University of Washington – Computer Science & Engineering
Spring 2026 Instructor: Naomi Alterman 2026-06-10

CSE 333 Final Exam

Last Name:	Solution	
First Name:	Sandeep	
Student ID Number:	333	
Name of person to your Left | Right	David	Alexis
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 24 pages, including this cover page. Show scratch work for partial credit and clearly write your final answers in the boxes and blanks provided.
- If you detach any sheets of this exam, make sure to write your student ID at the top of the page in case they get separated.
- The last sheet of paper is a reference sheet. *Detach it from the rest of the exam when you start.*
- The exam is closed book. 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, smart glasses, watches, and other digital wearables.
- You have 110 minutes to complete this exam.

Advice

- Skim *all* questions first and start where you feel the most confident.
- Skip questions that take a long time.
- Ask questions if you're unsure about a question's phrasing.
- Write assumptions you're making near your answers if you think they're necessary.

Question	1	2	3	4	5	6	7	Total
Possible Points	25	17	25	18	22	20	1	**128**

Q1: STL [25pt]

You're writing a C++ program to take your friends' favorite songs and make a playlist for them. Your friends have given you their song preferences as this STL map going from each friend's name to a vector of songs they like:

```
map<string, vector<string>> p = {
   {"Jonathan", {"Brother Louie", "Forever Young",
                 "Stereo Love", " Thriller"}},
   {"Kelsey",   {"Brother Louie", "Stereo Love",
                 "Blinding Lights", "Cruel Summer",
                 "Thriller"}},
   {"Joe",      {"Thriller", "Billie Jean", "Blinding Lights",
                 "Hotel California"}},
   {"Naomi",    {"Baby Shark", "Baby Shark", "Baby Shark",
                 "Baby Shark", "Baby Shark"}}
};
```

Assume all the code we write is in one file that starts with these lines and you won't need any other #includes:

```
#include <algorithm>
#include <map>
#include <string>
#include <vector>

using namespace std;
```

Also assume that song names will always be provided with consistent capitalization and spelling; there is no need to preprocess the song strings.

Part 1.A [15pt]

Let's start by implementing a function that goes through the favorites list and counts how many people like each song. It should return a map going from song name to the number of people that included it in their favorites. **If a song appears more than once in a person's list, it should still only increase the song's output count by one.** (This way Naomi can't force us to listen to "Baby Shark" again)

```
map<string, int> countPopularity(map<string, vector<string>> &p) {

  map<string, int> v;
  for (auto i: p) {
    vector<string> added;
    for (auto song: i.second) {
      if (find(added.begin(), added.end(), song) != added.end()) {
        continue;
      }
      added.push_back(song);
      if (v.count(song) == 0) {
        v[song] = 0;
      }
      v[song] += 1;
    }
  }
  return v;









}
```

Part 1.B [10pt]

Assuming `countPopularity()` function works, complete the `makePlaylist()` function that returns a vector of song names that **at least half of** our friends are willing to listen to. If there are an odd number of friends, round the count threshold up (eg, 7 friends would require 4 people to list the song in their favorites). The order of the list does not matter.

```
vector<string> makePlaylist(map<string, vector<string>> &p) {



  auto prefs = countPopularity(p);
  float min_cnt = p.size() / 2;
  vector<string> pl;
  for (auto pair : prefs) {
    if (pair.second >= min_cnt) {
      pl.push_back(pair.first);
    }
  }
  return pl;









}
```

Q2: Iterators and Smart Pointers [17pt]

Danni the Cat has asked us to write a C++ program to track where her toys are in the house. We start with this code, which will compile and run assuming we have correct implementations of the functions at the bottom:

```
#include <memory>
#include <iostream>
#include <string>
#include <vector>

using namespace std;

typedef string Toy;
struct Room {
    string name;
    vector<unique_ptr<Toy>> toys;
};

void print_room(const Room &room);
void move_a_toy(Room &from, Room &to);

int main(int argc, const char **argv) {
    Room bed{"Bedroom"};
    Room living{"Living Room"};

    bed.toys.push_back(unique_ptr<Toy>(new Toy("mouse")));
    living.toys.push_back(unique_ptr<Toy>(new Toy("spring")));
    living.toys.push_back(unique_ptr<Toy>(new Toy("feathers")));

    print_room(bed);
    print_room(living);

    move_a_toy(living, bed);

    print_room(bed);
    print_room(living);

    return 0;
}

// ...implementations of print_room() and move_a_toy() go here...
```

Part 2.A: Implement print_room [10pt]

Here's example output from the first call to `print_room(living)` above:

```
Living Room:
  spring
  feathers
```

Fill in the 5 blanks to complete the implementation of `print_room()`:

```
void print_room(const Room &room){

    cout << room.name << ":" << endl;

    for (auto it = __room.toys.begin() ____;

         it != ___room.toys.end()_______;

         ___________it++_____________________) {

        cout << "   " << _____**it___________ << endl;

    }

}
```

Part 2.B: Implement move_a_toy [7pt]

Danni wrote this implementation of `move_a_toy(from, to), which` removes a toy from the "from" room and places it in the "to" room. The code is fully correct, compiles and runs fine:

```
void move_a_toy(Room &from, Room &to) {              /* 1 */
    auto first_toy_it = from.toys.begin();           /* 2 */
    Toy *toy = first_toy_it->release();              /* 3 */
    from.toys.erase(first_toy_it);                   /* 4 */
    cout << "Moving toy " << *toy                    /* 5 */
         << " from " << from.name << " to "          /* 6 */
         << to.name << endl;                         /* 7 */
    to.toys.push_back(unique_ptr<Toy>(toy));         /* 8 */
}
```

She told us she first tried to use `get()` instead of `release()` on line 3 but it caused a segfault. She's not sure why and wants us to explain the problem to her.

Exactly which line number would the segfault occur on if she used `get()` on line 3?

5

Explain what caused the segfault and why using `release()` fixes it:

Getting the raw pointer with get() would cause the unique_pointer to think it was still responsible for freeing that raw pointer's memory. When we erase the unique_pointer on line 4, the unique_pointer will destruct and thus free the memory. When we then *dereference* the raw pointer on line 5 to print it out, we'll be reading from garbage memory. Although it's *possible* this *wouldn't* cause a segfault (it depends on what happened to the bytes at that address in the interim), in Danni's case they got overwritten with something that's not a valid memory address and trying to print out the bytes it pointed to caused a segfault.

Using release() causes the original unique_pointer to "relinquish control" of its memory, and so when it gets destructed on line 4 it doesn't free the data at the raw pointer (and we're able to pass the raw pointer both to cout *and* a new unique_pointer).

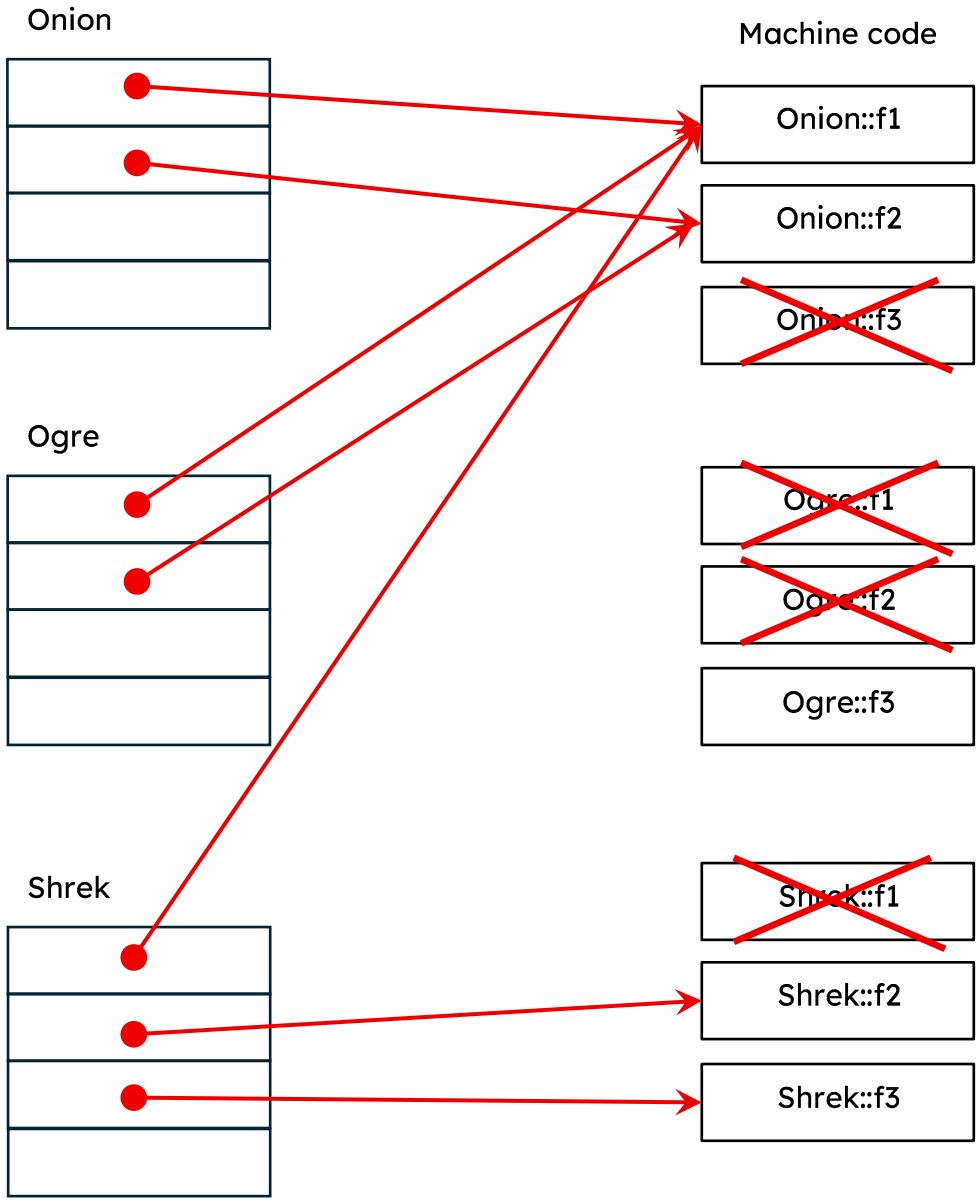
Q3: Inheritance [25pt]

For this question we'll be working with the code below, which compiles successfully:

```
#include <iostream>
using namespace std;

class Onion {
 public:
  virtual void f1() {
    cout << "Onion::f1" << endl;
  }
  virtual void f2() {
    cout << "Onion::f2" << endl;
  }
};

class Ogre : public Onion {
 public:
  void f3() {
    cout << "Ogre::f3" << endl;
  }
};

class Shrek : public Ogre {
 public:
  Shrek() {
    donkey = new int;
    *donkey = 1;
  }
  ~Shrek() {
     delete donkey;
  }
  void f2() {
    cout << "Shrek::f2" << endl;
  }
  virtual void f3() {
    cout << "Shrek::f3" << endl;
  }
 private:
  int *donkey;
};
```

Part 3.A: Mixed Dispatch [15pt]

Fill out the class vtables below by **drawing arrows** from the boxes on the left to their corresponding machine code on the right. **Cross out machine code boxes that don't exist** in the final executable. Make sure the arrows clearly indicate where they point towards. You may not need all blanks.



Part 3.b: Odds [6pt]

Given the following definition, write what each of the following method calls will print out (or if they'll cause a compiler error)

```
Onion* o = new Shrek();
```

i) `o->f1();`

☐ Doesn't compile ☑ Compiles and runs with this output:

Onion::f1

ii) `o->f2();`

☐ Doesn't compile ☑ Compiles and runs with this output:

Shrek::f2

iii) `o->f3();`

☑ Doesn't compile ☐ Compiles and runs with this output:

Part 3.c: Ends [4pt]

The following code causes a memory leak. Explain why and describe how to fix it without changing the `movie()` function itself.

```
void movie() {
    Onion* o = new Shrek();
    o->f1();
    delete o;
}
```

Because neither Onion nor Ogre (Shrek's superclasses) have explicitly declared virtual destructors, they instead have default synthesized **static** destructors.

Since o is of type "Onion pointer", this means that we'll call the (default synthesized) Onion destructor when we "delete o" at the end of movie(). The actual object in memory was of type Shrek, whose constructor allocated an int on the heap. Because the Shrek destructor never gets called, that int doesn't get deleted and will be leaked.

We can fix this by declaring a virtual destructor for Onion or Ogre that just does nothing.

Q4: Networking [18pt]

One of your TAs took pity on you and built a tool to automatically submit your CSE 333 homework instead of requiring you to manually use Git tags. The tool watches your project folder on attu and sends its contents to the course server at the deadline. Magic!!

Out of curiosity you look at the tool's source code (which you can read, because you're a systems programmer!!) and discover it's not doing anything magical. It is built from the same file, directory, and network operations you learned in class. The submission tool reads your homework files, sends them as HTTP requests to the server, and the server sends an HTTP response that it saved them successfully.

Here are the two main header files:

server.h

```
#include <string>
#include <vector>

using std::string;

class SubmissionServer {
 public:
  // Sets up the server so it can wait for
  // student submission clients.
  bool StartHTTPServer(int ai_family, int port, int* listen_fd);

  // Waits for one student client to connect.
  bool WaitForSubmission(int listen_fd,
                         int* client_fd,
                         string* client_addr,
                         uint16_t* client_port);

  // Reads an HTTP-like upload request from the connected client.
  bool ReadUploadRequest(int client_fd, string* request);

  // Sends a response message back to the connected client.
  bool SendUploadResponse(int client_fd,
                          const string& response);
};
```

client.h

```
#include <string>
#include <vector>

using std::string;
using std::vector;

class SubmissionClient {
 public:
  // Connects to the course submission server, possibly specified
  // as a domain name instead of an IP address
  bool ConnectToHTTPServer(const string& hostname,
                           const string& port,
                           int* server_fd);

  // Finds files in the current project directory.
  bool CollectSubmissionFiles(const string& directory,
                              vector<string>* files);

  // Opens one file and reads its contents into memory.
  bool ReadFileContents(const string& filename,
                        string* contents);

  // Sends one file to the course submission server.
  bool SendFileOverHTTP(int server_fd,
                        const string& filename,
                        const string& contents);
};
```

Part 4.A: Trivia [12pt]

Check the box for the best answer for each question.

1. Which POSIX function is most directly responsible for allowing the server to wait for incoming student clients?

- ☐ **A.** `read`
- ☑ **B.** `listen`
- ☐ **C.** `open`
- ☐ **D.** `readdir`

2. Which set of POSIX functions would most likely be used inside `SubmissionClient::`CollectSubmissionFiles?

- [ ] **A.** `socket, connect, write`
- [x] **B.** `opendir, readdir, closedir`
- [ ] **C.** `bind, listen, accept`
- [ ] **D.** `malloc, free, strcpy`

3. Which set of POSIX functions would most likely be used inside `SubmissionClient::`ReadFileContents?

- [x] **A.** `open, read, close`
- [ ] **B.** `socket, bind, listen`
- [ ] **C.** `accept, connect, close`
- [ ] **D.** `getaddrinfo, freeaddrinfo, listen`

4. Which POSIX function will `SubmissionClient` be using to create a socket that can send bytes to the course server?

- [ ] **A.** `accept`
- [x] **B.** `connect`
- [ ] **C.** `bind`
- [ ] **D.** `readdir`

5. Which function is most directly responsible for sending bytes over the network?

- [ ] **A.** `SubmissionClient::`CollectSubmissionFiles
- [ ] **B.** `SubmissionClient::`ReadFileContents
- [x] **C.** `SubmissionClient::`SendFileOverHTTP
- [ ] **D.** `SubmissionServer::`WaitForSubmission

6. Which POSIX function is most likely used by `SubmissionClient::ConnectToHTTPServer` to parse the `hostname` parameter?

- [ ] **A.** `inet_ntop`
- [x] **B.** `getaddrinfo`
- [ ] **C.** `socket`
- [ ] **D.** `htonl`

Part 4.B: HTTP messages [6pt]

When trying to submit your homework, you get this response (and your code doesn't get uploaded):

```
Server Error: parse_uri returned empty string
```

Lousy! You take a look at the server source code to find this function:

```
// Takes the "uri" (Uniform Resource Identifier) from an incoming
// HTTP request and returns a string for the local server-side
// file path where the student's code submission should be saved.
// Returns an empty string if the URI was invalid.
//
// Examples:
//   "/exercise-2.c " -> "./submissions/ex2.c"
//   "/homework-4/main.cc " -> "./submissions/hw4/main.cc"
string parse_uri(const string &uri) {
  if (uri.starts_with("/exercise-")) {
    return "./submissions/ex" + uri.substr(10);
  } else if (uri.starts_with("/homework-")) {
    return "./submissions/hw" + uri.substr(10);
  } else {
    return "";
  }
}
```

During testing, you take a little snoopy look with a spy program to see the bytes being sent from your computer. It starts with:

```
POST /ex5.cc HTTP/1.1

#include <cstdlib>
#include <iostream>

using namespace std;
```

…and then goes on to contain the rest of your submission.

In your own words, why is the server rejecting your submission?

We can see in the first line of the HTTP header that the URI of the file being submitted is "/ex5.cc", but the function being reported in the error will reject requests unless their paths start with "/exercise-" or "/homework-". Our submission filename is not matching the course's naming scheme.

What would be a better error message for the server to return?

Unrecognized submission name "FILENAME". Submissions need to start with "exercise-" or "homework-" followed by the assignment number.

Q5: Templates [22pt]

Consider the following program, which compiles and runs just fine thank you very much:

```
#include <iostream>
#include <string>

template <typename T>
T get_squishy(T a, T b, T c) {
  return a + a + b + b + c + c;
}

int main(int argc, const char **argv) {
    short one = 1;
    short two = 2;
    short three = 3;
    cout << get_squishy(1, 2, 3) << endl;
    cout << get_squishy<short>(1, 2, 3) << endl;
    cout << get_squishy(0x1, 0x2, 0x3) << endl;
    cout << get_squishy(1.1, 2.1, 3.1) << endl;
    cout << get_squishy(one, two, three) << endl;
    cout << get_squishy(std::string("1"),
                        std::string("2"),
                        std::string("3")) << endl;
    return 0;
}
```

Part 5.A: [8pt]

How many different versions of `get_squishy()`'s machine code get generated for the final executable? Which type T was each one was generated for?

Four variants – int, short, float (or double) and std::string

Part 5.B: [6pt]

What does the program output when we run it?

```
12
12
12
12.6
12
112233
```

Part 5.C: [8pt]

Please complete the Amoeba class below so that the following code will compile and run with the output "AdvayAdvayBarboraBarboraCamdenCamden":

```
cout << get_squishy(Amoeba("Advay"), Amoeba("Barbora"),
        Amoeba("Camden")) << endl;
```

You may not need all space provided.

```
class Amoeba {
  public:
    Amoeba(string name) {
      name_ = name;
    }
    string GetName() const {
      return name_;
    }

    Amoeba operator+(Amoeba &rhs) {
      return Amoeba(name_ + rhs.GetName());
    }









  private:
    string name_;



};
ostream &operator<<(ostream &os, const Amoeba &a) {
  return os << a.GetName();
}
```

Q6: Concurrency [20pt]

You are writing a backend service that tracks views on articles. To save time, you ask an AI tool to generate the first version of the code. The good news: it compiles. The bad news: it is classic AI slop. It looks plausible, but it has serious concurrency bugs.

Q6.A: Threading [16pt]

The server appears to use multiple worker threads. When a user views an article, the server calls record_view() on the corresponding Article object. Occasionally, a "statistics" thread calls get_views() to save the view count. Make the Article object thread-safe by filling in the blanks below. You may not need all blanks. You may assume that any standard library "#include"s you need have been made elsewhere in the file.

```
class Article {
  public:
    Article() {

      pthread_mutex_init(&mutex, NULL);

      views = 0;

      ____________________________________________________
    }
    virtual ~Article() {

      pthread_mutex_destroy(&mutex);

      ____________________________________________________
    }
    void record_view() {

      pthread_mutex_lock(&mutex);

      views += 1;

      pthread_mutex_unlock(&mutex);
    }
```

```
    int get_views() {

      pthread_mutex_lock(&mutex); int views = counter.views;
      pthread_mutex_unlock(&mutex); return views;

      return counter.views;

      ____________________________________________________
    }

  private:
    int views;

    pthread_mutex_t mutex;


    ____________________________________________________
};
```

Q6.B: Processes [4pt]

The AI tool gets surprisingly opinionated and says: "You should have fixed the concurrency problems by just using fork() to run the statistics-gathering code instead of using pthreads. You wouldn't have needed to modify the Article class at all to do that."

Huh. Is that true? Briefly explain. (there's more space on the next page if you need it)

It's not – because processes do not share memory, the statistics code will not pick up new views that are recorded after the processes have been forked

Continue your answer from 6.B if you need more space:

nah dogg

Bonus Q: What should we rename C++? [1pt]

Credit will be awarded for literally any answer other than a blank box.

C+Fuss