University of Washington – Computer Science & Engineering

Winter 2026 Instructors: Justin Hsia, Amber Hu 2026-02-09

CSE 333 MIDTERM

Last Name:	Perfect	
First Name:	Perry	
Student ID Number:	1234567	
Name of person to your Left | Right	Sidney Student	Leslie Learner
All work is my own. I had no prior knowledge of the exam content, nor will I share the content with others in CSE 333 who haven't taken it yet. Violation of these terms could result in a failing grade. **(please sign)**		

Do not turn the page until you are told to do so.

Instructions

- This exam contains 10 pages, including this cover page. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- The last page is a reference sheet. *Detach it from the rest of the exam when you are told to do so.*
- The exam is closed book (no laptops, tablets, wearable devices, or calculators). You are allowed one page (US letter, double-sided) of *handwritten* notes.
- Please silence and put away all cell phones and other mobile or noise-making devices. Remove all hats, headphones, and watches.
- You have 70 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.

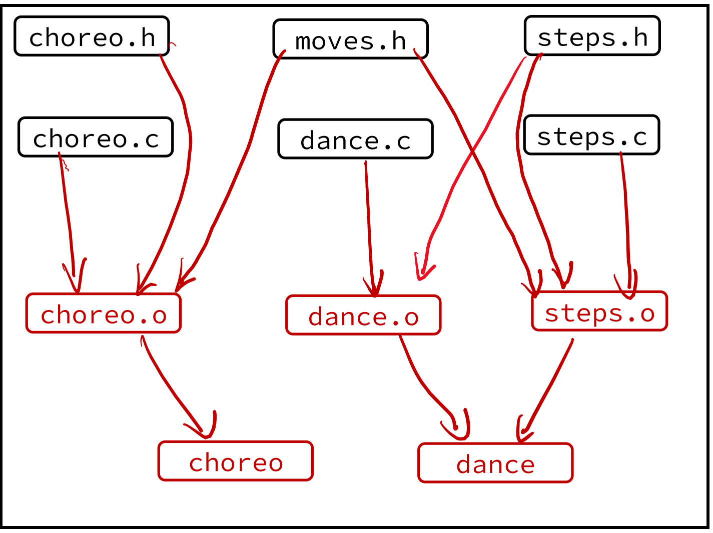
Question	**1**	**2**	**3**	**4**	**5**	**Total**
Possible Points	17	10	25	30	18	**100**

Question 1: MAKE a Song and Dance [17 pts]

Let `CFLAGS = -Wall -g -std=c17`. The automatic variables are defined on the reference sheet.

(A) Complete the corresponding directed acyclic graph for the `Makefile`. [5 pt]

```
dance: dance.o steps.o
   gcc $(CFLAGS) -o dance $^

choreo: choreo.o
   gcc $(CFLAGS) -o choreo $^

steps.o: steps.c steps.h moves.h
   gcc $(CFLAGS) -c $<

choreo.o: choreo.c choreo.h moves.h
   gcc $(CFLAGS) -c $<

dance.o: dance.c steps.h
   gcc $(CFLAGS) -c $<

clean:
   rm -f *.o choreo
```



Arrows connect sources to target and can go either way as long as you are consistent.

(B) Starting with only the source files (`.c` and `.h`) and `Makefile`, what should happen to the following files if we run "`make`" followed by "`make clean`"? Use "**C**" for created, "**CD**" for created and then deleted, and "**U**" for untouched (*i.e.*, unchanged or not created). [4 pt]

`steps.o` _**CD**_ `choreo.o` _**U**_ `dance.o` _**CD**_ `dance` _**C**_

`make` executes the top target (`dance`), which recursively builds `dance.o` and `steps.o`. However, `dance` was omitted from the `rm` command in the `clean` target, so is not deleted.

(C) Starting with only the source files (`.c` and `.h`) and `Makefile`, we run "`make choreo`" then modify `moves.h`. What will happen to the following files when we run "`make choreo`" again? Use "**M**" for modified and "**U**" for untouched. [4 pt]

`steps.o` _**U**_ `choreo.c` _**U**_ `dance.o` _**U**_ `choreo` _**M**_

The `choreo` target only invokes the `choreo.o` target, so `steps.o` and `dance.o` are untouched. Source files (`choreo.c`) aren't modified during compilation.

(D) One of the target recipes/commands is incorrect! Name the fix. [2 pt]

Add `dance` to the `rm` command in the `clean` target.

(E) Based on the good conventions mentioned in this class, suggest one improvement to the `Makefile`. [2 pt]

Add an **`all`** target at the top of the `Makefile` with sources `dance` and `choreo`.

Question 2: Trust the PREPROCESS [10 pts]

Suppose we have the following files (assume `stdio.h` and `stdlib.h` are linked but not shown):

os.h:

```
#ifdef TUX
char* os = "Linux";
#else
char* os = "Windows";
#endif
```

os.c:

```
#include "os.h"
#define TUX 1

int main(int argc, char** argv) {
  printf("%s %d\n", os, TUX);
  return EXIT_SUCCESS;
}
```

(A) The header file is missing a header guard! Following the style guide for this class, what name should we use for the guard macro? [2 pt]

`OS_H_`

(B) Complete the result of `cpp -P os.c` below. [5 pt]

```
char* os = "Windows";

int main(int argc, char** argv) {
  printf("%s %d\n", os, 1);
  return 0;
}
```

The preprocessor processes the file sequentially, meaning `TUX` is defined *after* it processes `os.h`. The `EXIT_SUCCESS` macro is defined in `stdlib.h` and evaluates to `0`.

(C) Is the output of `cpp -P -DTUX os.c` different than `cpp -P os.c` (Part B)? *Briefly* explain why or why not. [3 pt]

Yes. This command will define `TUX` *before* `os.h` is processed, meaning that `char* os = "Linux";` will be included instead.

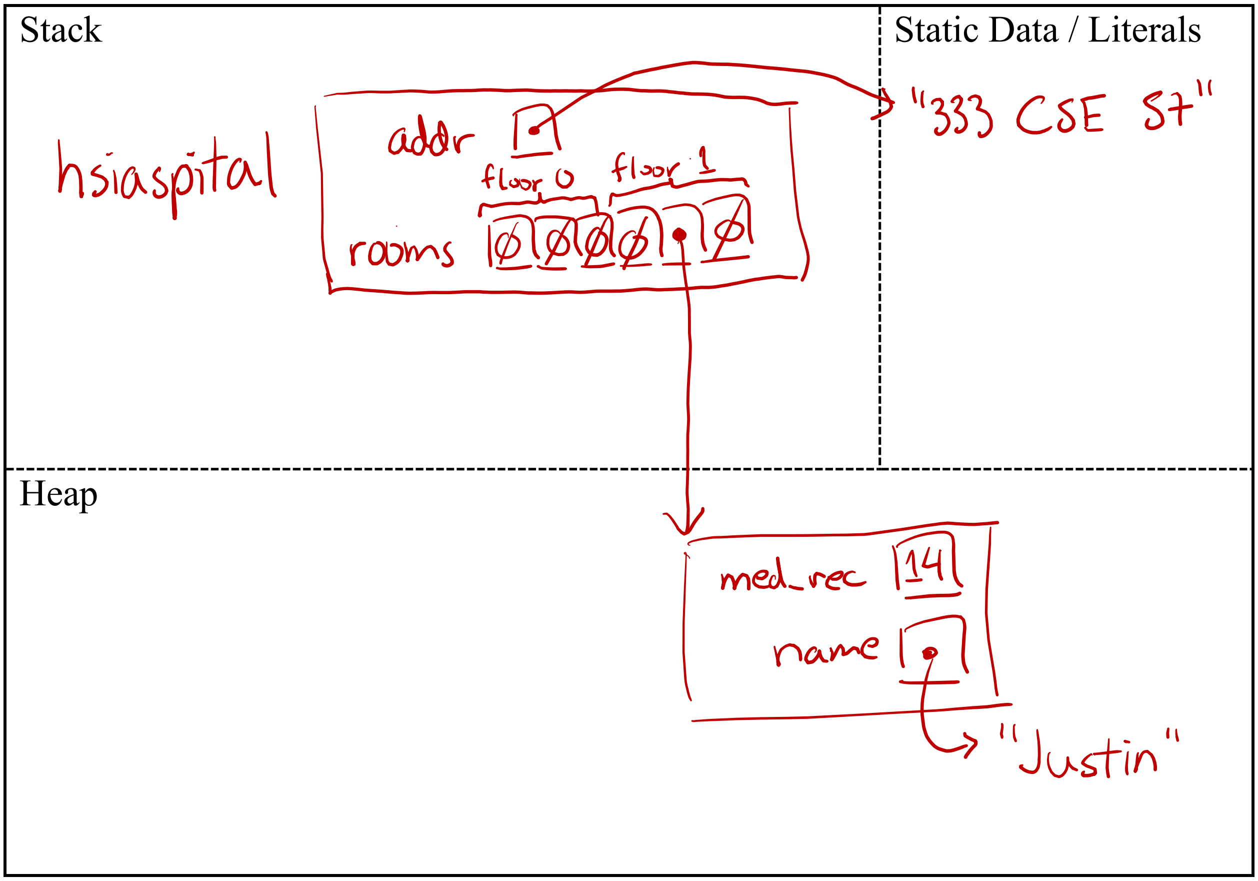
Question 3: Please Have PATIENTS [24 pts]

We're writing C software to help UW Medicine track patients in their hospitals using the following typedef-ed structs:

```
typedef struct patient_st {
  uint32_t med_rec;    // unique medical record number
  char* name;          // patient's name [Heap]
} Patient;
```

```
typedef struct hospital_st {
  char* addr;          // street address (string literal)
  Patient* rooms[NUM_FLOORS][NUM_ROOMS];  // room array (Patients on Heap)
                       // floors and rooms are 0-indexed
} Hospital;
```

(A) Draw a memory diagram for a small `Hospital hsiaspital`, defined in `main`, with address `"333 CSE St"`, `NUM_FLOORS` = 2, and `NUM_ROOMS` = 3. There is a single patient named Justin with medical record `14` in Floor `1`, Room `1` (the 2nd room on the 2nd floor); the rest of the rooms should be `NULL` (Ø). **Place data in the appropriate section of memory. Recall that C is row-major. Character arrays may optionally be written in string literal format** (*e.g.*, "hi"). **Don't forget to include variable and field names.** [8 pt]



(B) Complete the helper function `NewPatient()`, which generates a new patient with a Heap-allocated copy of the provided name, **in good C style**. The medical record number should be one greater than the last assigned number (can start at `0` or `1`). Assume `#include <string.h>` and `#include <stdlib.h>`. For this question, **you can skip error checking `malloc`**. [8 pt]

```
// Returns a pointer to a new patient with a Heap copy of name.
// Skip error checking of malloc.
Patient* NewPatient(char* name) {
  static uint32_t med_rec = 0;

  Patient* patient = (Patient*) malloc(sizeof(Patient));

  // needs to assign med_rec as well as increment it
  patient->med_rec = ++med_rec;  // med_rec++ also accepted

  // needs to account for space for null character
  patient->name = (char*) malloc(strlen(name) + 1);

  // must copy over null character (snprintf, strndup also accepted)
  // strcpy/strdup work but are stylistically worse (security)
  strncpy( patient->name, name, strlen(name) + 1 );

  return patient;
}
```

(C) Assuming all patients are generated using `NewPatient()`, implement `CloseHospital()` below to clean up all of the Heap memory managed by a `Hospital` instance. [4 pt]

```
// Cleans up Hospital's Heap memory
void CloseHospital(Hospital* h) {
  for (size_t i = 0; i < NUM_FLOORS; ++i) {
    for (size_t j = 0; j < NUM_ROOMS; ++j) {

      if (h->rooms[i][j] != NULL) {  // h->rooms[i][j] also works

        free(h->rooms[i][j]->name);  // must free name first

        free(h->rooms[i][j]);        // then free Patient
      }
}
```

(D) We want a function `LocatePatient` to return the <u>floor</u> and <u>room</u> (`size_t`) of a patient with a matching `med_rec` number in a given `Hospital`, if found. Following **good C style** guidelines (no need for `const`), propose a suitable declaration below. **Add commas wherever needed in the parameter list.** Assume `#include <stdbool.h>`. [5 pt]

```
bool LocatePatient(Hospital* hospital, uint32_t med_rec,
                   size_t* floor, size_t* room);
```

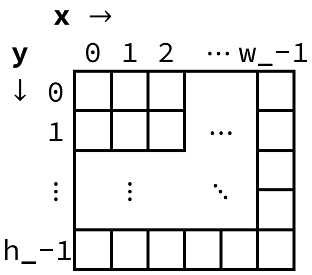
Must return `bool` to indicate not found case (`size_t` is unsigned, so no natural "invalid" value). Input parameters (`hospital`, `med_rec`) should come before output parameters (`floor`, `room`).

Question 4: It's All About How You FRAME It [30 pts]

Abbrev: constructor (**ctor**), copy constructor (**cctor**), assignment (**op=**), destructor (**dtor**).

```
struct Pixel {
  Pixel() : r(0), g(0), b(0) { }
  Pixel(uint8_t r, uint8_t g, uint8_t b) : r(r), g(g), b(b) { }
  uint8_t r, g, b;  // red, green, blue
};  // struct Pixel

class Frame {
 public:
  Frame() : w_(1), h_(1), pixels_(new Pixel[1]) { }
  Frame(const Frame& fr);  // DEEP copies data members
  Frame& operator=(const Frame& rhs);  // DEEP copies
  ...  // other methods mentioned in this question

 private:
  size_t w_;       // width in pixels
  size_t h_;       // height in pixels
  Pixel* pixels_;  // 1-D Heap array representing
                   // width*height pixels (row major)
};  // class Frame
```

Frame coordinates:



(A) Given `Pixel p1` and `Frame f1`, will the following work? Answer "Y" or "N." [4 pt]

`Pixel p2;` _Y_ def ctor   `Frame f2(f1);` _Y_ cctor
`Pixel p3 = p1;` _Y_ synth cctor   `f1 = new Frame;` _N_ new returns ptr

(B) Write out the inline definition of a `Frame` 2-arg ctor that takes `w` and `h` parameters. [3 pt]

```
Frame(size_t w, size_t h) : w_(w), h_(h),
                            pixels_(new Pixel[w*h]) { }
```

(C) We want to add a member function `GetPixel` to get a copy of the `Pixel` at a specified (`x`, `y`) coordinate (see diagram above on the right). Write out the definition in the implementation (`.cc`) file **using good C++ style. No need for bounds/error checking**. [4 pt]

```
Pixel Frame::GetPixel(const size_t x, const size_t y) const {
  return pixels_[w_*y + x];
}
```

Needs to return copy of `Pixel` instead of reference to private data member. Doesn't change `Frame` instance, so should be `const` method.

(D) The `Frame` destructor is defined inline as `~Frame() { delete[] pixels_; }`. Which data member is first *completely destroyed* during this destructor's execution? Circle one: [2 pt]

[✓ `b`]   `g`   `h_`   `pixels_`   `r`   `w_`

`~Frame()`'s body run, which destructs `pixels_` (calls `~Pixel()` on elements); `b` declared last.

(E) `pixels_` points to a 1-D array that represents a row-major 2-D grid on the Heap. Define the `Frame` assignment operator function below. [8 pt]

```
Frame& Frame::operator=(const Frame& rhs) {
  if (this != &rhs) {              // self-check
    w_ = rhs.w_;                   // copy dimensions
    h_ = rhs.h_;
    delete[] pixels_;              // deallocate old array
    pixels_ = new Pixel[w_*h_];    // deep copy into new array
    for(size_t i = 0; i < w_*h_; ++i)
      pixels_[i] = rhs.pixels_[i];
  }
  return *this;                    // enable chaining
}
```

(F) What type of function should the following be? Circle one: [2 pt]

```
Frame BlendFrames(const Frame& f1, const Frame& f2) {
  Frame out = f1;
  for (size_t i = 0; i < out.w_*out.h_; ++i) {
    Pixel& p1 = out.pixels_[i];
    Pixel& p2 = f2.pixels_[i];
    // average pixel values (RHS creates temp object)
    p1 = {(p1.r+p2.r)/2, (p1.g+p2.g)/2, (p1.b+p2.b)/2};
  }
  return out;
}
```

non-friend + member   friend + member   non-friend + non-member   [✓ friend + non-member]

Both objects are in the parameter list, so a non-member function. Accesses private field `pixels_` directly, so needs to be a `friend`.

(G) Assume that the `Frame` cctor (definition not shown) does a *deep* copy of data members. If `f1` and `f2` have width 2 and height 1, how many times are each of the following invoked (count *both* `Frame` and `Pixel` methods) during the execution of **`BlendFrames(f1, f2)`**? Assume no copy elision (*i.e.*, compiled with `-fno-elide-constructors`). [7 pt]

ctor __**6**__   cctor __**2**__   op= __**6**__   dtor __**5**__

Parameters are references, so no copying of arguments. `out` constructed via `Frame` cctor; deep copy will invoke `Pixel`'s default ctor for every pixel (x2) during execution of `new` and then `Pixel`'s op= to copy values over (x2). References in for-loop don't call anything. The {…} statement calls the `Pixel` 3-arg ctor, op=, then dtor (x2 each). Returns a copy, again invoking `Frame` cctor and calling `~Frame()` on `out`, which calls `~Pixel()` on its two `pixel_` array elements. The destruction of the returned copy is not counted here.

Question 5: CASE ClOsEd? [18 pts]

The `ctype.h` library provides the functions `toupper()` and `tolower()`. We are writing a **C program** (*i.e.*, no C++ streams) that prints the contents of a file to `stdout` in AlTeRnAtInG CaPs.

(A) Complete the reading and writing loop in `main` below using the **Cstdio library**. [15 pt]

```
FILE* fd = fopen(argv[1], "r");  // "rb" also accepted
char c, caps = 1;
size_t readlen;
while ((readlen = fread(&c, sizeof(char), 1, fd)) > 0) {
  // check for error
  if ( ferror(fd) != 0 ) {  // ferror(fd) also works
    perror("fread error");  // any suitable message
                            // fprintf(stderr,…) also accepted
    fclose(fd);
    return EXIT_FAILURE;
  }
  if (caps) {
    printf("%c", toupper(c));
  } else {
    printf("%c", tolower(c));
  }
  caps = !caps;  // 1 – caps or equivalent also accepted
}
fclose(fd);
return EXIT_SUCCESS;
```

(B) For this application, would using POSIX `read` (not buffered) instead of `fread` (buffered) be beneficial? *Briefly* explain why or why not. [3 pt]

No. This application reads 1 byte at a time in sequential order, amounting to many small reads. With buffering, these will amount to few actual reads from disk and many accesses to the internal buffer. Without buffering, these will all result in expensive reads from disk.