CSE 333 19su Final Exam

Date: August 23, 2019 | Instructor: Aaron Johnston

First Name:	Sample
Last Name:	Solutions
UW NetID:	ssolutions@uw.edu
Signature: All work is my own. I did not have advance knowledge of the exam and will not distribute knowledge to classmates who haven't taken it yet. Violating these terms may result in a failing grade.	*Sample Solutions*

Do not turn this page until 10:50 am.

Instructions

- This exam contains 15 pages (8 pieces of paper), including this cover page and the reference pages. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided. If you cannot fit your work or solutions in the space provided, you may use the two "extra space" pages at the end, but you must clearly mark that you have done so at the original question and on the extra page.
- The last page is a reference sheet. Please detach it from the rest of the exam.
- The exam is closed book (no laptops, tablets, calculators, or telepathy). You are allowed two pages (US letter, double-sided) of handwritten or typed notes.
- Please silence and put away all cell phones and other mobile or noise-making devices. Remove all hats, sunglasses, and headphones.
- 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. Note the point breakdown below – the questions are of varying difficulty and point value.
- Relax. You are here to learn. ☺

Point Breakdown

Question	1	2	3	4	5	6
Topic	C++ Templates	C++ Inheritance	C++ STL	Networking	Concurrency	(Bonus)
Total Points	18 pts	26 pts	20 pts	22 pts	14 pts	+1 pt

Question 1: C++ Templates [18 pts]

After hearing so many analogies about bad roommates in lecture, you decide to implement a task-tracking system in C++ to help Aaron smooth things over at home. In this question, you will deal with `Task` (a simple struct with a name and a priority) and `TaskList` (a class with a fixed-length array of `Task`s). To make the system generalizable, the user can specify their own priority scale, so `Task` accepts a type parameter for the type used to store priorities. For example, a user might want to rate `Task`s from 1 – 5 and would therefore write `Task<int>`, or they might want to rate `Task`s from 0.0 – 1.0 and would use `Task<double>`.

a) [4 pts] Write the definition for a simple templated struct `Task`. The struct should have two public fields: a name (of type `string`) and a priority (whose type is determined by a type parameter to the template).

```
template <typename P>
struct Task {
  string name;
  P priority;
};
```

b) [8 pts] Write the definition for templated class `TaskList`. The class should:

- Have a single private field, storing an array of `Task` objects.
- Take <u>two</u> template parameters: a <u>type parameter</u> describing the type used for `Task` priorities, and an <u>`int` template parameter</u> describing the number of `Task`s that can be added.
- Declare (but <u>do not implement</u>) a single function called `SetTask` that takes an `int index` (the index of the task to set in the internal array), a `string name` (the task name), and a parameter called `priority` whose type is specified using the class type parameter. `SetTask` should set the given index of the array to a `Task` containing the given name and priority, returning a `bool` to indicate success.

You do not need to declare any constructors, destructors, or other member functions.

```
template <typename P, int N>
class TaskList {
 public:
  bool SetTask(int index, string name, P priority);
 private:
  Task<P> tasks_[N];
};
```

c) [6 pts] Now suppose we change the `TaskList` class so that the priority of the `Task` object is no longer a template parameter for the class, but is instead a template parameter for the `SetTask` function itself.

i) List any changes that would need to be made to the field of `TaskList`. Write "none" if there are none.

We would need to use generic pointers instead of storing the Tasks by value. If we tried storing the Tasks by value in an array, we would run into a compiler error because the compiler has no way of knowing how big each Task would be when initializing the array if that type information is only available during subsequent calls to SetTask.

(Note: we accepted many solutions for this question depending on the implementation given in (a) and (b)).

ii) List any changes that would need to be made to the `SetTask` declaration from part (b). Write "none" if there are none.

We would need to add a template parameter to the SetTask declaration. Specifically, this would be a type parameter representing the type of the priority to be tracked by each Task.

(Note: we accepted many solutions for this question depending on the implementation given in (a) and (b)).

Question 2: Inheritance [26 pts]

Consider the following C++ classes. (Assume we have written `using namespace std;`)

```
class Animal {
 public:
  virtual void Eat() { cout << "Animal::Eat" << endl; }
};
```

```
class Dog : public Animal {
 public:
  void Eat() { cout << "Dog::Eat" << endl; Bark(); }
  void Bark() { cout << "Dog::Bark" << endl; }
};
```

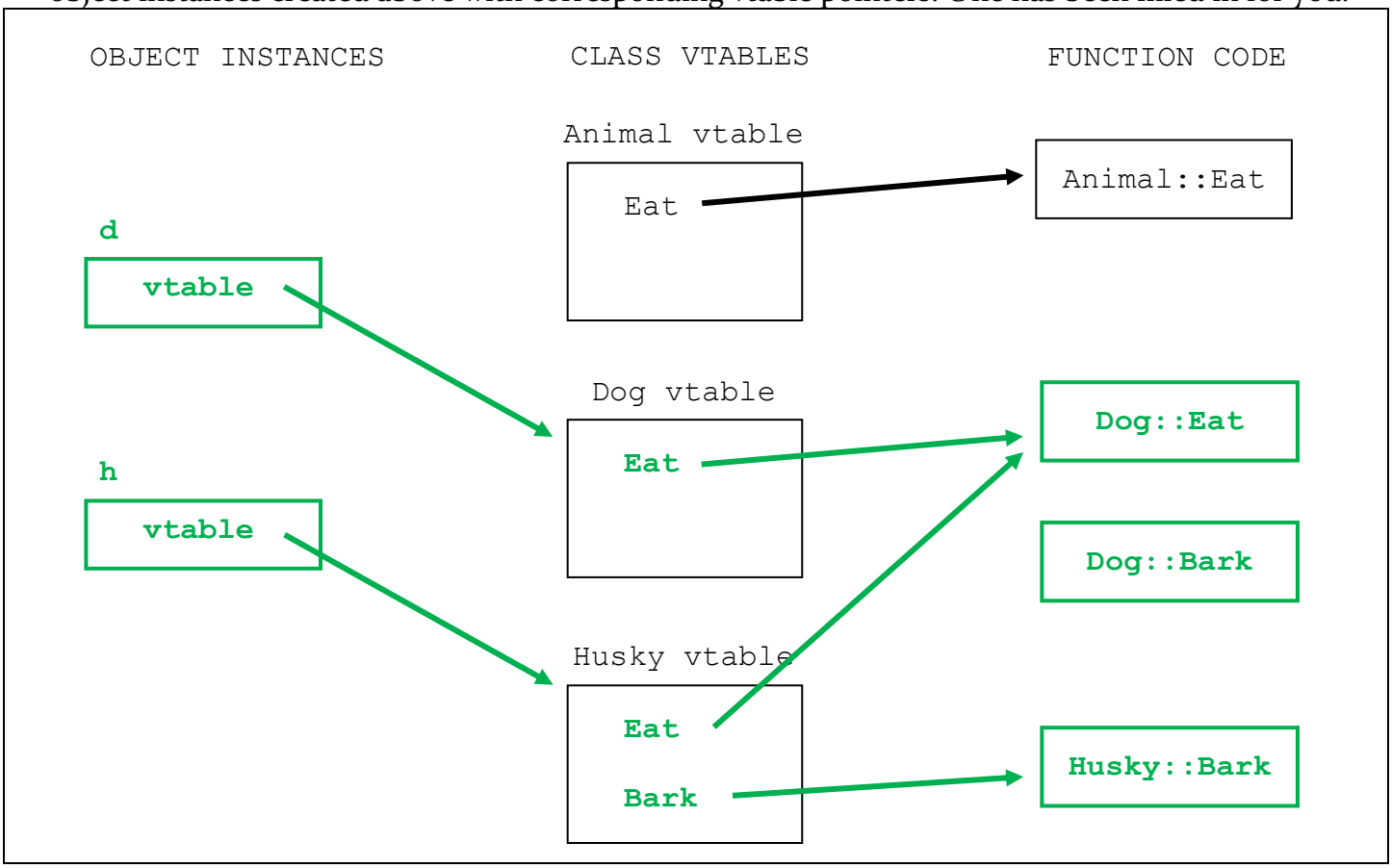
```
class Husky : public Dog {
 public:
  virtual void Bark() { cout << "Husky::Bark" << endl; }
};
```

Suppose we define the following variables:

```
Dog d;
Husky h;

Dog *d2d = &d;
Animal *a2h = &h;
Dog *d2h = &h;
```

a) [8 pts] Complete the following vtable diagram to show the contents of each vtable, then draw the object instances created above with corresponding vtable pointers. One has been filled in for you.



b) [10 pts] Using the variables defined above, for each of the following function calls, write the output that would be printed to `cout`. If the function call would cause a compiler error, write "Compiler Error".

Function Call	Output (or "Compiler Error")
`d2d->Eat()`	`Dog::Eat` `Dog::Bark`
`a2h->Eat()`	`Dog::Eat` `Dog::Bark`
`a2h->Bark()`	`Compiler Error`
`d2h->Eat()`	`Dog::Eat` `Dog::Bark`
`d2h->Bark()`	`Dog::Bark`

c) [8 pts] For each of the following snippets of code, fill in the blank with the most appropriate C++ style cast (Recall that you may choose from `static_cast<>`, `dynamic_cast<>`, `const_cast<>`, and `reinterpret_cast<>`).

```
Husky obj;
Animal *a_ptr = &obj;
Dog *d_ptr = static_cast<Dog *>(a_ptr);
```

```
void Pet(Dog *d_ptr) {
  Husky *h_ptr = dynamic_cast<Husky *>(d_ptr);
  ... // (additional code omitted)
}
```

```
double x = 25.3;
int y = static_cast<int>(x);
```

```
int64_t x = 0x7ffffffffe870; // The address of a C-style string
char *y = reinterpret_cast<char *>(x);
```

Question 3: C++ STL [20 pts]

For this question, you will use the following Date class:

```
class Date {
 public:
  int month;      // Valid values: 1 - 12
  int day;        // Valid values: 1 - 31

  bool operator==(const Date &other) {
    return month == other.month && day == other.day;
  }
}
```

You are building a C++ class called `LectureSchedule` to keep track of upcoming guest lectures. The class should have an STL map called `lectures_` as its (private) field, which maps from a C++ string object (representing the name of the guest lecturer) to an STL vector containing a list of `Date` objects. You can assume that `using namespace std;` has been written at the top of the file.

a) [4 pts] Write the declaration of the `lectures_` field. For this problem, you should store copies of the data itself -- do not store pointers of any kind in the STL containers.

```
map<string, vector<Date>> lectures_;
```

b) [10 pts] Assume any necessary constructors have been implemented for `LectureSchedule`. Write the implementation of a function called `AddLecture` that takes 2 arguments: a C++ string object representing the lecturer name, and a `Date` object (not a pointer) representing the date of a lecture for them. It should add a mapping from that lecturer to that date accordingly, unless the class already stores a mapping from that lecturer to that date (in which case it should do nothing). The function should return `0` if the mapping was added, or `-1` if the mapping already existed.

```
int LectureSchedule::AddLecture(const string &name, const Date &date) {

  vector<Date> &v = lectures_[name];
  auto it = std::find(v.begin(), v.end(), date);
  if (it != v.end()) {
    return -1;
  }
  v.push_back(date);
  return 0;

  // Another equally valid approach:

  for (Date d : lectures_[name]) {
    if (d == date) {
      return -1;
    }
  }
  lectures_[name].push_back(date);
  return 0;
}
```

c) [6 pts] Now, instead of storing a copy of the data itself in the `lectures_` field, suppose we want to store pointers to `Date` objects on the heap. For each of the following possible options, indicate whether it would work as intended, would cause a memory leak, or would not work correctly. Assume that the only changes made would be the field type and adding code to dereference the pointer as appropriate whenever accessing the data (even if that requires multiple steps). In particular, the `LectureSchedule` class has no destructor defined beyond the synthesized one.

`raw pointer (Date *)`	`unique_ptr<Date>`	`shared_ptr<Date>`	`weak_ptr<Date>`
WORKS	**WORKS**	**WORKS**	**WORKS**
MEMORY LEAK	**MEMORY LEAK**	**MEMORY LEAK**	**MEMORY LEAK**
INCORRECT	**INCORRECT**	**INCORRECT**	**INCORRECT**

A raw pointer would cause a memory leak because without a destructor, the class would not be able to clean up the allocated memory it points to. Either a unique_ptr or a shared_ptr would work because both are compatible with STL containers, and this use case is simple enough that there is no significant difference between them. weak_ptr would not work because a weak_ptr can only point to something that is also being pointed at by a shared_ptr – and in this situation, there is only one STL container pointing to the allocated memory.

Question 4: Networking [22 pts]

a) [4 pts] In the sockets API we looked at in class, the `sockaddr_in` struct represents an IPv4 address and `sockaddr_in6` represents an IPv6 address. In this question, for each of the structs listed below, describe what they are typically used for in networking code (i.e. why it's important for them to exist even though there is already `sockaddr_in` and `sockaddr_in6`).

i) `struct sockaddr` (Hint: usually used as `struct sockaddr *`)

Used to refer to a sockaddr struct of an unknown type (either sockaddr_in or sockaddr_in6). This struct has a field that lines up with both possible address structs, so it's possible to use it to determine what family the underlying struct belongs to (and cast the pointer appropriately).

ii) `struct sockaddr_storage`

Used to allocate a space that is guaranteed to be large enough to store either a sockaddr_in or a sockaddr_in6, even in situations where you don't know which you will get ahead of time.

b) [4 pts] For each of the following behaviors, identify what networking layer is most closely thought of as being responsible for handling that behavior.

i) Host A tries to send a long message to Host B in another city, broken up into many packets. A packet in the middle does not arrive, so Host A sends it again.

Transport Layer (Protocol commonly associated with this: TCP)

ii) Host A tries to send a message to Host B, but Host C and Host D are also trying to communicate on the same network, so Host A has to avoid interfering.

Data Link Layer (Protocol commonly associated with this: MAC)

c) [10 pts] The following is incomplete C++ code for a single-threaded server that should continuously read bytes from a single client and print them to `cout`, terminating when the client has closed the connection OR immediately after an `'x'` character has been printed. Since the server is single-threaded, while it is handling the client it is acceptable for it to make all other clients wait (and terminate before handling them). If there is a fatal error, the server should clean up and `exit()` (no error message necessary).

Assume that the **`Listen()`** function performs all the steps necessary (including all error checking!) to create a socket, call **`listen()`** on it, and return its file descriptor. Complete the code below (some lines are fill-in-the-blank, but you will certainly want to add more lines).

```
/* Returns a listening fd. Handles errors appropriately. */
int Listen();

int main(int argc, char **argv) {
  int listen_fd = Listen();

  struct sockaddr_storage caddr;
  socklen_t caddr_len = sizeof(caddr);

  int client_fd = accept(listen_fd,
      reinterpret_cast<struct sockaddr *>(&caddr), &caddr_len);


  if (client_fd == -1) // Note: technically unnecessary since the error
    exit(1);           // would be caught in the read loop below.


  char buf;

  while (1) {
    int res = read(client_fd, &buf, 1);
```

```
    if (res == -1) {
      if (errno == EINTR || errno == EAGAIN)
        continue;
      else
        exit(1);
    } else if (res == 0) {
      break;
    } else {
      cout << buf << endl;
      if (buf == 'x')
        break;
    }
  }

  close(client_fd);
  close(listen_fd);

  return 0;
}
```

d) [4 pts] The original versions of HTTP (including 1.1) were designed to use plain text characters sent over the network instead of alternatives like a binary encoding for the request and response. Describe one advantage of this design decision and one disadvantage. A few words are fine.

Advantage
Interpretable by humans Easy to experiment with and adopt
Disadvantage
Might be less efficient (for some definition of efficient) than a well-packed binary format

Question 5: Concurrency [14 pts]

Consider the following C code, utilizing the pthreads library to run multiple threads concurrently.

```
01  static int x = 0;
02
03  void *increase(void *arg) {
04    while (x < 2) {
05      x++;
06    }
07    return NULL;
08  }
09
10  int main(int argc, char **argv) {
11    pthread_t a, b;
12    pthread_create(&a, NULL, &increase, NULL);
13    pthread_create(&b, NULL, &increase, NULL);
14    pthread_join(a, NULL);
15    pthread_join(b, NULL);
16    return 0;
17  }
```

a) [3 pts] After running the code, is it possible for the final value of x be greater than 2? Circle YES or NO. If you answer YES, describe an interleaving that could cause that.

YES **NO**

Yes. Execution could go like this: Thread A checks x, x is 0, so it enters the while check. Then Thread B runs all the way through the increase function, increasing x to 2. Then Thread A continues, incrementing x again.

b) [3 pts] After running the code, is it possible for the final value of x be less than 2? Circle YES or NO. If you answer YES, describe an interleaving that could cause that.

YES **NO**

It is not possible for the final value of x to be less than 2 because neither thread's while loop can terminate if the value of x is less than 2. No matter how statements between threads are interleaved, the concurrency of the code cannot change the order of execution within each thread.

c) [4 pts] We want to prevent the possibility of having multiple values of x after running the code. To do so, we will add a `pthread_mutex_t` around the critical section of the code – the smallest possible region of code that will prevent a data race. In the space below, write the line numbers of all the lines that are part of the critical section:

Line Numbers:

4, 5, 6

(We also accepted 4, 5 as a solution because 6 is not a significant line of code)

d) [4 pts] Recall that the function `pthread_mutex_init` is used to initialize a `pthread_mutex_t` so it can then be locked and unlocked. If we added locking to the above code using `pthread_mutex_t` locks, should we call `pthread_mutex_init` in the **`main`** function or the **`increase`** function? Circle your answer and briefly explain why:

main **increase**

We want a single instance of a lock. Otherwise, if both threads created their own lock, each thread acquiring the lock would have no effect on the other thread – we need one thread acquiring the lock to prevent the other thread from doing so until the first thread is done.

Question 6: Bonus [+1 pt]

It's been there for you all quarter, through thick and thin, fighting on your behalf against segfaults and only complaining a little when you try to use an uninitialized variable. Now, it's time to show some appreciation for your best friend this quarter, the C/C++ compiler.

Describe (or draw a picture of!) what you would do if you could hang out with your best friend the C/C++ compiler for a day. Making any mark on this page, showing any amount of effort, will earn 1 point of extra credit to be applied to your score on this exam.

Alternatively, you may list Stuart's favorite color to get the point.

`Blue!`

Optional Extra Space for Work/Solutions

If you write any answers here, you MUST clearly indicate on the original problem where to find your answer and clearly indicate on this page which problem is being answered.

Optional Extra Space for Work/Solutions

If you write any answers here, you MUST clearly indicate on the original problem where to find your answer and clearly indicate on this page which problem is being answered.