University of Washington – Computer Science & Engineering

Spring 2018 Instructor: Justin Hsia 2018-06-05

CSE333 FINAL

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 contents nor will I share the contents with others in CSE333 who haven't taken it yet. Violation of these terms could result in a failing grade. **(please sign)**		

Do not turn the page until 12:30.

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. Please detach it from the rest of the exam.
- The exam is closed book (no laptops, tablets, wearable devices, or calculators). You are allowed two pages (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	24	16	16	19	24	16	**115**

Question 1: Potpourri – Nice to Smell, Hard to Spell [24 pts]

(A) Name *two* benefits to utilizing **const correctness** in C++. [4 pt]

(B) Are the following statements about **C++ templates** true or false? Answer T/F. [4 pt]

A template parameter must be a data type. _____

Using a template function instead of function overloading *doesn't* decrease the amount of machine code in your program. _____

Modularizing your code (creating header files and object code for distribution) is the same with and without templates. _____

The `template` keyword does not need to be on the same line as the function or class name. _____

(C) If `class D` is derived from `class B` and we have object instances `B b_obj` and `D d_obj`, **will the following C++ casts cause errors** (either compile-time or run-time)? Answer Y/N. [4 pt]

```
static_cast <D *> (&b_obj)      _____
dynamic_cast <D *> (&b_obj)     _____
dynamic_cast <B *> (&d_obj)     _____
reinterpret_cast <B *> (&d_obj) _____
```

(D) **The Internet** [4 pts]

How many *times* more IPv6 addresses are there compared to IPv4? Answer as a multiple.

Circle one per row:

How many IP addresses can a host be associated with?	**One**	**Multiple**
How many hosts can be associated with an IP address?	**One**	**Multiple**
How many results can a DNS lookup return?	**One**	**Multiple**
How many MAC addresses can a NIC have?	**One**	**Multiple**

(E) For the following **HTTP headers**, circle if they are used in requests or responses and *briefly* explain why that header is important. [4 pt]

`Content-Type`	Used in:	Request	Response
Importance:			
`User-Agent`	Used in:	Request	Response
Importance:			

(F) Complete the table below to compare forking processes and dispatching/spawning threads with `pthread`. [4 pt]

	Threads	**Processes**
Function to create		`fork`
Function for parent to get child's "return value"	`pthread_join`	
Where does the child start code execution?		

Question 2: C++ Standard Template Library [16 pts]

We are investigating a social site like Facebook, where connections are **bidirectional** (*e.g.* a friendship between Justin and Hal means that Justin is Hal's friend *and* Hal is Justin's friend).

(A) Given a user and their friend list (a vector of strings), we want to return all associated friendship links as `pair`s, where the pair (p1, p2) represents that p1 is p2's friend. Implement the function `friendPairs()` below. Hint: `auto` will save you writing. [7 pt]

```
#include <string>
#include <vector>
using namespace std;

vector<pair<string,string>> *friendPairs (string user,
                                          vector<string> friends) {









}
```

(B) We want to print our function results to `stdout` using the **`for_each()`** algorithm, which takes an iterator range and function pointer. Create a function to print out pairs in the format "(p1, p2)" and then fill in the call to **`for_each()`** [7 pt]

```
// define your function here




int main() {
  vector<string> friends({"Adam", "Hal", "Ruth"});      // this works
  auto result = friendPairs(string("Justin"), friends);

  ____________________________________________________________;  // print

  return 0;
}
```

(C) Duplicate friendships eventually show up in our data. Name a container we could move our data into that will automatically remove duplicates for us. [2 pt]

Question 3: Network Programming [16 pts]

(A) Complete the following inequalities for the relative "heights" (higher is "larger") of the following network layers: [2 pt]

Application _____ Physical Network _____ Transport

(B) *Briefly* define the following: [2 pt]

Host byte order:
Network byte order:

(C) Name two *server-side* programming functions that return file descriptors. [2 pt]

(D) *Briefly* explain the effects of **`socket()`** and **`bind()`** on the OS descriptor table. [4 pt]

`socket()`:
`bind()`:

(E) *Briefly* explain why the address family (of type `sa_family_t`) is always the first field in the socket-related structs. [2 pt]

(F) Name one advantage and one disadvantage to using a **non-blocking** socket instead of a **blocking** socket for network communications. [4 pt]

Advantage:
Disadvantage:

Question 4: Smart Pointers and Templates [19 pts]

A shared pointer will only increase its reference count when the copy constructor or assignment operator is invoked (*i.e.* a shared pointer's managed pointer is set from *another* shared pointer).

(A) Complete the main function below we've written to test this fact. Fill in the 4 statements involving shared pointers as well as the blanks in the program output. [6 pt]

```
#include <iostream>
#include <memory>

using namespace std;

int main() {
  // create a shared pointer to the int 3.

  _____________________________________________________;
  cout << "p1.use_count() = " << p1.use_count() << endl;

  // test copy constructor.

  _____________________________________________________;
  cout << "p2.use_count() = " << p2.use_count() << endl;

  // create a shared pointer to the same int that doesn't
  //   increase the reference count.

  _____________________________________________________;
  cout << "p3.use_count() = " << p3.use_count() << endl;

  // test assignment operator to update p3.

  _____________________________________________________;
  cout << "p3.use_count() = " << p3.use_count() << endl;
  return 0;
}
```

Program output:

```
p1.use_count() = _____

p2.use_count() = _____

p3.use_count() = _____

p3.use_count() = _____
```

Let's examine a **singly-linked list**. Assume that all necessary headers are included and we are `using namespace std`.

(B) Define a struct template named `Node` that uses shared pointers for its fields `value` and `next`. Include a *declaration* for a two-argument constructor that takes a shared pointer for the next node and a raw pointer for the value. [6 pt]

(C) Assume we have the function defined below to add a new node at the beginning of the list:

```
template <typename T>
shared_ptr<Node<T>> push(shared_ptr<Node<T>> head, T *val) {
  return shared_ptr<Node<T>>( new Node<T>(head, val) );
}
```

Assume we execute the following lines of code. Draw a memory diagram that includes the reference count of each smart pointer as "ref #" on the corresponding arrow. [7 pt]

```
shared_ptr<Node<int>> head;
head = push<int>(head, new int(2));
head = push<int>(head, new int(4));
shared_ptr<Node<int>> iter(head->next);
```

Question 5: C++ Inheritance [24 pts]

Consider the following C++ classes. The code below causes no compiler errors.

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

class A {
 public:
  virtual void f1() {       cout << "A::f1" << endl; }
          void f2() { f1(); cout << "A::f2" << endl; }
 protected:
  int x_ = 351;
};

class B : public A {
 public:
          void f1() {       cout << "B::f1" << endl; }
  virtual void f3() {       cout << "B::f3" << endl; }
 protected:
  int y_ = 333;
};

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

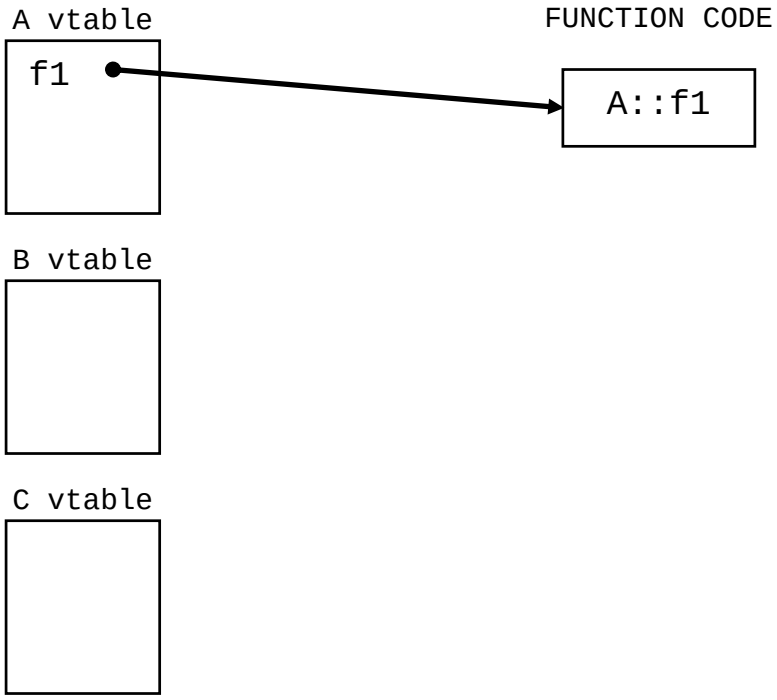
(A) Draw a conceptual diagram of a default-constructed object of `class B` below. Don't show `vptr`'s. [2 pt]

(B) We wish to write constructors for class A and class B to help us initialize our data members. Complete the definitions below: [3 pt]

```
A::A(int x)
```

```
B::B(int x, int y)
```

(C) Complete the **virtual function table diagram** below by adding the remaining class methods 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. [9 pt]



(D) Assume we have objects and pointers as defined in the two lines of code below. Then, for each row of the table below, **fill in the result** on the right, which should either be the corresponding `stdout` output, "compile error," or "runtime error." [10 pt]

```
A a; B b; C c;                          // object instances
A *ap1 = &a; A *ap2 = &b; B *bp1 = &c;  // pointers
```

`ap1->f1();`	
`ap1->f2();`	
`ap2->f1();`	
`ap2->f3();`	
`bp1->f2();`	
`bp1->f3();`	

Question 6: Pthreads [16 pts]

Consider the **C program** below that uses pthreads and compiles and executes without error.

```
   #include <stdio.h>
   #include <pthread.h>
1  int x = 3, ignore;
2  void *task1(void *p) {
3    x -= 1;
4    return NULL;
5  }
6  void *task2(void *p) {
7    x *= 2;
8    return NULL;
9  }
10 int main() {
11   pthread_t t0, t1;
12   ignore = pthread_create(&t0, NULL, &task1, NULL);
13   ignore = pthread_create(&t1, NULL, &task2, NULL);
14   pthread_join(t0, NULL);
15   pthread_join(t1, NULL);
16   printf("%d\n", x);
17   return 0;
18 }
```

(A) List ALL possible printed values of this program if it is run as is. Separate the possible values with commas in the box below. [4 pt]

(B) We will **add lock synchronization** to prevent the threads from interfering with each other. We will add the commands shown in the table below. In the right column, fill in the *half line position(s)* where we will insert the command (*e.g.* "16.5" would mean just before `return 0;` in `main`). [6 pt]

pthread Command	**Insert At Line(s)**
`static pthread_mutex_t lock;`	
`pthread_mutex_init(&lock, NULL);`	
`pthread_mutex_lock(&lock);`	
`pthread_mutex_unlock(&lock);`	

(C) After adding lock synchronization, how *many* printed values are still possible? [2 pt]

(D) Even without lock synchronization, we can guarantee a single possible output by moving a single line from our original code. Indicate which line to move and which half line position to move it to: [2 pt]

Move Line _____ to _____

(E) *Briefly* describe what is problematic about the solution to part D. [2 pt]

THIS PAGE PURPOSELY LEFT BLANK

CSE 333 Reference Sheet (Final)

C Library Header – stdlib.h

```
EXIT_SUCCESS  // success termination code
EXIT_FAILURE  // failure termination code

void   exit (int status);                // terminate calling process
```

Error Library – errno.h

```
errno         // # of the last error, usually checked against defined consts

EAGAIN        // try again
EBADF         // bad file/directory/socket descriptor
EINTR         // interrupted function
EWOULDBLOCK   // operation would block
```

C++ Standard Template Library – vector, list, map, etc.

```
.begin()    // get iterator to beginning (first element)
.end()      // get iterator to end (one past last element)
.size()     // get container size
.erase()    // erase elements

template <class T> class std::vector;
   • .operator[](), .push_back(), .pop_back()
template <class T> class std::list;
   • .push_back(), .pop_back(), .push_front(), .pop_front(), .sort()
template <class Key, class T> class std::map;
   • .operator[](), .insert(), .find(), .count()
template <class T1, class T2> struct std::pair
   • .first, .second
```

C++ STL Algorithms – algorithm

```
std::find()         // returns iterator to element in range that matches val
std::for_each()     // apply function to each element in range
std::min_element()  // get iterator to smallest element in range
std::max_element()  // get iterator to largest element in range
std::sort()         // sorts range into ascending order
```

C++ Smart Pointers Library – memory

```
template <class T> class unique_ptr;
```
- .get(), .reset(), .release()

```
template <class T> class shared_ptr;
```
- .get(), .use_count(), .unique()

```
template <class T> class weak_ptr;
```
- .lock(), .use_count(), .expired()

POSIX Headers – unistd.h, arpa/inet.h, netdb.h

```
ssize_t read (int fd, void* buf, size_t count);
ssize_t write (int fd, const void* buf, size_t count);
int getaddrinfo (const char* hostname, const char* service,
                 const struct addrinfo* hints, struct addrinfo** res);
int socket (int domain, int type, int protocol);
int connect (int fd, const struct sockaddr* addr, socklen_t addrlen);
int bind (int sockfd, const struct sockaddr* addr, socklen_t addrlen);
int listen (int sockfd, int backlog);
int accept (int sockfd, struct sockaddr* addr, socklen_t* addrlen);
```

Pthreads Header – pthread.h

```
pthread_t            // data type to identify a thread
pthread_mutex_t      // data type for a mutex

int pthread_create (pthread_t* thread, const pthread_attr_t* attr,
                    void* (*start_routine)(void*), void* arg);
int pthread_join (pthread_t thread, void** retval);
int pthread_detach (pthread_t thread);

int pthread_mutex_init (pthread_mutex_t* mutex,
                        const pthread_mutexattr_t* attr);
int pthread_mutex_lock (pthread_mutex_t* mutex);
int pthread_mutex_unlock (pthread_mutex_t* mutex);
```