CSE 333 Final Exam Sample Solution 6/11/14

Question 1. (14 points) One of the things we explored this quarter is the different software layers that are involved when an application program executes. For each of the following functions that can be called from a C/C++ program, indicate the layer in the software where the work performed by the function is carried out. In the blank space, write one of the following:

LIB	work done entirely in the C/C++ library function(s) without using the OS
OS	work is done entirely by the Operating System (OS, POSIX) layer
BOTH	work is partially done in a library function and partially done by the OS

You should answer "OS" if the C/C++ library function simply calls the underlying OS routine to do the work and the library functions does nothing other than pass arguments to the OS layer and return a result. If the C/C++ library function does non-trivial work and also calls an underlying OS routine, answer "BOTH".

_____**BOTH**_____ `printf`

______**OS**_______ `fork`

_____**BOTH**_____ `fopen`

______**OS**_______ `read`

______**LIB**______ `sqrt`

_____**BOTH**_____ `malloc`*****

______**LIB**______ `strlen`

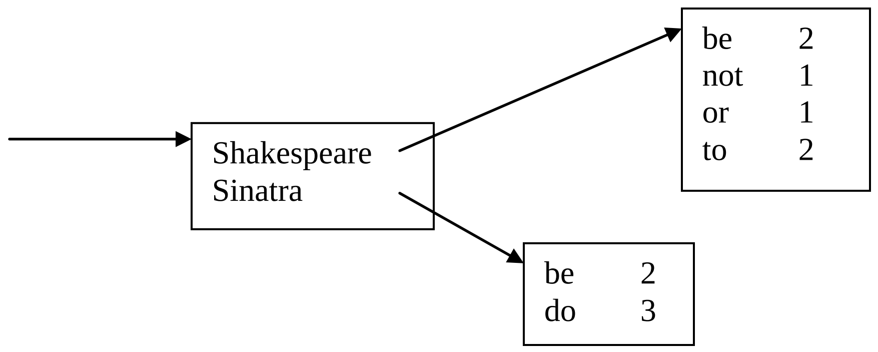
***`malloc` has to call the OS to increase the size of the heap if insufficient storage is available to satisfy a request. We did give half credit for LIB as the answer here since if there is enough available storage in the heap `malloc` can satisfy the request without help from the OS.**

Question 2. (22 points) The traditional C++ hacking question. We would like to implement some functions to keep track of how frequently different authors use various words.

As a specific example, suppose we have two authors “Shakespeare” and “Sinatra”. Each of them has written a few words:

Shakespeare:	“to be or not to be”
Sinatra:	“do be do be do”

We would like to use C++ maps to store information about the words used by these authors in a data structure that looks like this:



The first map is a set of <key,value> pairs where each key is an author name (a string) and the associated value is a pointer to the word count table for that author. Each of the word count tables are a set of <key,value> pairs where each key is a word (a string) and the associated value is the number of times the author has used that word (an int).

The C++ type definitions for these data structures are as follows:

```
// a WordCounts table stores <word,count> pairs
typedef map<string, int> WordCounts;

// An AuthorWordCounts table stores <author, WordCounts*> pairs
typedef map<string, WordCounts*> AuthorWordCounts;
```

All of these data structures are allocated on the heap and accessed via pointers.

When answering the questions, you should assume that all necessary headers like `<map>`, `<string>`, and so forth have been `#include`d for you and a `using namespace std;` directive has also been provided. You do not need to write these.

Question 2. (cont.) Some (incomplete) reference information. You should refer to this as needed, but probably should not spend a lot of time reading it carefully, at least at first.

- 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.
- As with any STL container, if `m` is a `map`, `m.begin()` and `m.end()` return iterator values that might be useful. 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 all STL containers, including `map`:
  - `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
- 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 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.

Hint for all parts of the question: be sure to keep track of which things are pointers and which things are not.

(Question continued on the next page. You may remove this page and the previous one for reference while answering the questions.)

Question 2. (cont.) Finally, something to do....

(a) (5 points) Implement function `CountWord` below. This function adds 1 to the count associated with the word in a table, creating a new entry for the word if needed. Hint: The answer can be very short, but we left a lot of space just in case.

```
// Increment the count for word w in table words, adding w
// to the table with a count of 1 if it was not previously
// present.
// pre: words != nullptr
void CountWord(string w, WordCounts *words) {
  // If w has never been seen before, accessing words[w]
  // will initialize the value to 0.
  (*words)[w] = (*words)[w]+1;  // ++(*words)[w] also works
}
```

Question 2. (cont.) (b) (7 points) Now give an implementation of the following function that increases the count for a word in the correct table for the given author. This function needs to allocate a new `WordCounts` table if this is the first time a word has been counted for that author. You should use the function from part (a) to increase the count for the word in the correct table. Hint: the answer probably won't need all of the space that is provided below.

```
// Increment the count for word w used by author who in
// table awc.  Allocate a new WordCounts table if this is
// the first time we've counted a word for this author.
// pre: awc != nullptr
void CountAuthorWord(string who, string w,
                              AuthorWordCounts *awc) {
  if ((*awc)[who] == nullptr) {
    (*awc)[who] = new WordCounts;
  }
  CountWord(w, (*awc)[who]);
}
```

Question 2. (cont.) (c) (10 points) Implement the following function so that it returns the name of the author who uses the most different words. In our example, the function should return "Shakespeare", since that author uses 4 distinct words, while "Sinatra" uses only 2.

If two or more authors tie for the largest number of distinct words, return the name of any one of them. If the table is empty, return an empty string ("").

Hint: You probably won't need all of this space either.

```
// Return the name of the author in table awc who uses the
// most different words.  If two or more authors tie for
// the maximum, return the name of any one of them.
// If the table is empty, return an empty string ("").
// pre: awc != nullptr

string LargestVocabulary(AuthorWordCounts *awc) {
  int max = 0;      // max number of words seen so far
  string who = "";  // author with max num. words
  for (const auto &author : *awc) {
    if (author.second->size() > max) {
      max = author.second->size();
      who = author.first;
    }
  }
  return who;
}
```

Edge case: this code works provided that at least one author has used 1 or more words. It would produce the wrong result if all authors had empty wordcount tables associated with them. But for this question is was ok to assume that authors are only added to the `AuthorWordCounts` table when we want to count a word, as done in part (b) above. The code could be fixed to avoid this problem by changing > to >= in the `if` statement test.

Question 3. (20 points) Help! We've lost part of this program. Here's what we've got:

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

class One {
public:
  virtual void f() { g(); cout << "One::f" << endl; }
  virtual void g() = 0;  // abstract function
  virtual void h() { cout << "One::h" << endl; }
};

class Two: public One {
public:

   // ????

};

class Three: public Two {
public:
  virtual void f() { cout << "Three::f" << endl; }
  virtual void g() { cout << "Three::g" << endl; }
};

int main() {
  One * p2 = new Two;
  One * p3 = new Three;
  p2->f();
  p3->f();
  p3->h();
  return 0;
}
```

(Continued next page. You may remove this page for reference if you wish.)

Question 3. (cont.) When we run this program it produces the following output:

```
strange
One::f
Three::f
weird
```

Use the code on the previous page and the output given above to figure out the contents of class `Two` and also to complete the diagrams below showing the program variables, the virtual function tables for all three classes, and the functions that these tables point to. To help get started, the virtual function table for class `One` is given below, and blank vtables for you to fill in are supplied for the other two classes. All functions in all classes are `virtual`.

(a) (10 points) Complete the code for class `Two` here:

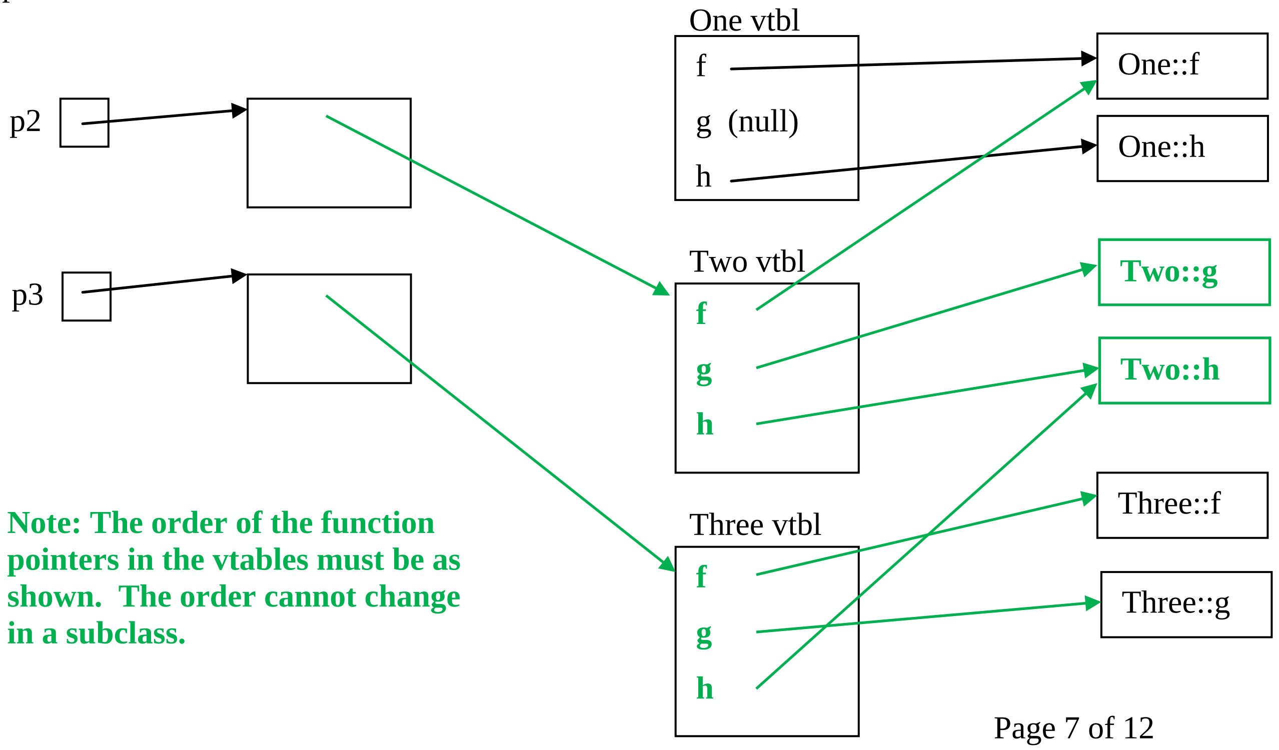
```
class Two: public One {
public:

  virtual void g() { cout << "strange" << endl; }

  virtual void h() { cout << "weird" << endl; }

};
```

(b) (10 points) Draw the rest of the virtual function tables, functions and necessary pointers below.



Note: The order of the function pointers in the vtables must be as shown. The order cannot change in a subclass.

Question 4. (12 points) Shared pointers and reference counting. Recall that a shared pointer keeps track of a reference count for the object that it points to. When a shared pointer is deleted or reassigned the reference count is adjusted as needed and, if that results in a reference count of zero, the object owned by the shared pointer is deleted.

For example, here is a sequence of operations on shared pointers and the resulting reference counts:

```
std::shared_ptr<int> x(new int(10)); // int(10) ref count = 1
std::shared_ptr<int> y = x;          // int(10) ref count = 2
 x = nullptr;                       // int(10) ref count = 1
 x = y;                             // ref count = 2 again
```

So what exactly happens when an assignment operator `p=q` for a shared pointer is executed? Below, write a pseudo-code algorithm that describes the detailed steps that must be done by the assignment operator to properly execute the assignment `p=q` when `p` is a `shared_ptr`. Be sure to describe exactly when reference counts are adjusted and how (incremented? decremented?), and when any delete operations are performed. Use informal, C-like code, and write things like "increase the reference count of ..." when it helps make things clear. The grader needs to understand what you write in order to award full credit. (Hint: there is likely to be at least one "if" statement in the answer.)

```
/* pseudo code for p=q when p is a shared pointer */

// do nothing if self-assignment (don't change ref. counts
// or accidentially delete something)

if (p == q) return;

// ordinary assignment - decrease reference count of old *p
// and delete it if this was the last reference, then copy q

decrease reference count of *p;

if (reference count of *p == 0)

   delete *p;

copy q into p;

increase reference count of *p (which is also *q);
```

Note: The question was intended to be about how reference counts are adjusted and when this can trigger delete operations while assigning `shared_ptr` values. We should have stated that more clearly to avoid issues about what happens if `q` is an ordinary pointer or if either pointer initially has the value `nullptr`, not to mention what has to be done to make a `shared_ptr` work properly if it is accessed simultaneously by multiple threads. We didn't look at those issues while grading the question. Getting those things right as well as all of the template issues, makes `shared_ptr` one of the trickiest parts of the C++ standard library.

Question 5. (15 points) Threads and locks. Consider the following program (which looks suspiciously like similar programs from previous CSE 333 final exams):

```
int x = 0;
int y = 0;

pthread_mutex_t lock;

void * worker(void * ignore) {
  pthread_mutex_lock(&lock);
  x = x + 1;
  y = y + 1;
  printf("x = %d, y = %d\n", x, y);
  pthread_mutex_unlock(&lock);
  return NULL;
}

int main() {
  pthread_t t1, t2;
  int ignore;
  pthread_mutex_init(&lock, NULL);
  ignore = pthread_create(&t1, NULL, &worker, NULL);
  ignore = pthread_create(&t2, NULL, &worker, NULL);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  printf("final x = %d, y = %d\n", x, y);
  return 0;
}
```

(a) (8 points) What output is produced by this program when it executes? If it is possible to get different output when the program is executed a second time, write two different possible outputs from two different executions. If the program always produces the same output, give that output and indicate that it is always the same.

The output will always be

```
x = 1, y = 1
x = 2, y = 2
final x = 2, y = 2
```

It is true the threads can run in any order after they are created. But whichever thread runs first will grab the lock, increment x and y, and print them before the other thread can do the same. Once both threads are finished the final line of output is produced.

(continued next page)

Question 5. (cont.) (b) (7 points) Now suppose we change the body of function `worker` by moving the lock operation down one line:

```
void * worker(void * ignore) {
  x = x + 1;
  pthread_mutex_lock(&lock);
  y = y + 1;
  printf("x = %d, y = %d\n", x, y);
  pthread_mutex_unlock(&lock);
  return NULL;
}
```

How does this change your answer to part (a), if at all? In particular, is it possible for the program to produce output that is different from and could not have been produced by any execution of the original program? If so, explain what output could be produced by this version of the program that could not have been produced before. Concise answers are appreciated.

In this case access to x is not synchronized with a lock. It would be possible for both threads to increment x before either thread acquires the lock to increment y and print the values. It is also possible that both threads could read the original value of x, both could add 1 to that, and both could store 1 in x as its final value. So in addition to the original output produced by the program, here are two other possibilities:

```
x = 2, y = 1
x = 2, y = 2
final x = 2, y = 2
```

```
x = 1, y = 1
x = 1, y = 2
final x = 1, y = 2
```

Question 6. (16 points) The internet protocols are organized in layers, with each layer being implemented on top of, and building on the layer below. From the bottom up, the layers we looked at were:

- physical
- data link
- network (IP)
- transport (TCP)
- application (HTTP, SMTP, etc.)

For each of the following tasks that are performed in the networking software, write the name of the layer that performs the task in the blank to the left.

____**IP**___ Deliver a single packet across the network on a "best-efforts" basis

appl (DNS) Resolve a domain name to a numeric IP address (for example, convert "www.cs.washington.edu" to 128.208.3.88)

__**TCP**___ Break a longer message into packets for transmission and reassemble them at the receiving end.

data link Route a packet from one device to another on the same subnet (local network)

__**TCP**___ Recover from errors if a packet is lost when transmitting a message

physical Convert analog signals (voltages, radio waves) to digital bits

__**TCP**___ Route packets to the correct "port" (like port 80 for HTTP)

__**appl**__ Transmit web pages between a server and a web browser (as in HW4)

Question 7. (1 free point) The best, and only reasonable, operating system is: (circle the letter of the *Correct* answer)

a) MS-DOS

b) Windows 95

c) Windows NT

d) Windows XP

e) Windows 7

f) Windows 8

g) OS X 10.6

h) OS X after 10.6

i) Ubuntu Linux

j) BSD Linux

k) Fedora Linux

l) Debian Linux

m) QNX

n) Android

o) iOS

p) Multics

q) OS/360

r) Unix

s) Plan 9

t) OS/161

u) I don't care – just give me my free point

v) I do care, but I'm not telling – I want the free point anyway

w) Other: ____________________________

All answers received the point (even the person who forgot to answer the question!)

Have a great summer break! See you in the fall!!

The CSE 333 staff