CSE 333 Final Exam December 13, 2017

Name _____ ID # _____

There are 12 questions worth a total of 135 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 _____ / 135

1. _____ / 16
2. _____ / 24
3. _____ / 32
4. _____ / 18
5. _____ / 6
6. _____ / 6
7. _____ / 6
8. _____ / 6
9. _____ / 6
10. _____ / 6
11. _____ / 6
12. _____ / 3

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) STL. Implement the template function `is_sorted`, below, so it returns `true` if its list argument is sorted in non-decreasing order, and returns `false` if not. Your function should work for any list whose elements can be compared using `operator<` (the usual minimal requirement for ordered values in STL containers). If the list is empty (contains no elements), the function should return `true`.

Examples: `is_sorted` should return `true` for the integer list { 1, 4, 7, 7, 10} and should return `false` for { 1, 4, 3, 7, 7, 10 }.

Restriction: The STL algorithms library contains a similar function. You may not use it in your solution. Your function should examine the elements in the list to determine if the list is or is not sorted.

Hints: Remember that if `x` is a STL container, `x.begin()` and `x.end()` are iterator values that might be useful. Variable declarations can often be simplified with `auto`. There is some reference information about the STL libraries at the end of the exam.

Write your code below. Assume that all necessary headers have been `#include`d, but there is no `using namespace std;` directive (you can add one if you want to).

```
template <typename T>
bool is_sorted(std::list<T> lst) {




}
```

Question 2. (24 points) And the point is? One of the summer interns left behind some C++ code that was supposed to implement circles containing a center point and a radius. The code is a bit odd, but we need to figure out what it does. Header guards are omitted to save space.

File Point.h:

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

class Point {
public:
  Point(): x_(0), y_(0) { cout << "Point()" << endl; }
  Point(const int x, const int y): x_(x), y_(y)
                       { cout << "Point(int,int)" << endl; }
  int x() {return x_;}
  int y() {return y_;}
private:
  int x_, y_;
};
```

File Circle.h:

```
#include "Point.h"
#include <iostream>
using namespace std;

class Circle {
public:
  Circle(): radius_(1) {
    center_ = new Point();
    cout << "Circle()" << endl;
  }
  Circle(const int radius, Point center): radius_(radius) {
    center_ = new Point(center);
    cout << "Circle(int,Point)" << endl;
  }
  ~Circle() {delete center_; cout << "~Circle()" << endl; }

  int radius() { return radius_; }
  bool isLarger(Circle other);
  void shift(Point newCenter);

private:
  int radius_;
  Point *center_;
};
```

(Code continued on the next page. You should remove this page from the exam. **Do not write anything on this page.** It will not be scanned for grading.)

Question 2. (cont.) More files:

File Circle.cc:

```
#include "Circle.h"
bool Circle::isLarger(Circle other) {
  return radius_ > other.radius();
}
void Circle::shift(Point p) {
  delete center_;
  center_ = new Point(p.x(), p.y());
}
```

File circletest.cc:

```
#include "Point.h"
#include "Circle.h"
#include <iostream>
int main() {
  Circle a = Circle(3, Point(1, 2));
  Circle b;
  std::cout << a.isLarger(b) << std::endl;
  Circle c = a;
  a.shift(Point(2, 2));
  return 0;
}
```

(a) (12 points) What output is produced when we compile and run this program? (It does run and terminate without crashing, although there may be some bugs). Note that the various constructors and destructors print messages when they are executed.

(continued on next page)

Question 2. (cont.) When we ran this program using valgrind, it produced the following messages (program output omitted):

```
==9531== Invalid free() / delete / delete[] / realloc()
==9531==    at 0x4C2B18D: operator delete(void*) (vg_replace_malloc.c:576)
==9531==    by 0x400E7F: Circle::~Circle() (Circle.h:22)
==9531==    by 0x400C9C: main (circletest.cc:10)
==9531==  Address 0x5a19040 is 0 bytes inside a block of size 8 free'd
==9531==    at 0x4C2B18D: operator delete(void*) (vg_replace_malloc.c:576)
==9531==    by 0x400A9A: Circle::shift(Point) (Circle.cc:8)
==9531==    by 0x400C8B: main (circletest.cc:11)
==9531==  Block was alloc'd at
==9531==    at 0x4C2A203: operator new(unsigned long) (vg_replace_malloc.c:334)
==9531==    by 0x400E35: Circle::Circle(int, Point) (Circle.h:19)
==9531==    by 0x400BFB: main (circletest.cc:7)
==9531==
==9531== Invalid free() / delete / delete[] / realloc()
==9531==    at 0x4C2B18D: operator delete(void*) (vg_replace_malloc.c:576)
==9531==    by 0x400E7F: Circle::~Circle() (Circle.h:22)
==9531==    by 0x400CA8: main (circletest.cc:8)
==9531==  Address 0x5a19090 is 0 bytes inside a block of size 8 free'd
==9531==    at 0x4C2B18D: operator delete(void*) (vg_replace_malloc.c:576)
==9531==    by 0x400E7F: Circle::~Circle() (Circle.h:22)
==9531==    by 0x400C52: main (circletest.cc:9)
==9531==  Block was alloc'd at
==9531==    at 0x4C2A203: operator new(unsigned long) (vg_replace_malloc.c:334)
==9531==    by 0x400DBE: Circle::Circle() (Circle.h:15)
==9531==    by 0x400C07: main (circletest.cc:8)
==9531==
==9531== HEAP SUMMARY:
==9531==     in use at exit: 0 bytes in 0 blocks
==9531==   total heap usage: 3 allocs, 5 frees, 24 bytes allocated
==9531==
==9531== All heap blocks were freed -- no leaks are possible
==9531==
==9531== For counts of detected and suppressed errors, rerun with: -v
==9531== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Answer questions about this output on the next page. Remove this page from the exam. **Do not write anything on this page.** It will not be scanned for grading.

(continued on next page)

Question 2. (cont.) (b) (6 points) What error(s) in the original code caused the errors reported in the valgrind output on the previous page? (Hint: it might be useful to trace part of the execution of the program in addition to puzzling over the valgrind output.)

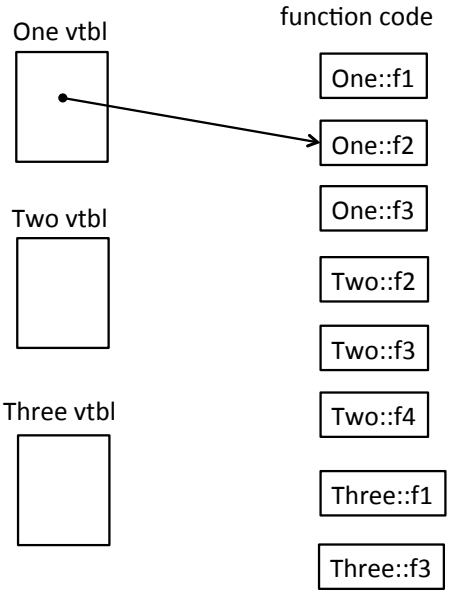
(c) (6 points) How should we fix the original code to eliminate these errors? Your change(s) should fix the problem(s) without changing the basic operation of the original code and should retain the original data representation and data structures of the objects to the extent possible. In other words, explain how to fix the problem(s) in the most straightforward, simplest way you can.

Question 3. (32 points) The usual, demented, dreaded virtual function madness. Consider the following program, which, when appropriate code is inserted in the blank space in `main`, does compile and execute with no errors.

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

class One {
public:
          void f1() { f3(); cout << "One::f1" << endl; }
  virtual void f2() {       cout << "One::f2" << endl; }
          void f3() {       cout << "One::f3" << endl; }
};

class Two: public One {
public:
          void f4() {       cout << "Two::f4" << endl; }
          void f2() { f1(); cout << "Two::f2" << endl; }
  virtual void f3() { f4(); cout << "Two::f3" << endl; }
};

class Three: public Two {
public:
  void f3() { f2(); cout << "Three::f3" << endl; }
  void f1() {       cout << "Three::f1" << endl; }
public:
};

int main() {
   [          ]
  return 0;
}
```

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

Question 3. (cont.) (a) (6 points) Complete the diagram below to show the layout of the virtual function tables for the classes given on the previous page. Be sure that the order of pointers in the virtual function tables is clear (i.e., which one is first, then next, etc.). One of the function pointers is already included to help you get started.



(b) (10 points) Now, for each of the following sequences of code, assume that we try to run the program with the given lines of code replacing the empty box in `main`. Either write the output that is produced when that program is executed, or, if an error occurs, give a concise description of the problem.

(i)
```
One *x = new Two();
x->f1();
```

(ii)
```
One *x = new Two();
x->f3();
```

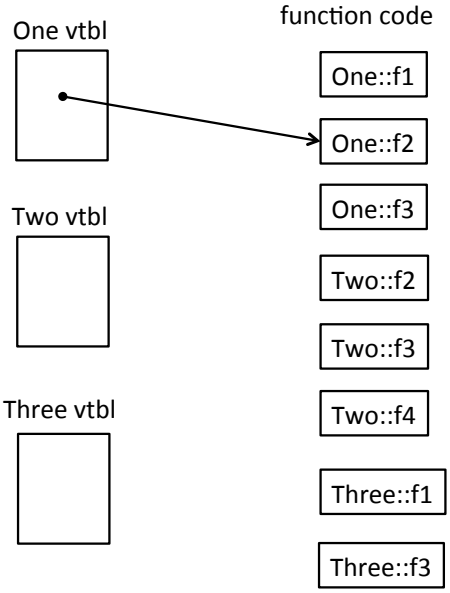
(iii)
```
Two *x = new Two();
x->f3();
```

(iv)
```
One *x = new Three();
x->f4();
```

(v)
```
Three *x = new Three();
x->f3();
```

(continued on next page)

Question 3. (cont.) (c) (6 points) Now, assume that we change the original code by writing the keyword `virtual` in front of every function declared in classes `One`, `Two`, and `Three`. Complete the vtable diagram below to show the vtable layouts for the three classes with this change. As before, be sure the order of pointers in each vtable is clear.



(d) (10 points) As before, show the output produced after changing all of the member functions in the classes to be `virtual` and then replacing the empty box in `main` with each of the following sequences of code. Either write the output that is produced when the program is executed, or, if an error occurs, give a concise description of the problem.

(i)
```
One *x = new Two();
x->f1();
```

(ii)
```
One *x = new Two();
x->f3();
```

(iii)
```
Two *x = new Two();
x->f3();
```

(iv)
```
One *x = new Three();
x->f4();
```

(v)
```
Three *x = new Three();
x->f3();
```

Question 4. (18 points) Too many things at once. Consider the following small program that uses pthreads. (Header files omitted to save space. This does compile and execute.)

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

int x = 0;
int y = 0;

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

int main() {
  pthread_t t1, t2, t3;
  int ignore;
  ignore = pthread_create(&t1, NULL, &worker, NULL);
  ignore = pthread_create(&t2, NULL, &worker, NULL);
  ignore = pthread_create(&t3, NULL, &worker, NULL);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  pthread_join(t3, NULL);
  printf("final x = %d, y = %d\n", x, y);
  return 0;
}
```

When we run this program it starts three threads and waits for them all to finish, and then prints the final value of the variables `x` and `y`.

(a) (6 points) What output would this program print if the threads do not interfere with each other (i.e., if, for example, the three threads were executed sequentially, one after the other, rather than running concurrently):

(continued on next page)

Question 4. (cont.) (b) (6 points) When the threads run concurrently, is it possible to get different output when the program is executed repeatedly? If it is, give three possible outputs that could be produced by the program. If there is only one or two possible outputs, write those and indicate that they are the only possible results.

(You should assume that the statements in each individual thread are executed in the order written, and not rearranged by the compiler or memory system to be executed out-of-order. You should also assume that the `printf` calls don't interfere with each other and that each line of output is printed correctly and separately from other output lines. If different executions lead to different outputs it is only because of the interaction between the threads as they run concurrently.)

(c) (6 points) Assuming that the threads are executed concurrently, as in part (b), what are the possible final values of variables x and y? Circle all that could possibly happen on some possible execution:

Possible final values for x:

0 1 2 3 4 5 6 7 8 9 10 or more

Possible final values for y:

0 1 2 3 4 5 6 7 8 9 10 or more

Some shorter questions…

Question 5. (6 points) Two ways to implement a concurrent server are to use threads and processes (fork).

(a) Give one distinct technical advantage of processes compared to threads.

(b) Give one distinct technical advantage of threads compared to processes.

Question 6. (6 points) TCP and UDP are two transport-level protocols. TCP implements streams while UDP sends "datagrams" – i.e., messages like those sent by IP except that UDP allows messages of any length, which it splits into packets to send via IP. For each protocol, give an example of one user-level application that would be better suited to that protocol than to the other one. (Think about apps that run on your computer or phone.)

TCP:

UDP:

Question 7. (6 points) C++ allows functions in classes to be either virtual, using dynamic dispatching to select methods (functions) at runtime, or non-virtual, which does not use the dynamic dispatch mechanism. In Java, all method calls are virtual and there are no non-virtual methods associated with objects. Give two technical reasons why it is sometimes useful in C++ programs to use non-virtual methods, even though this is rare:

(i)

(ii)

Question 8. (6 points) Describe one fundamental technical difference between a C++ reference and a C++ pointer (other than the obvious ones of the syntax used to declare or initialize them).

Question 9. (6 points) In a C++ class, assignment (`operator=`) and the copy constructor are two distinct functions, even though both of them normally assign a new value to an object. What is the fundamental reason that these two operations are separate rather than just having a single "assignment/initialization" operation for C++ classes? i.e., what is the fundamental difference between assignment and copy construction?

Question 10. (6 points) Why are there `weak_ptrs`? The main C++ smart pointers are `unique_ptr` and `shared_ptr`, which should be enough to handle most needs. But there is this third kind, `weak_ptr`. What is the primary reason it exists?

Question 11. (6 points) Two of the functions used by a network server to establish connections with client programs are `accept()` and `listen()`. What do these two functions do and in which order should server code execute them?

Question 12. (3 free points – all answers get the free points)

Who is the most awesome Star Wars character? Feel free to give your reason(s) and/or draw a picture to explain your choice.

Congratulations on lots of great work this fall !!
Have a great holiday break 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)