Question 1. (16 points) C++ STL. A map is a collection of <key, value> pairs. For this problem, implement a function `reverse_map(m)` that takes a map `m` of <key,value> pairs as input and returns a new map whose keys are the values from the old map and whose values are a list of the keys from the old map that referenced that value. For example, suppose we have a map of cartoon characters and studios the produced them:

```
{<"bugs bunny", "warner"> <"mickey mouse", "disney"> <"road
runner", "warner"> <"merida", "pixar"> <"jack-jack", "pixar">
<"elmer fudd", "warner"> < "pluto", "disney">}
```

Here is a possible result of `reverse_map(m)`:

```
{<"warner", ["bugs bunny", "elmer fudd", "road runner"]>
<"disney", ["mickey mouse", "pluto"]> <"pixar", ["jack-jack",
"merida"]>
```

Note that the keys in the maps and the entries in the vectors might be in any order. They do not need to be sorted – any ordering encountered by iterating through the original map is fine.

Write your implementation of `reverse_map` below. There is additional space on the next page to continue your answer if needed.

```
map<string, vector<string>> reverse_map(
                            const map<string, string> &m) {
  map<string, vector<string>> result;
  for (auto & pair : m) {
    result[pair.second].push_back(pair.first);
  }
  return result;

}
```

Notes: this solution relies on the useful map property that when we reference `result[x]` for the first time, a new default-constructed `vector<string>` will be created as the value of the map entry whose key is `x`.

The "`&`" reference in the for loop type is not strictly needed. Using it avoids making copies of each <key, value> pair from the original map and is likely more efficient, but either answer (with or without the reference) received credit.

Question 2. (22 points) Classes and memory. Here is a small C++ class that stores a list of string values using a single-linked list of nodes. The class has some problems, which we will explore and fix on the next few pages, but it does compile without problems.

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

class StringList {
public:
  // construct empty list
  StringList() : head_(nullptr), tail_(nullptr) { }

  // add new node with value n to the end of the list
  virtual void add(string s) {
    Node *p = new Node();
    p->val_ = s;
    p->next_ = nullptr;
    if (head_ == nullptr) {
      head_ = p;
    } else {
      tail_->next_ = p;
    }
    tail_ = p;
  }
  // stream output - friend so it can see the list structure
  friend std::ostream &operator<<(std::ostream &out,
                                  const StringList &s);

private:
  // list nodes
  struct Node {
    string val_;
    Node * next_;
  };
  // instance variables
  Node * head_;      // head of list or nullptr if list empty
  Node * tail_;      // last node in list or nullptr if empty
};

// stream output
std::ostream &operator<<(std::ostream &out,
                         const StringList &list) {
    // details omitted - writes the list to stream out using the
    // format [string1, string2, ..., stringn]
}
```

Remove this page from the exam, then continue with the problem on the next pages.
Do not write anything on this page. It will not be scanned for grading.

Question 2. (cont.) To test this class we ran the following main program.

```
int main() {
  StringList str1;  cout<<str1<<endl;
  str1.add("abc");  str1.add("def");
  cout<<str1<<endl;
  StringList str2(str1);
  cout<<str2<<endl;
  str2.add("xyz");
  cout<<str2<<endl; cout<<str1<< endl;
  return 0;
}
```

The results were not what we expected. When we ran the program with valgrind we got the following results (be sure to look at the program output in the middle of the valgrind details – note that both `str1` and `str2` seem to end up with the same value!):

```
==8897== Memcheck, a memory error detector
==8897== Copyright (C) 2002-2017, and GNU GPL'd, ...
==8897== Using Valgrind-3.13.0 and LibVEX; ...
==8897== Command: ./StringListStarter
==8897==
[]
[abc def]
[abc def]
[abc def xyz]
[abc def xyz]
==8897==
==8897== HEAP SUMMARY:
==8897==     in use at exit: 132 bytes in 6 blocks
==8897==   total heap usage: 6 allocs, 0 frees, 132 bytes allocated
==8897==
==8897== 132 (16 direct, 116 indirect) bytes in 1 blocks are
                              definitely lost in loss record 6 of 6
==8897==    at 0x4C2A1E3: operator new(unsigned long)
                              (vg_replace_malloc.c:334)
==8897==    by 0x400F68: StringList::add(std::string)
                              (StringListStarter.cc:16)
==8897==    by 0x400CDF: main (StringListStarter.cc:56)
==8897==
==8897== LEAK SUMMARY:
==8897==    definitely lost: 16 bytes in 1 blocks
==8897==    indirectly lost: 116 bytes in 5 blocks
==8897==      possibly lost: 0 bytes in 0 blocks
==8897==    still reachable: 0 bytes in 0 blocks
==8897==         suppressed: 0 bytes in 0 blocks
==8897==
==8897== For counts of detected and suppressed errors, rerun with:-v
==8897== ERROR SUMMARY:1 errors from 1 contexts(suppressed:0 from 0)
```

Remove this page from the exam, then continue with the problem on the next pages.
Do not write anything on this page. It will not be scanned for grading.

Question 2. (cont.) Now for the questions.

(a) (6 points) What's wrong? Give a brief explanation of the problem(s) with this code and explain why it is producing the incorrect results.

There are two problems:

- **The default copy constructor does a shallow copy of the `head_` and `tail_` instance variables. That means if we accidentally or deliberately copy-construct a new `StringList`, the new and old lists will share data.**
- **There is no destructor to delete the list nodes when a `StringList` is deleted, causing the memory leak. (Note that trying to correct this problem by itself with a destructor that deletes the nodes will cause multiple-delete crashes if the default copy constructor allows multiple `StringLists` to share data, i.e., share pointers to the same heap-allocated nodes.)**

(b) (16 points) Now show how to fix this class so it works correctly. If this involves adding or changing any code, write out the code below, and include everything needed.

But: you only need to make this class work correctly for this particular test program. There might be other things that we would want to do to improve the class or have better C++ style, but you don't need to include everything you can think of. Just fix the actual bugs in this particular program.

(There is a blank page after this with more space if you need it for your answer, which you probably will.)

1) Add a destructor. Solutions that used smart pointers received credit if done properly, but a simple list traversal to delete the nodes is sufficient.

```
virtual ~StringList() {
  Node *p = head_;
  while (p != nullptr) {
    Node *q = p->next_;
    delete p;
    p = q;
  }
}
```

(more space on the next page if needed) **(Solution continued on next page)**

Question 2. (cont.) Additional space for your answer to part (b) – changes and/or additions to the code to fix bugs.

2) Add a copy constructor that does a deep copy of its argument. (To have a truly correct `StringList` class we also need to either disable assignment or provide a version of `operator=` that does a similar deep copy. But that was not required for this problem.)

```
// copy constructor
StringList (const StringList &other) {
  // if other list is empty, initialize empty StringList
  if (other.head_ == nullptr) {
    head_ = tail_ = nullptr;
    return;
  }
  // at least one node in other.  clone first node.
  head_ = new Node();
  head_->val_ = other.head_->val_;
  head_->next_ = nullptr;
  // clone remaining nodes. src, dst are pointers
  // to most recently processed nodes in other and this
  Node *src = other.head_;
  Node *dst = head_;
  while (src->next_ != nullptr) {
    // copy next node from dst and advance
    dst->next_ = new Node();
    dst->next_->val_ = src->next_->val_;
    dst->next_->next_ = nullptr;
    src = src->next_;
    dst = dst->next_;
  }
  // tail_ points to last node in list
  tail_ = dst;
}
```

Notes: it would probably be better if we had included a convenience constructor for the `Node` struct type (i.e., `Node(value, next)`), but since that was not provided in the starter code, we deducted a point for solutions that used it without defining it.

It also is possible to implement the copy constructor by calling `add` on each element from the source `StringList`, and solutions that did that properly received full credit. The above solution clones the list in a single loop inside the copy constructor.

Question 3. (20 points) Virtual holidays! Consider the following C++ program, which does compile and execute successfully.

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

class One {
public:
          void m1() { cout << "H"; }
  virtual void m2() { cout << "l"; }
  virtual void m3() { cout << "p"; }
};

class Two: public One {
public:
  virtual void m1() { cout << "a"; }
          void m2() { cout << "d"; }
  virtual void m3() { cout << "y"; }
  virtual void m4() { cout << "p";}
};

class Three: public Two {
public:
          void m1() { cout << "o"; }
          void m2() { cout << "i"; }
          void m3() { cout << "s"; }
          void m4() { cout << "!"; }
};
```

```
int main() {
  Two t;
  Three th;
  One *op = &t;
  Two *tp = &th;
  Three *thp = &th;

  op->m1();
  tp->m1();
  op->m3();
  op->m3();
  tp->m3();

  op->m1();
  thp->m1();
  op->m2();
  thp->m2();
  tp->m2();
  tp->m1();
  tp->m3();
  thp->m3();
  tp->m4();
  cout << endl;
}
```

(a) (8 points) On the next page, complete the diagram showing all of the variables, objects, virtual method tables (vtables) and functions in this program. Parts of the diagram are supplied for you. **Do not remove** this page from the exam.

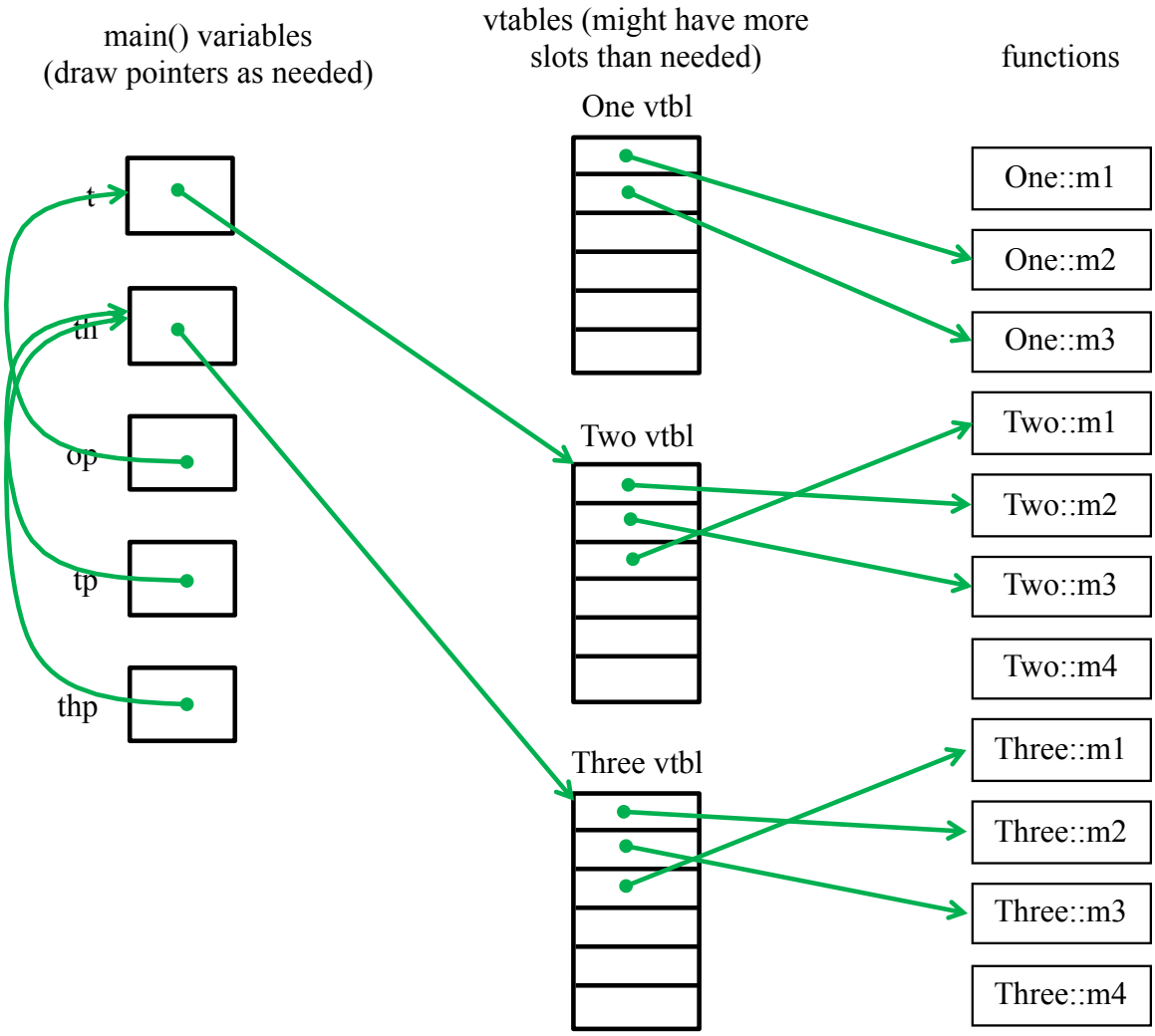
(b) (6 points) What does this program print when it executes?

HoyysHodiiossp

(c) (6 points) Modify the above program by removing and/or adding the `virtual` keyword in appropriate place(s) so that the modified program prints `HappyHolidays!` (including the `!` at the end). Draw a line through the `virtual` keyword where it should be deleted and write in `virtual` where it needs to be added. Do not make any other changes to the program. Any correct solution will receive full credit.

(Changes shown above in bold green)

Question 3. (cont.) Draw your answer to part (a) here. Complete the vtable diagram below. Draw arrows to show pointers from variables to objects, from objects to vtables, and from vtable slots to functions. Note that there may be more slots provided in the blank vtables than you actually need. Leave any unused slots blank.



Notes: The pointers in `One`'s vtable can be in either order – the first slot could point to `m3` and the second one to `m2`. BUT: once that choice is made, the first two vtable slots in `Two`'s and `Three`'s vtables must point to the appropriate versions of the same two functions in the same order as `One`'s vtable. `Two`'s vtable pointer to `m1` must follow those slots, and Three's vtable slots must have exactly the same order.

Functions that are not virtual do not have pointers to them in vtables.

Objects contain only a single pointer to the correct vtable, not multiple pointers to multiple functions or other extra data.

Question 4. (12 points, 2 each) Templates. Suppose we have the following two templates at the beginning of a C++ program.

```
template <typename T>
T calc(T one, int two) { return one; }

template <typename T>
T fcn(T one, T two) { return one; }
```

And then we have a program that contains these variable declarations:

```
double d = 1.0;
float f = 2.0;
char c = '!';
```

Now here are six possible uses of these templates. For each one, indicate if the call is legal or not (i.e., can the template expression be expanded with no errors). If the call is legal, give the type of T that was used to expand the template. If the call is not legal, explain the problem.

Hint: remember that in C and C++, a character constant like 'x' is treated as an integer constant whose value is the numeric value of that character in the ASCII character set (example: for ASCII character 'x' this is the integer value 120).

(a) `calc(c, 'c')`

Legal. `T` is `char`.

(b) `fcn(c, 'c')`

Legal. `T` is `char`.

(c) `calc(d, f)`

Legal. `T` is `double`. (`f` is truncated to an `int` when `calc` is called.)

(d) `fcn(d, f)`

Fails. A single type `T` cannot be both `double` and `float` at the same time.

(e) `calc(c, 17)`

Legal. `T` is `char`.

(f) `fcn(c, 17)`

Fails. A single type `T` cannot be both `char` and `int` at the same time.

Question 5. (12 points, 2 each) Smart pointers. 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<int>a(n);`

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

(b) `unique_ptr<int>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<int>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<int>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<int>e(new int(333));`

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

(f)
```
unique_ptr<int>temp(new int(0));
unique_ptr<int>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. (22 points) Networking – the Internet of Things (IoT). You have been hired by a new startup that is trying to exploit the Internet of Things, and your job is to implement part of a server that manages IoT devices. The server reads data from client IoT devices on the network and performs operations based on that data. Data is transferred using TCP sockets in variable-sized chunks that are called packets in this code (not to be confused with lower-level IP internet packets – in this problem we are talking about data packets for the server application).

Fortunately, one of your colleagues has already used example code from CSE 333 to implement the core parts of the server in C++, including code that listens for connections from IoT devices and accepts connections when an IoT device connects to the server. What you need to do is write the code that reads from the client TCP socket the bytes with the magic number, size, and data, and stores that information in packets that have the following format:

```
typedef struct packet {
  uint16_t magic;          // magic number = 0xC333
  uint16_t size;           // number of bytes in data
  unsigned char * data;    // heap (malloc) array with data
} packet_t;
```

The `magic` field is a magic number, and must be `0xC333` in a correctly formatted packet. The `size` field is the number of bytes in the char array `data` (note that this is a binary array of data bytes, not necessarily a C-format string with a `'\0'` byte at the end). The `magic` and `size` fields are 16-bit binary integers.

(a) (6 points) Binary data is transmitted over the network in the standard network byte order ("big-endian"). On our Linux machines, a `uint16_t` binary number is stored in memory in little-endian order. Complete the definition of the following macro so that `NTOH_16(x)` will expand to an expression that will convert a 16-bit (2 byte) value `x` read from the network into a `uint16_t` binary integer in host little-endian format. Write your answer after the `#define` below.

Hints: Remember to use enough parentheses to avoid problems. You may want to use shift and logical operators like <<, >>, `&` and `|` in your solution.

```
#define NTOH_16(x)  ( (((x)&0xFF)<<8) | (((x)&0xFF00)>>8) )
```

Note that right shift (>>) is guaranteed to fill vacated bits with 0's when the operand is unsigned, so this works as expected with 16-bit unsigned integer values.

There are other expressions that product the correct results and any of those received full credit if done properly.

Question 6. (cont.) (b) (16 points) On the next page, implement the C++ function `get_packet` so that it reads a complete packet from a network socket and returns it to the caller. In your implementation you can assume that the following `WrappedRead` function (from hw4) has already been implemented for you and it works as specified.

```
// A wrapper around "read" that shields the caller from dealing
// with the ugly issues of partial reads, EINTR, EAGAIN, etc.
//
// Reads at most "readlen" bytes from the file descriptor fd
// into the buffer "buf".  Returns the number of bytes actually
// read.  On fatal error, returns -1.  If EOF is hit and no
// bytes have been read, returns 0.  Might read fewer bytes
// than requested by readlen.
int WrappedRead(int fd, unsigned char *buf, int readlen);
```

The function you are to implement has the following specification:

```
// Read a single packet (magic number, size, and data) from
// socket_fd and return the data read using the output parameter
// result. *result is a packet struct already allocated by and
// belonging to the caller.  After reading the packet magic
// number and size, this function will allocate a byte array on
// the heap using malloc with the correct size to hold the packet
// data and then store the complete packet in *result.

// Returns:
//  -1 if an error occurs during reading or if the packet is
//     invalid or if malloc fails
//   0 if no bytes were received before an EOF occurred (i.e., no
//     packet data is available, not even the magic number)
//   1 if a valid packet was successfully received.
int get_packet(int socket_fd, packet_t *result) { … }
```

You may assume that when `get_packet` calls `WrappedRead` to read a 2-byte magic number or length that it will receive 2 bytes if the call succeeds. But when reading the following data buffer, `WrappedRead` might or might not return as many bytes as requested, and your code needs to continue until all of the bytes in the packet are read or until an error terminates reading.

Remove this page from the exam, then write your answer on the next pages. **Do not write anything on this page.** It will not be scanned for grading.

Question 6. (cont) (b) (10 points) Give your implementation of function `get_packet` below.

```
int get_packet(int socket_fd, packet_t *result) {

  int res = WrappedRead(socket_fd,
   (unsigned char *)&result->magic, sizeof(result->magic));
  if (res == 0) {
    return 0;
  } else if (res == -1) {
    return -1;
  }
  result->magic = NTOH_16(result->magic);
  if (result->magic != 0xC333) {
    return -1;
  }

  res = WrappedRead(socket_fd,
     (unsigned char *)&result->size, sizeof(result->size));
  if (res == -1 || res == 0) {
    return -1;
  }
  result->size = NTOH_16(result->size);

  result->data = (unsigned char *)
              malloc(sizeof(unsigned char) * result->size);
  if (result->data == NULL)
    return -1;

  int read = 0;
  while (read < result->size) {
    res = WrappedRead(socket_fd, result->data + read,
                                 result->size - read);
    if (res == -1 || res == 0) {
      return -1;
    }
    read += res;
  }

  return 1;
}
```

Question 7. (20 points) Threads. Consider the following simple C++ program, which prints a sequence of even numbers followed by a sequence of squares. It also contains an extra `#include` and a lock variable that might (☺) be useful later.

```
#include <pthread.h>
#include <iostream>

using namespace std;

static pthread_mutex_t lock;

void print(string what, int num) {
  cout << what << " " << num << endl;
}

// print first n even numbers: 2, 4, 6, ..., 2*n
// you may not modify this funciton
void print_evens(int n) {
  for (int i = 1; i <= n; i++) {
    print("evens", 2*i);
  }
}

// print first n squares: 1, 4, 9, 16, ... n*n
// you may not modify this function
void print_squares(int n) {
  for (int i = 1; i <= n; i++) {
    print("squares", i*i);
  }
}

int main(int argc, char** argv) {
  int nsquares = 4;
  int nevens = 5;

  print_evens(nevens);
  print_squares(nsquares);
  return 0;
}
```

Remove this page from the exam, then continue with the question on the next page. **Do not write anything on this page.** It will not be scanned for grading.

Question 7. (cont.) For this question we would like to modify this program so it executes the two functions `print_evens` and `print_squares` concurrently in separate threads. You may not modify the existing `print_evens` and `print_squares` code. You will need to add appropriate thread starter functions that accept parameters from `main` and call the existing functions with the appropriate arguments. You will also need to make whatever modifications are needed in function `print` so that each output line appears on a separate line by itself without output from the other thread interfering. The existing `print` and `main` functions are copied below and on the next page for you to modify (don't modify the code on the previous page). Make whatever changes and additions are needed to implement correct, concurrent C++ threaded code using pthreads.

```
// modify this function so it is thread safe
void print(string what, int num) {
  pthread_mutex_lock(&lock);
  cout << what << " " << num << endl;
  pthread_mutex_unlock(&lock);
}
// add additional thread starter function definitions here
void *thread_worker_evens(void *arg) {
  int n = *(int*)arg;
  print_evens(n);
  return NULL;
}
void *thread_worker_squares(void *arg) {
  int n = *(int*)arg;
  print_squares(n);
  return NULL;
}
```

(continue with `main` function on the next page)

Question 7. (cont.) Modify the main function below to replace the sequential calls to `print_evens` and `print_squares` with code that executes these two functions concurrently in independent threads and performs whatever other initialization, synchronization, and termination is needed for the concurrent program to work correctly.

```
int main(int argc, char** argv) {
  int nsquares = 4;
  int nevens = 5;


//  print_evens(nevens);
//  print_squares(nsquares);

  pthread_mutex_init(&lock, NULL);
  pthread_t th1, th2;


  pthread_create(&th1, NULL, &thread_worker_evens,
                                        (void*)&nevens);
  pthread_create(&th2, NULL, &thread_worker_squares,
                                        (void*)&nsquares);


  pthread_join(th1, NULL);
  pthread_join(th2, NULL);


  pthread_mutex_destroy(&lock);

  return 0;

}
```

Question 8. (1 free point – all answers get the free point) Draw a picture of something you plan to do over winter break!

Congratulations on lots of great work this quarter!!
Best wishes for the holidays, and say hello when you get back!
The CSE 333 staff