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>15</td>
<td>18</td>
<td>24</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>117</td>
</tr>
</tbody>
</table>
Question 1: Potpourri – Nice to Smell, Hard to Spell  [15 pts]

(A) Will the following lines of C++ code compile? Briefly explain. [3 pt]

<table>
<thead>
<tr>
<th></th>
<th>Y / N</th>
</tr>
</thead>
<tbody>
<tr>
<td>int &amp;* a1;</td>
<td></td>
</tr>
<tr>
<td>int ** a2;</td>
<td></td>
</tr>
</tbody>
</table>

(B) Circle the outcome of the following lines of code with const int* cip. [3 pt]

<table>
<thead>
<tr>
<th></th>
<th>Compiles / Invalid syntax / Violates constness</th>
</tr>
</thead>
<tbody>
<tr>
<td>const int** b1 = &amp;cip;</td>
<td></td>
</tr>
<tr>
<td>int* const * b2 = &amp;cip;</td>
<td></td>
</tr>
<tr>
<td>int** const b3 = &amp;cip;</td>
<td></td>
</tr>
</tbody>
</table>

(C) The following C casts compile and run without errors. Which type of cast is the most appropriate equivalent in C++ code? [3 pt]

```
const int i = (int) 4.2; ___________________________
int* iptr = (int*) &i; ___________________________
float* fptr = (float*) iptr; ___________________
```

(D) Briefly describe what slicing is and why it is a problem with STL containers. [4 pts]

<table>
<thead>
<tr>
<th>Slicing:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containers:</td>
</tr>
</tbody>
</table>

(E) Briefly explain whether the Ethernet payload or IP payload takes up a larger percentage of the overall packet sent over the network. [2 pt]

<table>
<thead>
<tr>
<th>(circle one)</th>
<th>Explain:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet / IP</td>
<td></td>
</tr>
</tbody>
</table>
Question 2: A NETWORKING Event For You [18 pts]

(A) What does a return value of 0 mean for the following network programming functions? [6 pt]

\begin{itemize}
  \item \texttt{getaddrinfo()}: \\
  \texttt{socket()}: \\
  \texttt{bind()}: \\
  \texttt{read}():
\end{itemize}

(B) Layers upon layers of structs! Fill in the blanks using the word bank below: [6 pt]

\begin{tabular}{cccc}
  addrinfo & AF_INET & AF_INET6 & in_addr \\
  in6_addr & in_port_t & sa_family_t & ss_family_t \\
  sockaddr & sockaddr_in & sockaddr_in6 & sockaddr_storage \\
\end{tabular}

An IPv6 address (just the address) is stored in a struct 

The above IPv6 address, plus other info, is stored inside a struct 

The above struct fits inside a more general struct 

(C) We are writing single-threaded server-side networking code that doesn’t use file I/O. Using comments/pseudocode, describe what we should do in order if we detect an unrecoverable error during write() based on the suggested style used in this course. You should write out something for each line of code you would expect to write, but it does not need to be properly formatted (e.g. just a description is fine). [6 pt]
**Question 3**: Let’s Give Your PHONE Signal a Boost  [24 pts]

Before smart phones, mobile phones used a predictive text system called **T9**, based on the mapping of a single numpad key to any of the corresponding letters shown in the image to the right. Note that the ‘1’, ‘*’, and ‘#’ keys won’t be used and that ‘0’ corresponds to [Space].

**Example**: a user would type ‘8’, then ‘4’, then ‘3’ to get the word “the”, though it could also predict longer words like “they” or “there”.

We will use **C++ STL** to generate our T9 predictive dictionary! The top of our file is shown below so that you are aware of what is globally available:

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <map>
using namespace std;
```

First, we will convert everything in a predefined index (*e.g.* imported from a file) of predictable words into the corresponding key presses (*e.g.* "the"→"843", "hello"→"43556").

(A) We will use a global map from lowercase letters to keys and a function to initialize this map. Complete just the requested portion below of that function: [4 pt]

```cpp
map<char, char> letters_to_keys;
void InitLettersMap() {
    __________________________; // key mapping for 'a'
    ... // assume all other key mappings happen here
    __________________________; // key mapping for 'z'
}
```

(B) Complete the function below to convert a string to lowercase *in place* (note the parameter type). **Hint**: use the char tolower(char) function. [3 pt]

```cpp
void StrToLower(string * const input) {
```
We want the strings to be predictable from any of the prefixes (i.e. substring starting at the beginning) of their keyed versions. Example: "the" is predictable from "8", "84", and "843".

(C) Complete the function to add a mapping from each prefix to the lowercase string. You may find the string member function string substr(size_t pos, size_t len); useful, which returns the substring of length len starting from position pos. [10 pt]

```cpp
map<______, vector<______>> predictions; // global prediction map
________ AddPrefixesToPredictions( ___________ word) {
    // lowercase the word
}
```

(D) Complete the function below to print out the contents of predictions. For example, if we’ve added "a" and "ax", it should print out the following (note the formatting): [7 pt]

```cpp
void PrintPredictions() {

2 : a, ax,
29 : ax,
}
```
Question 4: Get SMART With Your POINTERS  [18 pts]

We are building a template for a two-level array, which is an array of pointers to row arrays. This is how Java implements 2D arrays! We will use smart pointers in order to help with the memory cleanup. Below is an attempt at a definition of our class, where \( h \) is the “height/rows” and \( w \) is the “width/columns” of our two-level array:

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

using namespace std;

template <typename T, int h, int w> struct TwoLevelArray {
public:
    TwoLevelArray(); // def ctor
    TwoLevelArray& operator=(const TwoLevelArray& rhs); // op=
    T* operator[](size_t idx); // op[]

private:
    unique_ptr<T[]> level_one_[h]; // array of unique_ptrs to arrays
};
```

(A) Complete the definition of the default constructor below. Note: you are not allowed to try to add an initialization list. [3 pt]

```cpp
template <typename T, int h, int w>
TwoLevelArray<T, h, w>::TwoLevelArray() {
}
```

(B) The struct above uses the synthesized copy constructor. Will this work? Briefly explain your response. [3 pt]

(Circle one) Works? Yes No
Explain:
(C) Complete the definition of the assignment operator below using good style. [6 pt]

```
template <typename T, int h, int w>
TwoLevelArray<T, h, w>&
TwoLevelArray<T, h, w>::operator=(const TwoLevelArray &rhs) {
    for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
            level_one_[i][j] = rhs.level_one_[i][j];
        }
    }
    return *this;
}
```

(D) Complete the definition of the subscript operator so that it will return the address of the requested row. Hint: pay attention to the return type. [2 pt]

```
template <typename T, int h, int w>
T* TwoLevelArray<T, h, w>::operator[](size_t idx) {
    // Implementation
}
```

(E) What happens when we try to execute the following `main` function (circle one)? If there is an issue, give a brief explanation. If it works fine, how many unique pointers are constructed? [4 pt]

```
int main() {
    TwoLevelArray<int, 2, 3> ar;
    cout << "ar[0] = " << ar[0] << endl;
    ar[0][1] = 333;
    TwoLevelArray<int, 2, 3> ar_copy;
    ar_copy = ar;
    return EXIT_SUCCESS;
}
```

<table>
<thead>
<tr>
<th>Compiler Error</th>
<th>Double Delete</th>
<th>Memory Leak</th>
<th>Segmentation Fault</th>
<th>Works Fine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Explanation/#: 
Question 5: You Are Entitled To Your INHERITANCE  [20 pts]

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

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

class A {
public:
    void f1() { f2(); cout << "A::f1, "; }
    virtual void f2() { cout << "A::f2, "; }
private:
    int m_ = 333;
};
class B : public A {
public:
    void f1() { cout << "B::f1, "; }
    virtual void f3() { cout << "B::f3, "; }
private:
    int n_ = 451;
};
class C : public B {
public:
    virtual void f1() { cout << "C::f1, "; }
    void f2() { f3(); cout << "C::f2, "; }
};
```

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

(B) List out in order all functions (synthesized or defined) that are called during the execution of the code below. Make sure to use the full Class::Function names. [3 pt]

```c++
A* ap = new B;
delete ap;
```
(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. [7 pt]

(D) Assume we have objects and pointers as defined in the five 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, “compiler error,” or “runtime error.” [8 pt]

B b;       // object instances
C c;
A *ap1 = &c;  // pointers
B *bp1 = &b;
B *bp2 = &c;

<table>
<thead>
<tr>
<th></th>
<th>FUNCTION CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>bp1-&gt;f1();</td>
<td></td>
</tr>
<tr>
<td>bp1-&gt;f2();</td>
<td></td>
</tr>
<tr>
<td>bp2-&gt;f2();</td>
<td></td>
</tr>
<tr>
<td>bp2-&gt;f3();</td>
<td></td>
</tr>
<tr>
<td>ap1-&gt;f1();</td>
<td></td>
</tr>
<tr>
<td>ap1-&gt;f3();</td>
<td></td>
</tr>
</tbody>
</table>
Question 6: Staying A-THREAD Of The Game  [22 pts]

We are going to write a multithreaded C program that will count all of the instances of a specified integer in an integer array. The top of the file is given below:

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

#define NUM_THREADS 4
#define AR_SIZE 100
#define COUNT_ME 2
static int global_count = 0, ignore;
static pthread_mutex_t count_lock;

typedef struct {
    int *ar_ptr; // array pointer
    int elements; // number of consecutive elements to check
} thd_arg;

(A) Complete the function below that the children threads will run. Assume that the passed values are found on the Heap. [8 pt]

```c
// Adds the # of occurrences of COUNT_ME in a portion // of the array to global_count and cleans up. // Assumes count_lock is initialized already.
void *ThreadMain(void *arg) {
    thd_arg *a = ____________________________;
    int local_count = 0;
    for (int i = 0; i < ___________; i++) {
        if (__________________________) {
            local_count++;
        }
    }
    ____________________________;
    ____________________________;
    ____________________________;
    ____________________________;
    return __________;
}
```
How would you expect this multithreaded code to perform compared to a sequential version of the code in the following scenarios? Briefly explain your choices. [4 pt]

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Faster</th>
<th>About the Same</th>
<th>Slower</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single CPU:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple CPUs:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Explanation:

It turns out that we can avoid using a mutex altogether if we make use of the return value from `ThreadMain`!

What changes need to be made to the following lines in `ThreadMain`? (other minor changes are needed but are not part of this question) [4 pt]

Line 2: ______ local_count = ____________________________;

Line 12: return __________;

Complete the portion of `main` below that will sum up the local counts of each thread. Here, we allow you to skip error checking by storing into the `ignore` variable. [6 pt]

```c
pthread_t thds[NUM_THREADS]; // array of thread ids
________ retval; // declare a needed variable
...
// threads created here

// Wait for all child threads to finish and add their counts to global_count.
for (int i = 0; i < NUM_THREADS; i++) {
    ______________;
    ______________;
    ______________;
}
```
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 element (pass 1 iterator) or range (pass 2 iterators)

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

```cpp
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

```c
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

```c
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);
int pthread_mutex_destroy (pthread_mutex_t* mutex);
```