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.

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Points</td>
<td>24</td>
<td>16</td>
<td>16</td>
<td>19</td>
<td>24</td>
<td>16</td>
<td>115</td>
</tr>
</tbody>
</table>
**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]

<table>
<thead>
<tr>
<th>Header</th>
<th>Used in:</th>
<th>Request</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-Type</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Importance:

<table>
<thead>
<tr>
<th>Header</th>
<th>Used in:</th>
<th>Request</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Agent</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Importance:

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

<table>
<thead>
<tr>
<th></th>
<th>Threads</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function to create</td>
<td></td>
<td>fork</td>
</tr>
<tr>
<td>Function for parent to get child’s “return value”</td>
<td>pthread_join</td>
<td></td>
</tr>
<tr>
<td>Where does the child start code execution?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 pairs, 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]

```cpp
#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
```

```cpp
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]

\[
\text{Application} \quad \text{Physical} \quad \text{Network} \quad \text{Transport}
\]

(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]

```cpp
#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]

```cpp
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:

```cpp
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]

```cpp
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);
```

```cpp
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]

```
Question 5: C++ Inheritance  [24 pts]

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

```cpp
#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:
    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]

```cpp
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]

![Virtual Function Table Diagram]

(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]

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ap1-&gt;f1();</code></td>
<td></td>
</tr>
<tr>
<td><code>ap1-&gt;f2();</code></td>
<td></td>
</tr>
<tr>
<td><code>ap2-&gt;f1();</code></td>
<td></td>
</tr>
<tr>
<td><code>ap2-&gt;f3();</code></td>
<td></td>
</tr>
<tr>
<td><code>bp1-&gt;f2();</code></td>
<td></td>
</tr>
<tr>
<td><code>bp1-&gt;f3();</code></td>
<td></td>
</tr>
</tbody>
</table>
Question 6: Pthreads  [16 pts]

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

```c
#include <stdio.h>
#include <pthread.h>

int x = 3, ignore;

void *task1(void *p) {
    x -= 1;
    return NULL;
}

void *task2(void *p) {
    x *= 2;
    return NULL;
}

int main() {
    pthread_t t0, t1;
    ignore = pthread_create(&t0, NULL, &task1, NULL);
    ignore = pthread_create(&t1, NULL, &task2, NULL);
    pthread_join(t0, NULL);
    pthread_join(t1, NULL);
    printf("%d\n", x);
    return 0;
}
```

(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]

<table>
<thead>
<tr>
<th>pthread Command</th>
<th>Insert At Line(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>static pthread_mutex_t lock;</td>
<td></td>
</tr>
<tr>
<td>pthread_mutex_init(&amp;lock, NULL);</td>
<td></td>
</tr>
<tr>
<td>pthread_mutex_lock(&amp;lock);</td>
<td></td>
</tr>
<tr>
<td>pthread_mutex_unlock(&amp;lock);</td>
<td></td>
</tr>
</tbody>
</table>
(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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT_SUCCESS</td>
<td>success termination code</td>
</tr>
<tr>
<td>EXIT_FAILURE</td>
<td>failure termination code</td>
</tr>
<tr>
<td>void exit(int status);</td>
<td>terminate calling process</td>
</tr>
</tbody>
</table>

Error Library – errno.h

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errno</td>
<td># of the last error, usually checked against defined consts</td>
</tr>
<tr>
<td>EAGAIN</td>
<td>try again</td>
</tr>
<tr>
<td>EBADF</td>
<td>bad file/directory/socket descriptor</td>
</tr>
<tr>
<td>EINTR</td>
<td>interrupted function</td>
</tr>
<tr>
<td>EWOULDBLOCK</td>
<td>operation would block</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Class</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>std::vector</td>
<td>.begin(), .end(), .size(), .erase()</td>
</tr>
<tr>
<td>std::list</td>
<td>.push_back(), .pop_back(), .push_front(), .pop_front(), .sort()</td>
</tr>
<tr>
<td>std::map</td>
<td>.operator<a href=""></a>, .insert(), .find(), .count()</td>
</tr>
<tr>
<td>std::pair</td>
<td>.first, .second</td>
</tr>
</tbody>
</table>

C++ STL Algorithms – algorithm

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>std::find()</td>
<td>returns iterator to element in range that matches val</td>
</tr>
<tr>
<td>std::for_each()</td>
<td>apply function to each element in range</td>
</tr>
<tr>
<td>std::min_element()</td>
<td>get iterator to smallest element in range</td>
</tr>
<tr>
<td>std::max_element()</td>
<td>get iterator to largest element in range</td>
</tr>
<tr>
<td>std::sort()</td>
<td>sorts range into ascending order</td>
</tr>
</tbody>
</table>
### C++ Smart Pointers Library – memory

<table>
<thead>
<tr>
<th>Template Class</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique_ptr</td>
<td>.get(), .reset(), .release()</td>
</tr>
<tr>
<td>shared_ptr</td>
<td>.get(), .use_count(), .unique()</td>
</tr>
<tr>
<td>weak_ptr</td>
<td>.lock(), .use_count(), .expired()</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssize_t read</td>
<td>(int fd, void* buf, size_t count)</td>
</tr>
<tr>
<td>ssize_t write</td>
<td>(int fd, const void* buf, size_t count)</td>
</tr>
<tr>
<td>int getaddrinfo</td>
<td>(const char* hostname, const char* service, const struct addrinfo* hints, struct addrinfo** res)</td>
</tr>
<tr>
<td>int socket</td>
<td>(int domain, int type, int protocol)</td>
</tr>
<tr>
<td>int connect</td>
<td>(int fd, const struct sockaddr* addr, socklen_t addrlen)</td>
</tr>
<tr>
<td>int bind</td>
<td>(int sockfd, const struct sockaddr* addr, socklen_t addrlen)</td>
</tr>
<tr>
<td>int listen</td>
<td>(int sockfd, int backlog)</td>
</tr>
<tr>
<td>int accept</td>
<td>(int sockfd, struct sockaddr* addr, socklen_t* addrlen)</td>
</tr>
</tbody>
</table>

### Pthreads Header – pthread.h

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pthread_t</td>
<td>// data type to identify a thread</td>
</tr>
<tr>
<td>pthread_mutex_t</td>
<td>// data type for a mutex</td>
</tr>
<tr>
<td>int pthread_create</td>
<td>(pthread_t* thread, const pthread_attr_t* attr, void* (<em>start_routine)(void</em>), void* arg)</td>
</tr>
<tr>
<td>int pthread_join</td>
<td>(pthread_t thread, void** retval)</td>
</tr>
<tr>
<td>int pthread_detach</td>
<td>(pthread_t thread)</td>
</tr>
<tr>
<td>int pthread_mutex_init</td>
<td>(pthread_mutex_t* mutex, const pthread_mutexattr_t* attr)</td>
</tr>
<tr>
<td>int pthread_mutex_lock</td>
<td>(pthread_mutex_t* mutex)</td>
</tr>
<tr>
<td>int pthread_mutex_unlock</td>
<td>(pthread_mutex_t* mutex)</td>
</tr>
</tbody>
</table>