Problem 1: Multiple Choice Madness (24 points)

Circle exactly one answer for each of the following questions:

i. It is possible for C code to determine the endian-ness of the underlying CPU.

a) true b) false

ii. In C, a pointer is a variable that contains an address. If you add 2 to a pointer, then:

a) the resulting value is the address plus 2

b) the resulting value depends on what value the pointer points to

c) the resulting value depends on the type of the pointer

d) a segmentation fault is thrown

iii. When you pass a struct as an argument to a C function, then:

a) the struct is passed by value (i.e., a copy of the struct is made, including copying each field in the struct))

b) the struct is passed by reference (i.e., a pointer to the struct is passed)

c) a compiler error is thrown, since you cannot pass structs as arguments

d) what happens depends on the type of fields in the struct

iv. When you pass an array as an argument to a C function, then:

a) the array elements are passed by value (i.e., a copy of the array is made, including copying each element of the array)

b) since arrays are really just pointers, a pointer to the first element of the array is passed and no array elements are copied

c) a compiler error is thrown, since you cannot pass arrays as arguments

d) what happens depends on the type of the array

v. The purpose of a header guard is to:

a) prevent more than one .c file from including a particular .h file

b) prevent the header file from being included indirectly, as a side-effect of including some other .h file that includes it

c) document the contents and purpose of the header file

d) prevent the header file from being included twice, directly or indirectly

vi. A C++ reference:

a) serves as an alternative name for an object or variable (i.e., is an alias)

b) serves as a pointer to an object or variable

c) cannot be used as a parameter of a function

d) cannot be passed as an argument to a function

vii. What does "const" in the following code imply?

```
void foo (const int *x) {   …   }
```

a) the value of the pointer "x" cannot be changed inside the function foo

b) the function foo cannot have any side-effects

c) nothing; const in this case has no effect

d) the value that the pointer "x" points to cannot be changed inside the function foo

viii. What does "const" in the following code imply?

```
void Foo::bar (int *x) {   …   } const;
```

a) the method bar( ) cannot mutate any of its parameters

b) the method bar( ) cannot have any side-effects at all

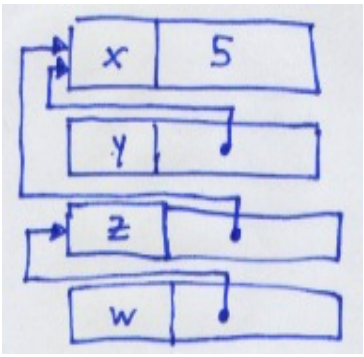
c) the method bar( ) cannot mutate any of Foo's state

d) the method bar( ) can only invoke const-y functions and methods

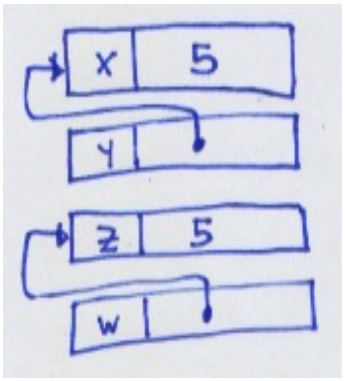
ix. Which of the following box and arrow diagrams correctly represents the following code?

```
int x = 5;
int *y = &x;
int &z = x;
int *w = &z;
```

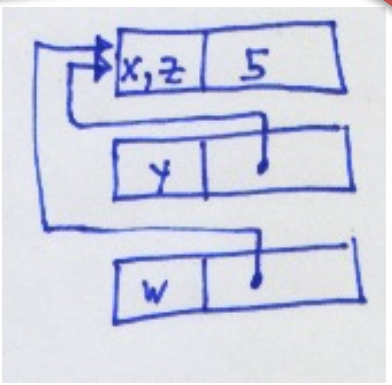
a)



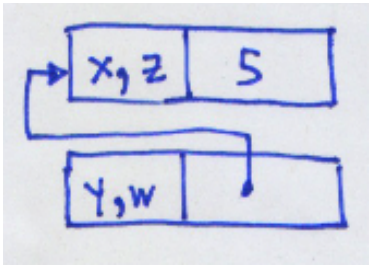
b)



c)



d



x. The destructor of an object that is heap-allocated:

a) is invoked when the function in which it is allocated returns

b) is never invoked

c) must be invoked manually

d) is invoked when somebody uses "delete" to deallocate the object

xi. A vtable:

a) exists for each class, and contains a function pointer for each method in the class

b) exists for each class, and contains a function pointer for each virtual method in the class

c) exists for each object instance, and contains a function pointer for each method in the object's class

d) exists for each object instance, and contains a function pointer for each virtual method in the object's class

xii. Slicing occurs when:

a) the value of a derived class is assigned to an instance of a base class

b) a pointer to a derived class is cast to, and assigned to, a pointer to a base class

c) an N-element array is assigned to an M-element array, where M<N

d) an element is deleted from a std::vector

CSE 333 ~~Final~~ 2nd Midterm Exam August 18, 2017

Sample Solution

Question 1. (20 points) STL. Complete the function `ChangeWords` below. This function has as inputs a vector of strings, and a map of <string, string> key-value pairs. The function should return a new `vector<string>` value (not a pointer) that is a copy of the original vector except that every string in the original vector that is found as a key in the map should be replaced by the corresponding value from that key-value pair.

Example: suppose that the vector `words` is `{"the", "secret", "number", "is", "xlii"}` and the map `subs` contains the pairs `{{"secret", "magic"}, {"xlii", "42"}}`. Then `ChangeWords(words, subs)` should return a new vector containing `{"the", "magic", "number", "is", "42"}`.

Hint: Remember that if `m` is a map, then referencing `m[k]` will insert a new key-value pair into the map if `k` is not already a key in the map. You need to be sure your code doesn't alter the map by adding any new key-value pairs. (Technical nit: `subs` is not a `const` parameter because you might want to use its `[]` subscripting operation in your solution, and `[]` is not a `const` function. It's fine to use `[]` as long as you don't actually change the contents of the map `subs`.)

Write your code below. Assume that all necessary headers and a `using namespace std;` directive have already been written for you.

```
vector<string> ChangeWords(const vector<string> &words,
                           map<string,string> &subs) {

  vector<string> result;
  for (const auto &word: words) {
    if (subs.count(word) != 0) {
      result.push_back(subs[word]);
    } else {
      result.push_back(word);
    }
  }
  return result;
}
```

There are, of course, many other possible solutions and, if done correctly, those received full credit.

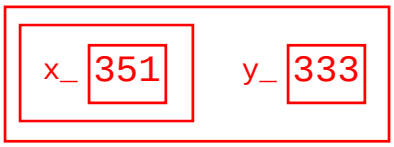
Question 5: C++ Inheritance [24 pts]

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]

Subobject of class A
within class B object.

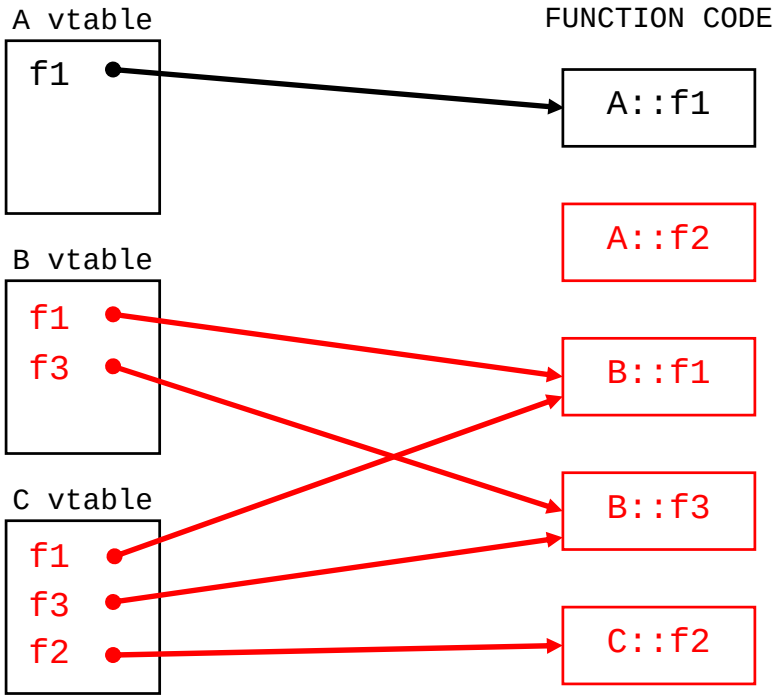


(B) We wish to write constructors for class A and class B to help us initialize our data members. Complete the definitions below: [3 pt]

```
A::A(int x) : x_(x) { }
```

```
B::B(int x, int y) : A(x), y_(y) { }
// must call base class constructor in initializer list
```

(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();`	`A::f1`
`ap1->f2();`	`A::f1` `A::f2`
`ap2->f1();`	`B::f1`
`ap2->f3();`	`compile error` (A has no `f3`)
`bp1->f2();`	`B::f1` (static dispatch of `f1`) `A::f2`
`bp1->f3();`	`B::f3`

2) Casting. `static_cast()` a spell on you

For each of these casts in C++, will it be okay, cause a compile time error, or cause a runtime error?

```
struct A {
     int x;
};
struct B {
     float y;
};
struct C : public B {
     char z;
};

void modify(A* aptr);

int main() {
     A a;
     B b;
     C c;

     B* bptr = static_cast<B*>(&c); // OK, CT Err, RT Err

     C* cptr = static_cast<C*>(&b); // OK, CT Err, RT Err
```

NOTE: This runtime error is silent, there is no way to detect it at runtime. It is an error because it is undefined behavior what will happen when you use cptr now

```
     A* aptr = static_cast<A*>(&b); // OK, CT Err, RT Err

     bptr = &c;
     C* cptr_dyn = dynamic_cast<C*>(bptr); // OK, CT Err, RT Err

     cptr_dyn = dynamic_cast<C*>(&b); // OK, CT Err, RT Err
```

NOTE: This runtime error is that the dynamic_cast returns nullptr, so it can be detected at runtime.

```
     const A const_a;
```

```
    modify(&const_a); // OK, CT Err, RT Err

    modify(const_cast<A*>(&const_a)); // OK, CT Err, RT Err

    int64_t u64 = 0;
    int32_t u32_r = reinterpret_cast<int32_t>(u64);
                        // ^ OK, CT Err, RT Err
    int32_t u32_s = static_cast<int32_t>(u64);
                        // ^ OK, CT Err, RT Err

    double f64_s = static_cast<double>(u64);
                        // ^ OK, CT Err, RT Err
    double f64_r = reinterpret_cast<double>(u64);
                        // ^ OK, CT Err, RT Err
    double* f64_ptr = reinterpret_cast<double*>(&u64);
                        // ^ OK, CT Err, RT Err
}
```

3) Smart pointers (q5 18au). Suppose we have the following declarations at the beginning of a C++ program:

```
int n = 17;
int *x = &n;
int *y = new int(42);
```

Now here are six code fragments that use these variables with a unique_ptr. Each one of these parts is separate from the others – i.e., answer the question for each part assuming it executes immediately after the above variable declarations and nothing else.

For each part, indicate if there is a compile-time error in the code, and, if so, what is wrong; or if the code will compile but will lead to some sort of runtime error or memory leak, describe what will happen; or else indicate that the use of the smart pointer is correct and will delete the memory it manages properly.

(a) `unique_ptr a(n);`

Won't compile. The type of n (int) is not a pointer type.

(b) `unique_ptr b(x);`

Compiles, but fails during execution because the unique_ptr attempts to delete a local variable that is not allocated on the heap.

(c) `unique_ptr c(y);`

Works (There is a potential problem if other code either deletes y or uses that pointer after the heap object is deleted by the smart pointer, but that is not a problem with the smart pointer itself.)

(d) `unique_ptr d(&n);`

Compiles, but fails during execution because the unique_ptr attempts to delete a local variable that is not allocated on the heap.

(e) `unique_ptr e(new int(333));`

Works (The program still leaks memory referenced by y, but that is is a separate issue from whether the smart pointer is used correctly.)

(f)
```
unique_ptr temp(new int(0));
unique_ptr f(temp.get());
```

Compiles, but fails during execution because both unique_ptrs try to delete the single int allocated on the heap (double delete). Many answers said that the code wouldn't compile because two unique_ptrs can't reference the same location. But a compiler doesn't attempt to simulate program execution or track runtime behavior. This code will compile because all of the types are correct.

Question 6. (12 points, 2 each) Recall that the network protocol stack is organized in a sequence of layers. Listed alphabetically they are:

- Application (HTTP, etc.)
- Data link
- Network (IP)
- Physical
- Transport (TCP, UDP, etc.)

For each of the following, identify the layer in the network stack that performs the described operation. Be sure you identify the layer, not the protocol (e.g., IMAP is a protocol, but that does not necessarily identify a particular layer). If it matters for a particular question, you can assume we are using TCP sockets for stream connections.

Note: Answers only needed to include the layer name. The protocol names are shown here for additional information, but were not required.

(a) Transmit an Ethernet packet on the local network from one host machine's NIC interface address to another's.

Data link

(b) Route an incoming message to the correct port to handle a particular service (e.g., web pages, ssh remote login, email, etc.)

Transport (TCP)

(c) Forward a packet from the local wired or wireless network to a different local network if its destination address is not on the same local network.

Network (IP)

(d) If two packets that make up a message arrive out of order, rearrange them into the correct order before they are transmitted to the process reading the data.

Transport (TCP)

(e) Send a `200 OK` message to a web browser in reply to a request for something from a web server.

Application (HTTP)

(f) Discover whether `www.erehwon.com` is a known Internet domain name.

Application (DNS)

Question 6. (12 points) A bit of networking. When we were describing how a network server works, we listed 7 steps that need to be done to establish communication with a client, exchange data, and shut down. In the list below, fill in the name of the function that is used at each step (the reference information at the beginning of the exam may be useful for this), then give a 1-sentence description of the purpose of that step. Step 6 (read/write) is done for you as an example, and the function name for step 2 is also provided. You should fill in the rest of the table.

1. Function: **getaddrinfo** Purpose: **Get ip address and port on which to listen**

2. Function: **socket** Purpose: **Create a socket**

3. Function: **bind** Purpose: **Bind socket created in step 2 to address/port from step 1**

4. Function: **listen** Purpose: **Identify socket as listening socket to which clients can connect**

5. Function: **accept** Purpose: **Accept client connection and get new socket fd that can be used to communicate with client**

6. Function: **read/write** Purpose: exchange data with the client using the socket

7. Function: **close** Purpose: **Shut down client socket and free resources**

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]

2, 4, 5, 6

read 3, read 3, write 6, write 2 → **2**

read 3, write 2, read 2, write 4 → **4**

read 3, write 6, read 6, write 5 → **5**

read 3, read 3, write 2, write 6 → **6**

(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;`	0.5 (or 1.5)
`pthread_mutex_init(&lock, NULL);`	11.5 (or 10.5)
`pthread_mutex_lock(&lock);`	2.5, 6.5
`pthread_mutex_unlock(&lock);`	3.5, 7.5

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

4 and 5 still possible (swap order of thread execution).

2

(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 __**14**__ to _**12.5**_

Equivalently, Move Line 13 to 14.5.

Yes, also possible to put both "work" statements in the same thread:

- Move Line 3 to 6.5 or 7.5
- Move Line 7 to 2.5 or 3.5

Yes, also possible to move the `printf()` statement:

- Move Line 16 to 10.5 or 11.5

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

Worse than sequential. We create a thread, but then wait until it finishes before we create the next one.

Creating one thread that does no work involves unnecessary overhead.

Moving the `printf()` means the output doesn't reflect the work done.