CSE 333 Final Exam March 16, 2016

Name _____ ID # _____

There are 8 questions worth a total of 100 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.

Relax, you are here to learn.

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

Score _____ / 100

1. _____ / 20
2. _____ / 12
3. _____ / 16
4. _____ / 20
5. _____ / 15
6. _____ / 10
7. _____ / 6
8. _____ / 1

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 `list`s 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:
  - `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 operation on `vector`s:
  - `c.push_back(x)` – copy x to end of c
- Some additional operations on `map`s:
  - `m.insert(x)` – add copy of `x` to `m` (a key-value pair for a `map`)
  - `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).
- You are free to 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 use these.

Some POSIX 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
- ssize_t **read**(int fd, void *buf, size_t count)
- 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)

Question 1. (20 points) A bit of C++ coding. As we know, a web page is formed from a mix of tags like `<title>` or `</p>` that control formatting and layout, and plain text content that is displayed as-is. One of your colleagues is working on a program to discover how frequently different tags are used and has asked for your help in writing a key function to do the counting. The input to this function is a C++ `list<string>` that contains the individual words and tags from a set of web pages. The function is required to allocate a new `map<string,int>` on the heap and store in it a collection of (`string,int`) pairs, where each `string` is a tag and the corresponding `int` is how often it appears in the input list. The function should return a pointer to the newly-allocated `map`.

Example: Suppose the input list contains the following strings:

```
"<title>" "fun" "stuff" "</title>" "<p>" "this" "is"
"interesting" "</p>" "<p>" "and" "so" "is" "<this" "</p>"
```

Then the output map produced by the function for this input list should contain the following (string, count) pairs in any order; the ordering is not specified:

```
("<title>", 1), ("</title>", 1), ("<p>", 2), ("</p>", 2)
```

You may assume that all strings in the input list have at least one character, and that strings do not have extra leading or trailing whitespace that needs to be deleted.

Note: A tag is a string that begins with "<" and ends with ">". All other strings should be ignored by the function.

Note: The input words might be from a well-formed, properly nested web document, or they might be complete nonsense. Your function should not attempt to analyze the input beyond determining for each input string whether or not it is a tag and, if it is, count it.

Write your answer on the next page. You can remove this page if you wish.

Question 1. (cont.) Write your implementation below. The function heading is given for you, and you should assume that all necessary headers have already been `#included`. Hint: you probably won't need nearly all of this space.

```
// Return a new heap-allocated map that stores (string,int)
// pairs, where the strings are html tags (e.g., "<thing>")
// and each int is the number of times that string occurs
// in the input list of words. The list of words may contain
// other strings that are not tags and are not counted.
map<string, int> * tagmap(const list<string> & words) {




}
```

Question 2. (12 points) Smart pointers. The new summer intern, A. Hacker, has decided that C++ smart pointers are the solution to all memory management problems and has been going through the company's code repository replacing all pointers (things with type `T*`) with C++ shared pointers (with type `shared_ptr<T>`).

One of the packages that Hacker has changed is one that uses a double-linked list. The original code had nodes that were defined like this:

```
class IntNode {
  int val;          // payload
  IntNode *next;    // next/prev pointers
  IntNode *prev;
};
```

Hacker has replaced that definition with this one:

```
class IntNode {
  int val;                  // payload
  shared_ptr<IntNode> next; // next/prev ptrs
  shared_ptr<IntNode> prev;
};
```

(a) (6 points) One of the other programmers tells Hacker that this can't work. Even though it uses shared pointers everywhere, code that uses the new nodes will still have memory leaks. Is the other programmer right? How is this possible? (Be brief)

(b) (6 points) After listening to the objections, Hacker proposes replacing all of the `shared_ptr<IntNode>` declarations with `weak_ptr<IntNode>`. With this change, will memory management work properly with no leaks or other bugs? Be brief.

Question 3. (16 points) C++ classes. Consider the following program, which compiles and links successfully. (What happens after that is something we'll get to later. ☺)

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

class Base {
public:
  Base() {
    cout << "Base constructor" << endl;
    ia_ = new int[5];
  }
  virtual ~Base() {
    cout << "Base destructor" << endl;
    delete[] ia_;
  }
  Base& operator=(const Base& rhs) {
    if (this != &rhs) {
      delete[] ia_;
      ia_ = rhs.ia_;
      cout << "Base assignment" <<  endl;
    }
    return *this;
  }
private:
  int *ia_;
};

class Derive : public Base {
public:
  Derive() {
    ja_ = new int[5];
    cout << "Derive constructor" << endl;
  }
  virtual ~Derive() {
    cout << "Derive destructor" << endl;
    delete[] ja_;
  }
private:
  int *ja_;
};

int main() {
  Base b1;
  Derive d1;
  b1 = d1;
  return 0;
}
```

(Question continued on next page – you may remove this page if you wish.)

Question 3. (cont.) (a) (8 points) What does this program print when it is executed?

(Reminder/hint: when an object of a derived class is constructed, the base class constructor for that object executes before the derived class constructor. When the object is deleted, the destructors run in the reverse order – derived class destructor first.)

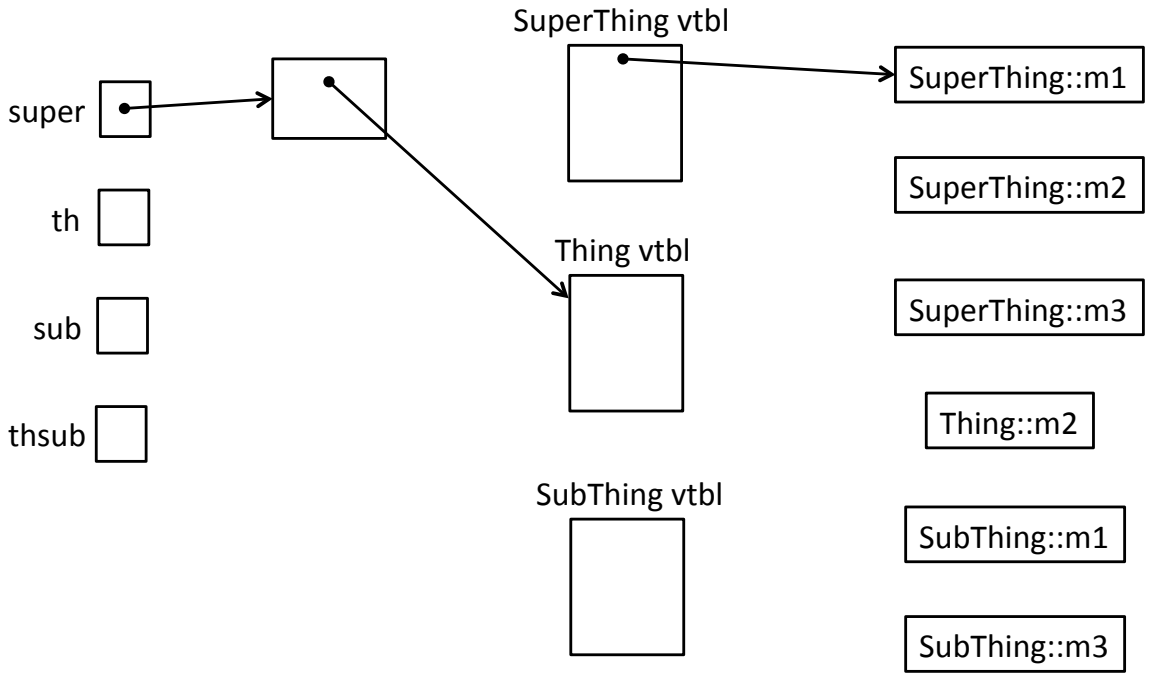
(b) (8 points) Unfortunately, after the program finishes printing the output you described in your answer to part (a), it crashes and does not exit normally. The memory management software detects some sort of problem. What's wrong and what is the error in the code? (Be specific and concise. You do not need to fix the problem – just explain it precisely.)

Question 4. (20 points) The always entertaining virtual function question. The following program compiles, runs, and produces output with no error messages or other problems. Answer questions about it on the next page.

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

class SuperThing {
public:
  virtual void m1() { m2(); cout << "super::m1" << endl; }
          void m2() {       cout << "super::m2" << endl; }
          void m3() {       cout << "super::m3" << endl; }
};
class Thing: public SuperThing {
public:
  virtual void m2() { m1(); cout << "thing::m2" << endl; }
};
class SubThing: public Thing {
public:
  virtual void m1() {       cout << "sub::m1" << endl; }
          void m3() { m2(); cout << "sub::m3" << endl; }
};

int main() {
  SuperThing *super = new Thing();
  Thing      *th    = (Thing*)super;
  SubThing   *sub   = new SubThing();
  Thing      *thsub = sub;
  ///// HERE /////
  cout << "---" << endl;
  th->m1();
  th->m3();
  cout << "---" << endl;
  sub->m1();
  sub->m3();
  cout << "---" << endl;
  thsub->m1();
  thsub->m3();

  return 0;
}
```

(Question continued on next page – you may remove this page if you wish.)

Question 4. (cont.) (a) (8 points) Complete the following diagram to show the runtime state of the program when execution reaches the comment `///// HERE /////` in function `main`. The diagram should include the variables in `main` (already supplied), the objects they point to, pointers from objects to their vtables, and pointers from vtables to the correct functions. To save time, boxes for the variables in main, the vtables, the functions, and the first object created by the program, have been provided for you. A couple of the arrows representing some of the pointers are also included to get you started. You need to supply all additional objects and pointers needed (if any). Be sure that the order of pointers in the various vtables is clear.



(b) (12 points) What does this program print when it is executed?

Question 5. (15 points) Concurrency. The arguments to the following program are a string (`argv[1]`) and a list of one or more files (`argv[2]`, `argv[3]`, etc.). The program does a parallel search to count the number of occurrences of the string in the files. A separate worker thread is created to count each file and then update a global total. Answer questions about this program following the end of the code on the next page. **Do not** detach this page – you may want to write some answers on it.

```
#includes omitted - assume all necessary ones are provided

static char *search_word = NULL;   // Word to look for
static int nfound = 0;     // total number of occurrences of
                           // search_word in all files


// Return number of occurrences of search_word in file
// whose name is fname.  Does exit(1) on failure.
int numOccurences(char *fname) {
  /* Implementation omitted */
}
// Thread "starter" function: search file whose name is given in
// arg and add number of occurrences of search_word to nfound.
void *thread_main(void *arg) {
  char *fname = (char *)arg;
  int count = numOccurences(fname);
  printf("  File: %s - number found: %d\n", fname, count);
  nfound += count;
  return NULL;
}

// Search for string (argv[1]) in files (argv[2], ...) and print
// total number of occurrences of that string in those files.
int main(int argc, char** argv) {
  if (argc < 2) {
    fprintf(stderr, "usage: %s STRING [FILE...]\n", argv[0]);
    return EXIT_FAILURE;
  }
  search_word = argv[1];
  int num_threads = argc - 2;
  printf("Searching for: %s\n", search_word);
  pthread_t *thds = malloc(sizeof(pthread_t) * num_threads);

  for (int i = 0; i < num_threads; i++) {
    if (pthread_create(&thds[i],NULL,&thread_main,argv[2+i])!=0){
      fprintf(stderr, "pthread_create failed\n");
      return EXIT_FAILURE;
    }
  }  // function main continued on next page
```

Question 5. (cont.) Listing of main program, continued.

```
  for (int i = 0; i < num_threads; i++) {
    if (pthread_join(thds[i], NULL) != 0) {
      fprintf(stderr, "pthread_join failed\n");
      return EXIT_FAILURE;
    }
  }
  printf("Total number found: %d\n", nfound);
  free(thds);
  return 0;
}
```

(a) (5 points) Are there any possible race conditions (possible synchronization errors) if this program is used to search a *single* file for the string? Give a precise explanation of why or why not.

(b) (5 points) Are there any possible race conditions (possible synchronization errors) if this program is used to search *multiple* files for the string? (i.e., more than one file name is given in the program arguments) Give a precise explanation of why or why not.

(c) (5 points) If there are any race conditions (possible synchronization errors) in the code, modify or add missing pieces to the code to make it thread-safe. You should write your corrections on the code listing, showing what needs to be changed or inserted. You can declare additional global or local variables as needed. If no changes are needed, simply say so. Some possibly useful function prototypes:

- `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)`

Question 6. (10 points) In the hw3 project we wrote a complex file structure to disk holding hash tables, indexes, and other information about collections of documents.

(a) (5 points) One of the very last things written to the file was a *checksum* field in the file header. What was the purpose of this field (briefly)?

(b) (5 points). The header of the file also contained a "magic number" (in hex, 0xCAFEF00D) and we were very careful to write this at the very end, after everything else had been written. Why? Why wait until the end instead of writing it earlier?

Question 7. (6 points) Our favorite summer intern, A. Hacker, has been messing around in the network code and has decided that the TCP/IP networking software layers have too much overhead. Since TCP chops messages up into small packets to be delivered by IP, he proposes to eliminate TCP sockets from the code and simply send long messages as multiple packets directly, thereby saving the cost of using TCP. Is this a reasonable change? Would we be missing anything if we simply chopped the messages into packets and sent them via IP? If so, what would be missing? (Keep it brief and describe the main issue(s).)

Question 8. (1 free point) Which of the following best matches your opinion about daylight savings time – the reason all the clocks moved forward one hour on Sunday morning. (Circle)

(a) It's great! There's still sunlight when I emerge from the basement to go home.

(b) Really didn't like having to get up an hour earlier for my 8:30 exam this Monday.

(c) Daylight? What is this "daylight" of which you speak?

(d) Love it. Means spring break can't be too far behind.

(e) Stupid idea. Let's leave all the clocks set to standard time.

(f) Everyone should use a sundial. Noon is when the sun is directly overhead.

(g) Stupid idea. All clocks should be set to GMT universal time so we don't have to deal with time zones.

(h) Time zones? Isn't that something that went out with the end of passenger railroads?

(i) Meh. Might get used to it by late summer, and then they just change it back again.

(j) Good reminder that it's time to get the yacht out of storage and go boating again.

(k) Don't care. I'm going to catch up on sleep during spring break so it won't matter what it says on the clock.

(l) Great idea. I'd never remember to replace the smoke detector batteries without it.

(m) It's great! But *not* the day before finals week!!

(x) I don't have an opinion, but please give me my free point anyway.

(y) I do have an opinion, but I'm not going to tell. Please give me my free point anyway.

(z) None of the above, but I still want my free point. My real opinion about daylight savings time is:

Have a great time over spring break!! See you next quarter!!!
The CSE 333 staff