University of Washington – Computer Science & Engineering

Winter 2026 Instructors: Justin Hsia, Amber Hu 2026-03-18

CSE 333 FINAL

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 14 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 110 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	6	Total
Possible Points	12	18	20	21	20	20	111

Question 1: Potpourri – Nice to Smell, Hard to Spell [12 pts]

(A) Assume `D` is a derived class of class `B`. If we have `B* ptr = new B;`. If we try to assign **`D* new_ptr = ?_cast<D*>(ptr);`** will the following C++ casts compile ("C") or cause a compiler error ("E")? [4 pt]

`static_cast` **_C_** `dynamic_cast` **_C_**

`const_cast` **_E_** `reinterpret_cast` **_C_**

`static` – allowed, but dangerous. `dynamic` – compiles, but returns nullptr at runtime. `const` – not a const conversion. `reinterpret` – pointer-to-pointer conversion works.

(B) For the following scenarios, indicate whether it will result in a double delete ("D"), memory leak ("M"), or no issue ("N"). [4 pt]

`unique_ptr<int> p1(new int(1));` `p1.release();`	__M__	After release, pointer becomes unmanaged.
`unique_ptr<int> p2(new int(2));` `p2.reset();`	__N__	Reset calls `delete` on pointer.
`shared_ptr<int> p3(new int(3));` `shared_ptr<int> p4(p3.get());`	__D__	Two separate reference counts of 1.

`shared_ptr<int> p5(new int(5));`
`weak_ptr<int> p6(p5);`
`p6.lock();`
	__N__	Temp `shared_ptr` from lock gets destroyed; reference count correctly drops to 0.

(C) Assume we have the following `ThreadMain` function that gets run by 2 threads. Which of the following final values of g are possible ("Y"/"N") after joining all threads? [4 pt]

```
int g = 3;

void* ThreadMain(void* args) {
  g = g + 3;
  g = g - 3;
  return nullptr;
}
```

`-3` **__N__** `0` **__Y__** `6` **__Y__** `9` **__N__**

"L" is short for load, and "S" is short for store. Both threads sequentially execute:
(1) L1, (2) +3, (3) S1, (4) L2, (5) –3, (6) S2.

- `-3`: At least 1 thread stores 6 during S1, so it's not possible to get to –3 from there.
- `0`: Possible ordering is 1:L1=3, 2:L1=3, 1:S1=6, 1:L2=6, 2:S1=6, 1:S2=3, 2:L2=3, 2:S2=0
- `6`: Possible ordering is 1:L1=3, 1:S1=6, 1:L2=6, 2:L1=6, 1:S2=3, 2:S1=9, 2:L2=9, 2:S2=6
- `9`: Max of g at any point is 3+3+3=9, but always end with subtracting 3, so not possible.

Question 2: All in a Day's NET-WORK [18 pts]

(A) What does a **positive (>0) return value** mean for the following network programming functions? Where applicable, you can write "not possible." [8 pt]

`socket``()`:	Succeeded and returned a valid file descriptor (for `connect`/`bind`).
`connect``()`:	Not possible. Only return values are `0` (success) and `-1` (failure).
`accept``()`:	Succeeded and returned a valid file descriptor (for `read`/`write`).
`write``()`:	Succeeded; the number of bytes written (to the NIC buffer).

(B) Fill in the blanks using the word bank below: [6 pt]

`addrinfo`	`AF_INET`	`AF_INET6`	
`in_addr`	`in6_addr`	`in_port_t`	`sa_family_t`
`sockaddr`	`sockaddr_in`	`sockaddr_in6`	`sockaddr_storage`

An **IPv4** address (*just the address*) is stored in a `struct` `__in_addr_____`.

What allows us to differentiate an IPv4 vs. IPv6 address? `__sa_family_t___`.

DNS results are found in a linked list of `struct` `__addrinfo_____`.

(C) Name one piece of metadata that might be included in the header of the packet portion associated with the following network layers. [4 pt]

Network layer: *e.g.*, IP address of source or destination
Transport layer: *e.g.*, sequence #, port, protocol

Question 3: The KEY to Student Success [20 pts]

Student data uses a variety of identifiers, and systems often don't agree on which to use. For example, Canvas uses both UW **netid** (*e.g.*, `hhusky`) and a special **Canvas ID** (*e.g.*, `4012345`), while Gradescope uses **email** (*e.g.*, `hhusky@uw.edu`). Both systems use comma-separated values files (`.csv`), but we need them to talk to each other! We will use the following files (*i.e.*, what the columns in each row are):

- Canvas roster file: "`netid,canvas_id,name,section`"
- Gradescope grade file: "`first_name,last_name,SID,email,section,score,…`"
- Canvas upload file: "`canvas_id,score,comment`"

We will use **C++ STL** to combine a Canvas roster file and a Gradescope grade file to produce a corresponding Canvas upload file. The top of our file is shown below:

```
#include <iostream>
#include <fstream>
#include <map>
#include <string>
#include <boost/algorithm/string.hpp>

using namespace std;
```

(A) Complete the helper function below; *you may not need to fill in all blanks*. `in.getline()` returns `false` on an error. The first two arguments to `boost::split` are `vector<string>& output` and `const string& input`. [4 pt]

```
static const int kBufSize = 1024;
char buf[kBufSize];  // buffer for getline()

// Reads line of CSV and splits by commas into output parameter data.
// Returns true if successful and false on bad read.
bool GetRow(ifstream& in, vector<string>* data) {
  if (in.getline(buf, kBufSize)) {
    __string line(buf)_____________________;
    boost::split(__*data_____________,
                 __line_____________,
                 boost::is_any_of(","),
                 boost::token_compress_off);
    __return true_____________________________;
  } else {
    __return false____________________________;
  }
}
```

Could have done the string conversion of `buf` directly in the 2nd argument of `boost::split` (*i.e.*, `buf` or `string(buf)`), in which case the 1st blank could be left blank.

(B) Complete the following snippet of `main` to read the Canvas roster file `r_file` and generate a mapping from netid to Canvas ID. [6 pt]

- You may assume that all rows in the file are properly formatted and have unique IDs.

```
ifstream r_file(argv[1], ifstream::in);  // open Canvas roster file
__map<string, string>______ canvas_ids;  // map netids to canvas ids
__vector<string>___________ row_data;
while (GetRow(__r_file__, __&row_data__)) {
  // Assumes row is "netid,canvas_id,name,section"
  __canvas_ids[row_data[0]] = row_data[1]_____________________________;
}
```

Keeping Canvas IDs as string simplifies things; no conversion needed as output to file is `string`.

Type of `row_data` can be inferred from `GetRow()` parameter type.

(C) Complete the following snippet of `main` that comes after the code in Part B to read the Gradescope grades file `g_file` and output the corresponding Canvas upload file to `stdout`. In addition to the provided blanks, the body of the while-loop is left for you to fill in. [10 pt]

- You may assume that all rows in the file are properly formatted; however, you may NOT assume that all students in the grades file are found in the Canvas roster (*i.e.*, students drop).
- You can access `kBufSize`, `buf` (global), `canvas_ids`, and `row_data` (Part B).
- Useful `string` member functions to extract a netid from email:
  - `string substr(size_t pos, size_t len);` returns the substring of length `len` starting from position `pos`.
  - `size_t find(char* s);` returns the position of the first occurrence of `s`.

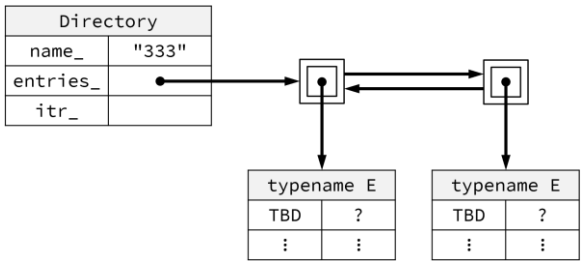
```
ifstream g_file(argv[2], ifstream::in);  // open Gradescope file
__GetRow(g_file, &row_data)__;           // discard header row
while (GetRow(__g_file__, __&row_data__)) {
  // Assumes row is "first_name,last_name,SID,email,section,score,…"
  // Outputs rows "canvas_id,score,comment" (leave comment blank)
  string email(row_data[3]);
  string netid(email.substr(0, email.find("@")));
  // only output if student is in roster
  if (canvas_ids.count(netid) > 0) {
    cout << canvas_ids[netid] << "," << row_data[5] << "," << endl;
  }
}
```

Discarding header row could also be `g_file.getline(buf, kBufSize)`.

Roster check could also be `canvas_ids.find(netid) != canvas_ids.end()`.

Question 4: SMART Revisions [21 pts]

We want to create a C++ wrapper class for directories to abstract away the POSIX library details. For all parts of this question, you may assume the appropriate C++ and POSIX libraries are included, along with `using namespace std;`.




```
struct dirent {
  ino_t          d_ino;
  off_t          d_off;
  unsigned short d_reclen;
  unsigned char  d_type;
  char           d_name[256];
};
```

As seen above, a `Directory` object should have private fields for (1) the name of the directory, (2) a doubly-linked list of directory entries, and (3) an iterator to the "current" entry within the directory. We want to be able to use a wrapper class for directory entries, so we will make this a template class. To account for the unknown directory entry size, we will use an STL container of smart pointers (`unique_ptr`) to reduce the cost of container copying and automatically handle memory management.

(A) Fill in the data member types below for `Directory` to match the above description. [3 pt]

```
template <typename E> class Directory {
 public:
  // declare/define member functions

 private:
  __string_____________________________ name_;
  __list<unique_ptr<E>>________________ entries_;
  __list<unique_ptr<E>>::iterator______ itr_;
};  // class Directory
```

For reasons beyond the scope of this class, the `itr_` type requires the keyword `typename` in front of it to compile properly. We did not grade for this.

(B) Give an inline definition (*i.e.*, within the class declaration in a `.h` file) for `get_size()`, which returns the number of entries in the directory. [3 pt]

```
size_t get_size() const { return entries_.size(); }
```

Other integral types (*e.g.*, `int`) accepted for return type.

(C) Complete the definition of the 1-arg `Directory` constructor on the opposite page, which will open the directory and build the entries list from the directory entries. [5 pt]

- `typename E` is assumed to have a 1-arg constructor that takes a `dirent`.
- The iterator should be initialized to the first entry of the directory.
- Most of the POSIX code is provided.
- You must include an initializer list.

```
template <typename E>
Directory::Directory(string name) : __name_(name)__ {
  DIR* dirp = opendir(name.c_str());
  if (!dirp) {
    cerr << "Error opening directory" << name << endl;
    exit(EXIT_FAILURE);
  }
  dirent* entry;
  while ( (entry = readdir(dirp)) ) {
    __entries_.push_back(unique_ptr<E>(new E(*entry)))_________;
  }
  __itr_ = entries_.begin()_____________________________________;
  __closedir(dirp)______________________________________________;
}
```

In while-loop, `entries_.push_back(make_unique<E>(*entry))` also accepted.

(D) Complete the definition of `ReadDir` below, which mimics the `readdir()` POSIX function. To preserve the privacy of entries, we return a raw pointer to a heap-allocated copy. You may assume that `E` has a default constructor. [6 pt]

```
// Returns ptr to heap copy of current dir entry (nullptr if at
// end), then moves to next dir entry, if possible.
template <typename E> __E*_______________ Directory::ReadDir() {
  if (__itr_ == entries_.end()_____________________________) {
    return nullptr;  // at end of directory
  } else {
    __E* entry = new E_____________________;  // heap allocate
    __*entry = **itr_______________________;  // copy
    __++itr_______________________________;  // advance
    __return entry________________________;  // return
  }
}
```

Heap allocate and copy could have been combined: `E* entry = new E(**itr_);`.

Any reasonable advancement operation was accepted (*e.g.*, `itr_++`), even if operator doesn't exist for `List::iterator` (*e.g.*, `+=`, `+`).

(E) Give two lines of C++ code to (1) create a local `Directory` object for directory `"333"`, using the POSIX `dirent` for directory entries, and (2) call `ReadDir` without leaking memory. [4 pt]

```
Directory<dirent> dir("333");
unique_ptr<dirent> entry(dir.ReadDir());
```

Question 5: Linguistic INHERITANCE [20 pts]

Consider the following **C++ classes**, which compile without errors.

```
class B {
 public:
          void f1() {       cout << "B::f1, "; }
  virtual void f2() { f1(); cout << "B::f2, "; }

 private:
  int b_ = 1969;
};

class C : public B {
 public:
  virtual void f1() {       cout << "C::f1, "; }
  virtual void f3() {       cout << "C::f3, "; }

 private:
  int c_ = 1972, ansi_ = 1989;
};

class CPP : public C {
 public:
          void f2() { f3(); cout << "CPP::f2, "; }

 private:
  int* cpp_ = new int(1985);
};
```

(A) Draw a conceptual/memory diagram of a default-constructed object of `class C` below. Don't show `vptr`'s. [2 pt]

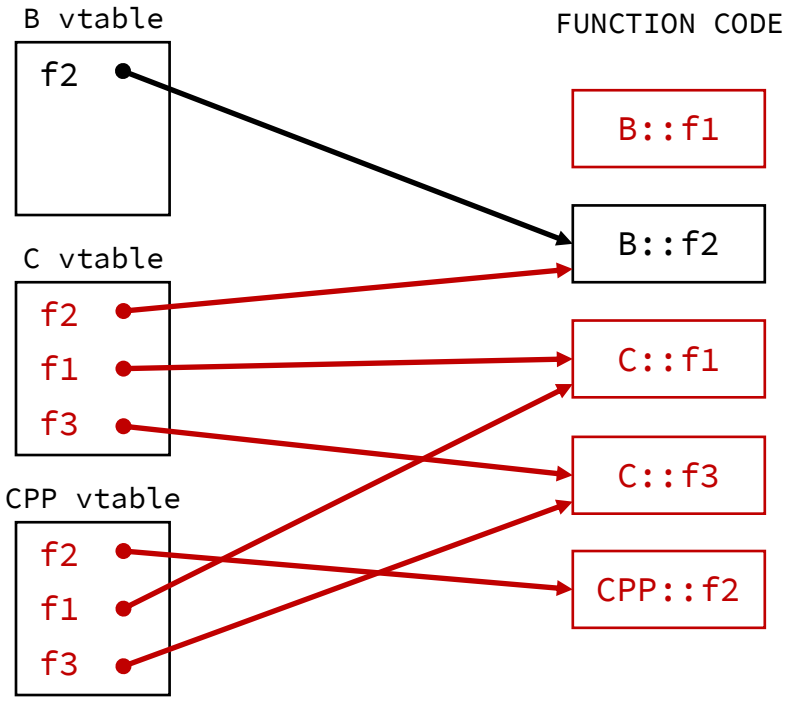
Orientation and sizing doesn't matter

- Class C object
- Class B subobject

(B) The code `B* ptr = new CPP; delete ptr;` causes a memory leak! *Briefly* describe what you would add to the class definitions to fix this issue. [4 pt]

- Add a virtual destructor to `B` to invoke dynamic dispatch in this case. Beyond this question, would want to add a virtual destructor to `C` as well.
- Define a destructor for `CPP` that calls `delete cpp_;`.

(C) Complete the **virtual function table diagram** below by adding the remaining class methods (whether dispatched statically or dynamically) on the right and then drawing the appropriate function pointers from the vtables. *Ordering of the function pointers matters!* One is already included for you. [6 pt]



(D) Assume we have objects and pointers as defined below. For each row of the table, **fill in the result** on the right with the corresponding `stdout` output, "compiler error," or "runtime error." [8 pt]

```
B* bc = new C;
B* bp = new CPP;
C* cp = new CPP;
```

`bc->f1();`	`B::f1,` (static)
`bc->f2();`	`B::f1, B::f2,` (dynamic inherited, `B::f1` is static)
`bc->f3();`	`compiler error` (`B` has no `f3`)
`bp->f2();`	`C::f3, CPP::f2,` (dynamic, `f3` is dynamic inherited)
`cp->f1();`	`C::f1,` (dynamic inherited)
`cp->f2();`	`C::f3, CPP::f2,` (dynamic, `f3` is dynamic inherited)

Question 6: Results Hanging by a THREAD [20 pts]

We are writing a *multi-threaded* C++ program that will total the ballots in an election between Candidates A and B (stored in an `int` array). The top of the file is given below:

```
#include <pthread.h>
#include <iostream>

static const int kNumThreads = 2;
static const int kNumBallots = 100;

// global ballot count indices: 0 = no vote, 1 = A, 2 = B
static int counts[] = {0, 0, 0};
static pthread_mutex_t counts_lock;

struct thd_arg {
  int* ptr;  // ballot array pointer
  int  num;  // number of ballots to count
};
```

(A) Complete the snippet of `main` below that creates the children threads and splits the ballot counting work evenly between them. The function that the children threads will run is called `ThreadMain`. For simplicity, skip error checking and assume `kNumBallots` is a multiple of `kNumThreads`. The passed argument should be *heap-allocated*. [6 pt]

```
int ballots[] = {...};        // ballots array
pthread_t thds[kNumThreads];  // array of thread ids
pthread_mutex_init(&counts_lock, nullptr);
for (int i = 0; i < kNumThreads; ++i) {
  thd_arg* args = __new thd_arg_____________________________;
  args->num = __kNumBallots/kNumThreads_____________________;
  args->ptr = __ballots + i * args->num_____________________;
  pthread_create(__&thds[i], nullptr, ThreadMain, args_____);
}
```

For `args->ptr`, could use `kNumBallots/kNumThreads` in place of `args->num`.

For `pthread_create`, arg1 could be `thds+i`, arg3 could be `&ThreadMain`, and arg4 could have a `static_cast` or `reinterpret_cast` to `void*`, though it's not necessary.

(B) Complete the thread safe function `ThreadMain` on the opposite page. All elements in the ballot array contain the values `0`, `1`, or `2`, which should be tallied in their corresponding indices in `counts[]`. Assume `arg` is heap-allocated. [6 pt]

```
// Reads num ballots from the ballot array and adds the votes
// to the corresponding indices of the counts[] global array.
// Takes ownership of arg. Assumes counts_lock is initialized.
void* ThreadMain(void* arg) {
  thd_arg* a = static_cast<thd_arg*>(arg);
  for (__int i = 0; i < a->num; ++i_____________________________) {
    __pthread_mutex_lock(&counts_lock)_____________________________;
    __counts[a->ptr[i]] += 1_______________________________________;
    __pthread_mutex_unlock(&counts_lock)___________________________;

  }
  __delete a_____________________________________________________;
  return __nullptr_____________;
}
```

(C) Assume `kNumBallots` is `100`, and we have two CPUs available to us. Will the multithreaded code perform better when `kNumThreads` is `2` or `100`? *Briefly* explain. [4 pt]

Better when `kNumThreads` is `2`. With our locking scheme, only one ballot can be counted at a time, so the number of threads is somewhat irrelevant. However, there's a lot more overhead with 100 threads (*e.g.*, context switching, saved state), which would degrade performance.

(D) It turns out that we can improve our multithreaded performance by changing the way we lock in the code! *Briefly* describe how we accomplish this and why it helps our performance. Hint: It will require 3 mutexes. [4 pt]

Change:

Use 3 mutexes, one for each element of `counts[]`.

Not needed in this response, but the easiest way to implement this in code would be to convert `counts_lock` to an array, then change the arguments to `pthread_mutex_lock` and `pthread_mutex_unlock` in `ThreadMain` to `&counts_lock[a->ptr[i]]`.

Why It Helps:

Instead of locking the whole array, now multiple threads can tally ballots simultaneously as long as they are for different responses/candidates.