allocation

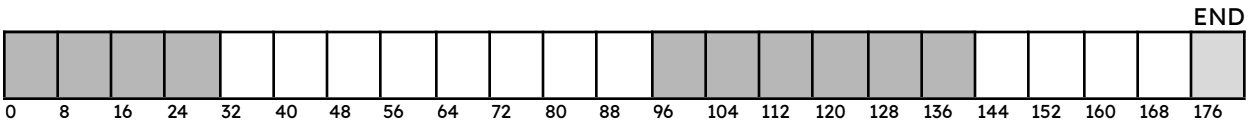
A) Describe a benefit and a drawback of using an **explicit free list** over an **implicit free list** in a dynamic memory allocator (1-2 sentences each):

Benefit (1-2 sentences): An explicit free list allows searching for a free block to completely ignore the headers of allocated blocks (whereas an implicit free list must read them), simplifying searching for a free block.
Drawback (1-2 sentences): Needing to accommodate the free list pointers in the metadata of free blocks is likely to increase the minimum block size of the allocator, meaning we may have more heap fragmentation overall (and thus lower utilization).

B) What is the purpose of **coalescing** free blocks? What problem might arise if coalescing does not happen? Explain in 2-3 sentences.

Explain (2-3 sentences): Coalescing eliminates false fragmentation in the heap (adjacent free blocks in memory). False fragmentation can lead to the allocator missing a span of contiguous space that *could* fulfill an allocation due to it being fragmented into multiple free blocks. This harms heap utilization.

For the remaining parts, consider a dynamic memory allocator on a 64-bit machine with an **explicit free list**, **8-byte boundary tags**, and **16-byte alignment**. Allocated and free heap blocks have both a header and a footer. Answer the following questions given the current state of the heap shown below (allocated blocks are shaded):



C) Supposing that the allocated blocks on the heap were the results of calls to `malloc(12)` and `malloc(24)`, how much **internal** fragmentation is on the heap?

44 bytes

D) Supposing that the most recent request was `malloc(24)`, which of the following allocation strategies will result in `malloc(1)` allocating a block starting at address 144? CIRCLE all that apply.

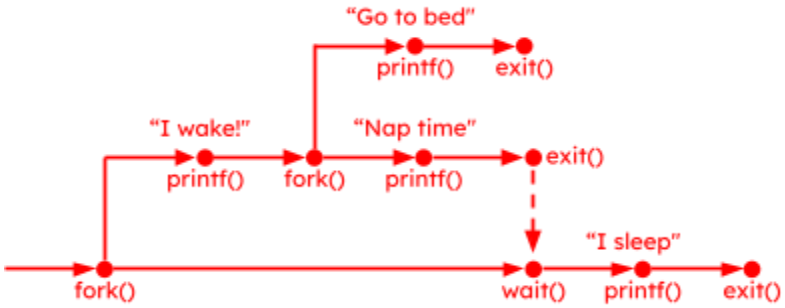
First-fit Next-fit Best-fit

Question 3: Processes

Consider the following code snippet:

```
1    pid_t p1 = fork();
2    if (p1 != 0) {
3        wait(NULL);
4        printf("I sleep\n");
5        exit(0);
6    }
7    printf("I wake!\n");
8    pid_t p2 = fork();
9    if (p2 == 0) {
10       printf("Go to bed\n");
11       exit(0);
12   }
13   printf("Nap time\n");
14   exit(0);
```

A) Draw a **process graph** for the given code snippet.



B) What is **one possible ordering** of the code snippet's `printf` statements?

Possible answers:
"I wake!" -> "Go to bed" -> "Nap time" -> "I sleep"
"I wake!" -> "Nap time" -> "I sleep" -> "Go to bed"
"I wake!" -> "Nap time" -> "Go to bed" -> "I sleep"
"I wake!" must go first, and "I sleep" must come after "Nap time".

C) Is it possible for a *child process created in the snippet* to spend time as a **zombie**? If so, which `fork` call creates it? CIRCLE all that apply:

Not Possible Possible (created on Line 1) Possible (created on Line 8)

Question 4: Virtual Memory

Consider a system with the following properties:

- 16-bit virtual addresses
- 4 KiB of physical memory
- 512 byte pages
- Each Page Table Entry (PTE) contains 1 bit each for Valid, Dirty, Read, Write, and Execute
- An 8-entry, 2-way set associative TLB with LRU replacement policy

A) Calculate the following values:

Physical Page Offset Width:	9 bits	# of PTEs:	2^7 (128)
TLB Tag Bit Width:	5 bits	# of Bits per PTE:	8 bits

B) Suppose we're translating a virtual address to access some data. Are the following statements always true, sometimes true, or never true? CIRCLE one option per row.

1) If there is a valid entry in the TLB for a VPN, then there is a valid PTE for that VPN.	Always	Sometimes	Never
2) If there is a valid PTE for a VPN, then there is a valid entry in the TLB for that VPN.	Always	Sometimes	Never
3) The TLB is invalidated after a context switch.	Always	Sometimes	Never
4) The same virtual addresses from different processes map to different physical addresses.	Always	Sometimes	Never
5) The Physical Page Offset and Virtual Page Offset have the same width.	Always	Sometimes	Never
6) If we have previously accessed a page, then there is a valid PTE for that page.	Always	Sometimes	Never

Question 5: Design Questions

A) We've seen that, using an IEEE 754 single-precision (32-bit) `float`, there are some `int`s that cannot be represented without **rounding**. We want to make a new float so that this does not happen.

Following the same IEEE 754 conventions, what is the **minimum number of bits** each field needs so that every **`unsigned int`** can be represented with a normalized encoding? The S field has been provided for you.

Field	Minimum Number of Bits
S	0
E	6
M	31

Reminder: bias = $2^{w-1}-1$

B) Instead of making physical memory fully associative, we could make it **direct-mapped** and use a virtual address to directly determine its page's physical address. Briefly explain one benefit and one drawback of this idea.

Benefit (1-2 sentences): Individual translations of valid pages are faster (no need to check a TLB or page table due to a simple mapping function) OR no need to store page tables in memory for each process.
Drawback (1-2 sentences): Conflicts between virtual pages stored in physical memory would increase dramatically, increasing the amount of times we need to page in/out from disk. This would slow down translation significantly.

C) Your friend is redesigning the x86-64 register saving conventions and would like to make more registers *callee*-saved instead of *caller*-saved.

Keeping everything else about register saving and calling conventions the same, does writing assembly such that each of the following registers is callee-saved **prevent procedure calls from working correctly**? CIRCLE one in each row:

i) `%r11`	Prevents	Does Not Prevent
ii) `%rax`	Prevents	Does Not Prevent