CSE 333 18au Final Exam December 12, 2018

Name _____ ID # _____

There are 8 questions worth a total of 125 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed electronics, closed telepathy, open mind. If you don't remember the exact syntax for something, make the best attempt you can. We will make allowances when grading. Don't be alarmed if there seems to be more space than is needed for your answers – we tried to include more than enough blank space.

There are two pages at the end of the exam with reference information that may be useful.

Relax, you are here to learn.

Please wait to turn the page until everyone is told to begin

Score _____ / 125

1. _____ / 16
2. _____ / 22
3. _____ / 20
4. _____ / 12
5. _____ / 12
6. _____ / 22
7. _____ / 20
8. _____ / 1

Note: Please write your answers only on the specified pages. Reference pages and pages with only questions and explanations will not be scanned for grading, and you should feel free to remove them from the exam.

There is an extra blank page after the last question at the end of the exam if you need additional space for one or more answers. That page will be graded if it contains answers.

There are two pages of reference information following the blank page at the end. You may remove these pages. They will not be scanned or graded.

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) {









}
```

(more space on the next page if needed)

Question 1. (cont.) Additional space for your answer if needed.

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.

(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.)

(more space on the next page if needed)

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

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"; }
          void m4() { cout << "p";}
};

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

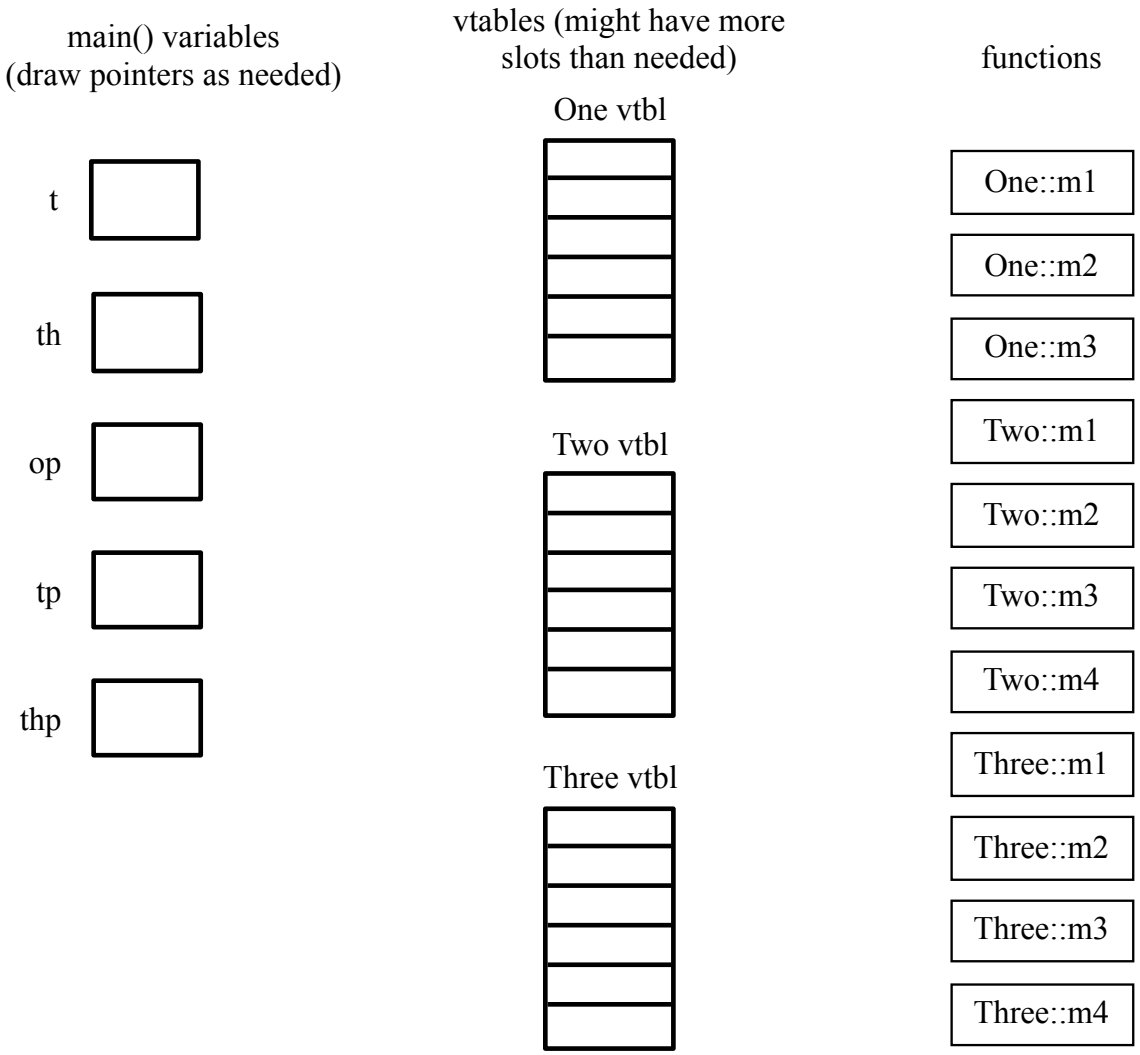
```
int main() {
  Two t;
  Three th;
  One *op = &t;
  Two *tp = &th;
  Three *thp = &th;

  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?

(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.

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.



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')`

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

(c) `calc(d, f)`

(d) `fcn(d, f)`

(e) `calc(c, 17)`

(f) `fcn(c, 17)`

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);
```

(b)
```
unique_ptr<int>b(x);
```

(c)
```
unique_ptr<int>c(y);
```

(d)
```
unique_ptr<int>d(&n);
```

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

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

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)
```

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. (b) (cont) Give your implementation of function `get_packet` below.

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




}
```

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) {

  cout << what << " " << num << endl;

}

// add additional thread starter function definitions here
```

(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);









  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

Extra space for answers, if needed. Please be sure to label which question(s) are answered here, and be sure to put a note on the question page so the grader will know to look here.

Reference information. Here is a collection of information that might, or might not, be useful while taking the test. You can remove this page from the exam if you wish.

C++ strings: If `s` is a string, `s.length()` and `s.size()` return the number of characters in it. Subscripts (`s[i]`) can be used to access individual characters.

C++ STL:

- If `lst` is a STL `vector`, then `lst.begin()` and `lst.end()` return iterator values of type `vector<...>::iterator`. STL `lists` and `sets` are similar.
- A STL `map` is a collection of `Pair` objects. If p is a `Pair`, then `p.first` and `p.second` denote its two components. If the `Pair` is stored in a `map`, then `p.first` is the key and `p.second` is the associated value.
- If `m` is a `map`, `m.begin()` and `m.end()` return iterator values. For a `map`, these iterators refer to the `Pair` objects in the `map`.
- If `it` is an iterator, then `*it` can be used to reference the item it currently points to, and `++it` will advance `it` to the next item, if any.
- Some useful operations on STL containers (lists, maps, sets, etc.):
  - `c.clear()` – remove all elements from c
  - `c.size()` – return number of elements in c
  - `c.empty()` – true if number of elements in c is 0, otherwise false
- Additional operations on `vectors`:
  - `c.push_back(x)` – copy x to end of c
- Some additional operations on `maps`:
  - `m.insert(x)` – add copy of `x` to `m` (a key-value pair for a `map`)
  - `m.count(x)` – number of elements with key `x` in `m` (0 or 1)
  - `m[k]` can be used to access the value associated with key `k`. If `m[k]` is read and has never been accessed before, then a <key,value> `Pair` is added to the map with `k` as the key and with a value created by the default constructor for the value type (0 or `nullptr` for primitive types).
- Some additional operations on `sets`
  - `s.insert(x)` – add `x` to `s` if not already present
  - `s.count(x)` – number of copies of `x` in `s` (0 or 1)
- You may use the C++11 `auto` keyword, C++11-style `for`-loops for iterating through containers, and any other features of standard C++11, but you are not required to do so.

More reference information. You can also remove this page if you wish.

Some POSIX I/O and TCP/IP functions:

- int **accept**(int sockfd, struct socckaddr *addr, socklen_t *addrlen);
- int **bind**(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
- int **close**(int fd)
- int **connect**(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
- int **freeaddrinfo**(struct addrinfo *res)
- int **getaddrinfo**(const char *hostname, const char *service,
  const struct addrinfo *hints, struct addrinfo **res)
  - Use NULL or listening port number for second argument
- int **listen**(int sockfd, int backlog)
  - Use SOMAXCONN for backlog
- off_t **lseek**(int fd, off_t offset, int whence)
  - whence is one of SEEK_SET, SEEK_CUR, SEEK_END
- ssize_t **read**(int fd, void *buf, size_t count)
  - if result is -1, errno could contain EINTR, EAGAIN, or other codes
- int **socket**(int domain, int type, int protocol)
  - Use SOCK_STREAM for type (TCP), 0 for protocol, get domain from address info struct (address info struct didn't fit on this page – we'll include it later if needed)
- ssize_t **write**(int fd, const void *buf, size_t count)

Some pthread functions:

- pthread_create(thread, attr, start_routine, arg)
- pthread_exit(status)
- pthread_join(thread, value_ptr)
- pthread_cancel (thread)
- pthread_mutex_init(pthread_mutex_t * mutex, attr) // attr=NULL usually
- pthread_mutex_lock(pthread_mutex_t * mutex)
- pthread_mutex_unlock(pthread_mutex_t * mutex)
- pthread_mutex_destroy(pthread_mutex_t * mutex)

Basic C memory management functions:

- void * **malloc**(size_t size)
- void **free**(void *ptr)
- void * **calloc**(size_t number, size_t size)
- void * **realloc**(void *ptr, size_t size)