CSE 333 Final Exam August 19, 2016

Name _____ ID # _____

There are 6 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. _____ / 24
2. _____ / 22
3. _____ / 22
4. _____ / 16
5. _____ / 14
6. _____ / 2

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 and `set`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 (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 `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.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 `set`s
  - `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.

Question 1. (24 points) A bit of C++ coding. This question concerns a class that stores a list of strings using a single-linked list. The constructor and private data are given below as they would appear in the header file `StringList.h`. Remaining functions, including the destructor, are omitted to save space.

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

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

private:
  // list nodes with convenience constructor
  struct Node {
    string val;
    Node * next;
    Node(string n, Node* nxt): val(n), next(nxt) { }
  };
  // instance variable
  Node * head_;      // head of list or nullptr if list empty
};
```

Your job is to declare and implement an assignment operator (`operator=`) for this class. If `s1` and `s2` are two `StringList` objects, then the assignment `s1=s2` should replace the existing contents of `s1` with a complete copy (clone) of the list `s2`.

Hint: be careful to avoid memory leaks and any other memory-management problems.

(a) (4 points) Give a correct declaration of the assignment operator (`operator=`) that should be added to the `StringList` class declaration above.

(continued on next page)

Question 1 (cont.) (b) (20 points) Give a complete implementation of the `StringList` assignment operator (`operator=`) as it would appear in the implementation file `StringList.cc`. Some header files have been `#include`d for you. Your solution must be self-contained – i.e., you can only use the `Node` constructor on the previous page to create list nodes; you can't call other `StringList` functions.

```
#include <string>
#include "StringList.h"
using namespace std;

// write the StringList operator= implementation below
```

Question 2. (22 points) Templates, STL, function pointers, and smart pointers all at once! – oh my!!! Don't panic – the answers to this question are actually quite short.

Sometimes a function is called many times to recomputed values. If that turns out to be expensive we may be able to save time by storing previously computed values in a cache and reuse them instead of computing them from scratch each time.

For this problem we want to implement parts of a class named `FunctionCache`. The idea is that an instance of `FunctionCache` stores a pointer to a function *f* and a map of <argument, result> pairs, where the result in each pair is computed by applying function *f* to the corresponding argument. The class also includes a function `apply` that is used by clients to compute values instead of of calling *f* directly. When `apply(x)` is called, it looks in the map to see if x is stored there as a key. If so, it returns the result found in the map without calling function *f*. If x does not appear as a key in the map, then `apply` calls *f* to compute *f*(x), then stores <x, *f*(x)> in the map for future use, and finally returns the result to the caller.

Because we want this to work for all single-argument functions, the class is a template whose parameters are the argument and result types of the function. Here is the main part of this template, except for the `apply` function code:

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

// Wrapper class for a function with argument type T and
// result type U.
template <typename T, typename U>
class FunctionCache {

public:
  // construct a new FunctionCache.  Argument f is a pointer
  // to the function to be used to compute results.
  FunctionCache(U (*f)(T)): cache_(new map<T, U>()), func_(f) { }

  // destructor
  ~FunctionCache() {
    delete cache_;
  }

  // return func_(val), but use cache_ to store and retrieve
  // previously computed values instead of always calling func_
  U apply(T val) { ... }

private:
  map<T, U> *cache_;    // cache of <argument, result> pairs
  U (*func_)(T);        // the function
};
```

(Continued on the next page. You may remove this page for reference.)

Question 2. (cont.) Here is an example of how a client program might use the FunctionCache template:

```
// sample function - return x/2.0
double half(int i) { return i/2.0; }

int main(int argc, char **argv) {
  FunctionCache<int, double> hcache(half);
  double x = hcache.apply(3);  // computes and saves <3,1.5>
  double y = hcache.apply(4);  // computes and saves <4,2.0>
  double z = hcache.apply(3);  // returns 1.5 from the cache
  ...                          //   without recomputing
  return 0;
}
```

(a) (16 points) Provide the full definition of function `apply` for the `FunctionCache` template. Write your solution below. Hint: the solution may be quite short – do not be alarmed.

```
  // return func_(val), but use cache_ to retrieve previously
  // computed values, and store new values there

  U apply(T val) {









  }
```

(continued on next page)

Question 2. (cont.) (b) (6 points) The original version of the `FunctionCache` template uses explicit memory management (`delete`) in the destructor to deallocate the `cache_` map data when an instance of `FunctionCache` is destroyed.

Another way to handle the heap data would be to use smart pointers instead of an explicit `delete` to ensure that the `cache_` data is deallocated when a `FunctionCache` object is destroyed.

Below, give a precise description of the changes that need to be made to the original template to make this change. If any code needs to be added or altered, write the new code here and be sure it is clear where the changes should be made and what, if anything, in the original code should be deleted or replaced. If any changes are needed in your implementation of `apply` in part (a), also describe them here.

Question 3. (22 points) The customary virtual madness. The following program compiles, runs, and produces output with no error messages or other problems.

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

class A {
public:
          void w() { cout << "A::w" << endl; }
  virtual void x() { cout << "A::x" << endl; }
          void y() { cout << "A::y" << endl; }
};

class B: public A {
public:
  void w() {      cout << "B::w" << endl; }
  void x() { y(); cout << "B::x" << endl; }
  void z() { y(); cout << "B::z" << endl; }
};

class C: public B {
public:
  void x() { w(); cout << "C::x" << endl; }
  void y() {      cout << "C::y" << endl; }
};

int main() {
  A* a = new C();
  a->w();
  a->x();
  a->y();
  cout << "---" << endl;
  B* p = new C();
  p->w();
  p->x();
  p->y();
  p->z();
  cout << "---" << endl;
  C* c =  new C();
  c->w();
  c->x();
  c->y();
  c->z();
  return 0;
}
```

What does this program print when it executes? Write your answer to the right of the code above.

Question 4. (16 points) A bit of concurrency. Consider the following C++ program, which does compile and execute with no apparent errors.

```
#include <iostream>
#include <pthread.h>
const int SIZE = 5;

void *threadFunc(void * arg) {
  int *intArr = (int *) arg;
  for (int i = 0; i < SIZE; i++) {
    intArr[i] = i;
  }
  return nullptr;
}

int main (int argc, char **argv) {
  const int NUM_ARRAYS = 3;

  int *arrays[NUM_ARRAYS];
  for (int i = 0; i < NUM_ARRAYS; i++) {
    arrays[i] = new int[SIZE];
    for (int j = 0; j < SIZE; j++)
      arrays[i][j] = -1;
  }

  pthread_t threads[4];
  for (int i = 0; i < NUM_ARRAYS; i++) {
    pthread_create(threads + i, nullptr, threadFunc, arrays[i]);
  }

  for (int i = 0; i < NUM_ARRAYS; i++) {
    pthread_join(threads[i], nullptr);
  }

  for (int i = 0; i < NUM_ARRAYS; i++) {
    std::cout << "Array " << i << std::endl;
    for (int j = 0; j < SIZE; j++) {
      std::cout << arrays[i][j] << " ";
    }
    std::cout << std::endl;
  }
}
```

(continued on the next page – you may remove this page for reference)

Question 4. (cont)

(a) (4 points) Does this program produce the same output every time it is executed? (yes or no – no explanation needed)

(b) (12 points) If the program always produces the same output, write that output below. If it is possible for the program to produce different output when it is run more than once, show two of the possible results that the program could produce.

Question 5. (14 points, 2 each) The Berkeley sockets API is built on top of the layered network architecture we discussed in class. Here is an alphabetical list of the layers (i.e., this is not necessarily the order in which the layers call each other to get work done):

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

Suppose we are using the network sockets API to communicate between a web browser and a server, i.e., just like in HW4. For each of the following, identify the layer in the network stack that performs the described operation. Be sure you identify the layer, not just the protocol (i.e., POP is a protocol, but that would not be a good answer since it doesn't necessarily identify a particular layer.) If it matters, you should assume that we're using TCP sockets for the stream between the browser and server.

(a) Decide how to route a packet to the next node in the network in order to get it closer to its final destination.

(b) Check whether a packet is missing from a stream and, if so, arrange to have it retransmitted.

(c) Resolve a domain name (like www.cs.washington.edu) into an internet address (like 128.208.3.88).

(d) Convert analog voltages on the network wire to digital 0's and 1's.

(e) Send the "404 not found" error reply in response to an attempt to retrieve a non-existent web page.

(f) Route messages intended for different services (e.g., email, web, remote login) to the appropriate process(es) running on the machine that should handle them.

(g) Forward a packet from one network to another if the destination is another machine that is not connected to the same local network.

Question 6. (2 free points – all answers get the free points)

(a) (1 free point) If you had a chance to learn (or continue) to play a musical instrument, which one would you choose? (circle)

i) Piano

ii) Guitar

iii) Violin/fiddle

iv) Trumpet

v) Trombone

vi) French Horn

vii) Flute

viii) Oboe

ix) Bassoon

x) Clarinet

xi) Saxophone

xii) Drums

xiii) Bass

xiv) Zither

xv) Ondes Martenot

xvi) Tuba

xvii) Crumhorn

xviii) Xylophone

xix) Bagpipes

xx) Something else – describe _____

(b) (1 free point) Why did you pick the instrument that you selected above?

Have a great summer vacation!!
The CSE 333 staff