University of Washington – Computer Science & Engineering

Autumn 2026 Instructor: Justin Hsia 2026-11-02

CSE 333 MIDTERM

Last Name:		
First Name:		
Student ID Number:		
Name of person to your Left | Right		
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	24	29	20	**100**

Question 1: BUILDing Green Spaces [17 pts]

The automatic variables are defined on the reference sheet.

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

```
CFLAGS = -Wall -g -std=c17

all: house park

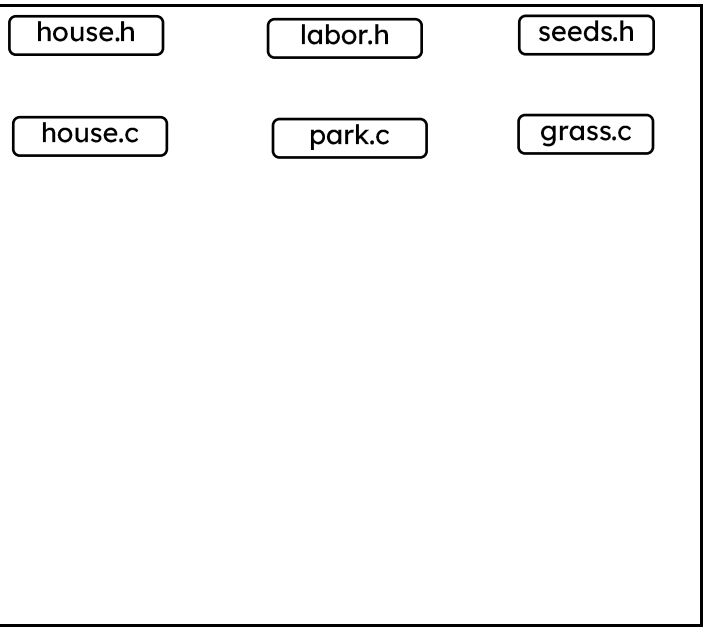
house: house.o grass.o
	gcc $(CFLAGS) -o $@ $^

park: park.o grass.o
	gcc $(CFLAGS) $^

house.o: house.c house.h labor.h
	gcc $(CFLAGS) -c $<

park.o: park.c labor.h
	gcc $(CFLAGS) -c $<

grass.o: grass.c seeds.h
	gcc $(CFLAGS) -c $<
```



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

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

seeds.h ____ park.o ____ grass.o ____ house ____

(D) 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]

park.c ____ grass.o ____ house ____ park ____

(E) Based on the good conventions mentioned in this class, suggest one improvement to the `Makefile`. Only your first suggestion will be graded. [2 pt]

Question 2: Coffee (PRE)PROCESSing [10 pts]

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

cafe.h:

```
#ifdef BEANS
#define BEV "coffee"
#define TEMP(x) (10 * x)
#else
#define BEV "green tea"
#define TEMP(x) 10 + (8 * x)
#endif
```

cafe.c:

```
#include "cafe.h"

int main(int argc, char* argv[]) {
  // Print a message to the user
  printf("Brewing %s at %d degrees\n", BEV, TEMP(12));
  return EXIT_SUCCESS;
}
```

(A) The header file is missing a header guard! According to the CSE 333 style guide, what name should we use for the guard macro? [2 pt]

(B) In 1–2 sentences, *briefly* explain the purpose of header guards. [4 pt]

(C) Complete the result of `cpp -P cafe.c` below. [4 pt]

```
int main(int argc, char* argv[]) {




}
```

Question 3: ARRAY-ngement of Elements [24 pts]

In HW1, you implemented a doubly-linked list data structure in C. Here, we will look at an ArrayList, which maintains an array of limited capacity and pushes elements (here, C-strings) into the next unused index of the array. If the buffer is full, we double its previous capacity and copy all elements from the old buffer to the new one.

```
typedef struct {
  char** arr;  // pointer to heap-allocated buffer of elements
  size_t len;  // number of elements present/pushed
  size_t cap;  // size of the buffer ("capacity")
} ArrayList;
```

You may assume that `stdlib.h`, `stdio.h`, `stdbool.h`, and `string.h` are included. For full credit, your code should comply with CSE 333 style guidelines.

(A) Complete `ArrayList_Push`, which sets the next element of the `ArrayList` to a *dynamically allocated copy* of `str`. For this question, **skip error checking of `malloc` and `realloc`**. See the reference sheet for `string.h` functions and `realloc`. [8 pt]

```
void ArrayList_Push(ArrayList* lst, char* str) {
  if (lst == NULL) return;
  size_t n = strlen(str);
  char* elem = _____________________________________________________;
  _____________________________________________________;
  if (lst->len == lst->cap) {  // if out of space, double size
    lst->cap _____________________________________________________;
    _____________________________________________________;
  }
  _____________________________________________________;
  _____________________________________________________;
}
```

(B) Complete `ArrayList_Free`, which takes in a pointer to a dynamically allocated `ArrayList` and frees the list and all of its dynamically-allocated data. [4 pt]

```
void ArrayList_Free(ArrayList* lst) {
  if (lst == NULL)  return;
  for (size_t i = 0; _________________; ++i) {
    _____________________________________________________;
  }
  _____________________________________________________;
  _____________________________________________________;
}
```

(C) We want a function `ArrayList_Remove`, which takes a pointer to a *dynamically allocated* `ArrayList` named `lst` and index `i` (zero-indexed). It attempts to remove and return element `i` (if valid) from `lst`, and returns whether or not it was successful. Following **good C style** guidelines (no need for `const`), propose a suitable declaration below. **Add commas wherever needed in the parameter list.** [4 pt]

```
______ ArrayList_Remove(_____________________________________________
                        _____________________________________________
                        _____________________________________________);
```

(D) Draw a memory diagram after executing the code snippet below (in `main`). **Place data in the appropriate section of memory. Character arrays may optionally be written in string literal format** (*e.g.*, "hi")**. Don't forget to include variable and field names.** [8 pt]

```
// ArrayList_Allocate dynamically allocates an empty list with capacity 5
ArrayList* al_ptr = ArrayList_Allocate();
ArrayList_Push(al_ptr, "333");
ArrayList al = *al_ptr;
```

Stack	Static Data / Literals
Heap	

Question 4: What's in *Your* WALLET? [29 pts]

Abbreviations: constructor (**ctor**), copy constructor (**cctor**), assignment (**op=**), destructor (**dtor**). "`tm`" is `struct tm` defined in `time.h`. You only need to know that it is a C-struct.

```
struct Card {
  Card() = default;
  Card(uint64_t n, tm e, uint16_t c) : num(n), expir(e), code(c) { }
  uint64_t num;   // credit card number
  tm expir;       // expiration date
  uint16_t code;  // security code
};  // struct Card

class Wallet {
 public:
  Wallet(string name) : owner_(name), size_(0), cards_(nullptr) { }
  Wallet(const Wallet& w) = delete;
  ~Wallet() { delete[] cards_; }
  ...  // other methods mentioned in this question

 private:
  string owner_;  // name of wallet's owner
  size_t size_;   // # of cards in wallet
  Card* cards_;   // 1-D Heap array of credit cards
};  // class Wallet
```

(A) Given `Card c1` and `Wallet w1`, will the following work? Answer "**Y**" or "**N.**" [4 pt]

`Card c2;` _____ `Wallet* w2 = new Wallet;` _____

`c1 = c2;` _____ `Wallet w3(w1);` _____

(B) Write out the inline definition of the `Card` cctor, **using good C++ style**. [3 pt]

(C) Write out the inline definition of `get_owner`, which returns the `Wallet`'s owner, **using good C++ style**. [3 pt]

(D) Which data member is first *completely destroyed* during the execution of `Wallet`'s destructor? Circle one: [2 pt]

cards_ code expir num owner_ size_

(E) Assume `AddCard` below is called when **`size_` is 2**. How many times are each of the following methods invoked during the execution of each line? If executed multiple times, write the *total* invocations associated with that line. For credit, *do not leave blanks*. [6 pt]

```
void Wallet::AddCard(const Card& new_card) {
  Card* old = cards_;
  cards_ = new Card[size_+1];
  for (size_t i = 0; i < size_; ++i)
    cards_[i] = old[i];
  cards_[size_] = new_card;
  delete[] old;
  ++size_;
}
```

	ctor	cctor	op=	dtor
	____	____	____	____
	____	____	____	____
	____	____	____	____
	____	____	____	____
	____	____	____	____
	____	____	____	____

(F) Complete the member function `RemoveCard` below, which attempts to remove and return the card at index `i` (zero-indexed) in the wallet through the output parameter and report whether a removal occurred. [8 pt]

```
bool Wallet::RemoveCard(size_t i, Card* const card) {
  if (i >= size_) return false;
  _____________________________________________;  // set output param
  // alloc new array and copy values over
  _____________________________________________;  // holds old array
  _____________________________________________;  // make new array
  for (size_t j = 0; j < ________; ++j)
    _____________________________________________;
  for (size_t j = ________; j < size_; ++j)
    _____________________________________________;
  _____________________________________________;
  _____________________________________________;
  return true;
}
```

(G) According to the Rule of 3, what additional modification would be recommended for the `Wallet` class? [3 pt]

Question 5: tI ESREVER dna tI pilF [20 pts]

Write a **C program** (*i.e.*, no C++ streams) that prints the contents of a file to `stdout` *in reverse*.

(A) Complete the code in `main` below using the **POSIX I/O library**. For this question, **skip error reporting** (`fprintf`/`perror`). *Assume all necessary headers are included.* [16 pt]

```
________ fd = open(argv[1], __________);
off_t fsize = ...;  // assume the file size in bytes is stored in fsize
char* buf = (char*) malloc(_______________);  // space for file contents
________ len, bytes_read = 0;
while ((len = read(______, _________________, _________________)) != 0) {
  if (len == -1) {
    if (_____________________________________________) {
      ____________________________________________________________;
      ____________________________________________________________;
      ____________________________________________________________;
    }
    continue;
  }
  ____________________________________________________________;
}
// print out what we read IN REVERSE
for (off_t i = __________; __________; __________) {
  if (write(__________, __________, __________) < 1) {
    ...  // assume proper error checking code is here
  }
}
_____________________________________________;
_____________________________________________;
return EXIT_SUCCESS;
```

(B) We mentioned that writing to files descriptors is very expensive. **Describe** a way to use the buffer differently (using POSIX still) to reduce the number of I/O writes in the code above. *Briefly* **explain** why this helps. [4 pt]