CSE 333 Final Exam 3/19/14

Name _____

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. _____ / 18
2. _____ / 19
3. _____ / 18
4. _____ / 12
5. _____ / 10
6. _____ / 10
7. _____ / 12
8. _____ / 1

Question 1. (18 points) The traditional C++ hacking question. Complete the function `HasDups` on the next page so it returns `true` (1) if its argument contains more than one copy of some string, and returns `false` (0) if all of the strings are different. For example, if the input list contains {"apple", "banana", "cherry", "banana", "kumquat" } the `HasDups` function should return true ("banana" appears more than once). If the input is {"apple", "banana", "cherry", "kumquat" } the function should return false.

For full credit, your function must scan the input list only once. You will need to use some other container to keep track of the words found in the list, but this must be a set, map, or some other container that provides operations that can check for duplicates in time faster than $O(n)$ (i.e., you cannot store the words in a new list and scan or sort the entire list looking for duplicates).

Hints and (incomplete) reference information:

- Remember that a STL `list` is a linked list underneath, so you must use iterators to scan it; you can't use `[]` subscripts to access individual list elements.
- If `lst` is a STL `list`, then `lst.begin()` and `lst.end()` return iterator values of type `list<...>::iterator` that might be useful.
- 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, including `list`:
  - `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
  - `c.push_back(x)` – copy x to end of c
  - `c.push_front(x)` – copy x to front of c
- Some additional operations on STL containers including `set` and `map`:
  - `c.insert(x)` – add copy of `x` to `c` (a key-value pair for a `map`, a value for a `set`)
  - `c.count(x)` – number of items in `c` with key `x` (returns only 0 or 1 for a `set`)
- 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.

(Write your answer on the next page. You may remove this page from the exam if you wish.)

Question 1. (cont.) Complete the definition of function `HasDups` below. Some useful `#include` directives as well as a `using` directive have been provided for convenience. You should add any additional libraries or code that you need.

```
#include <string>
#include <list>



using namespace std;

// Return true if the list words contains two or more
// copies of some string, otherwise return false.
bool HasDups(const list<string> &words)  {









}
```

Question 2. (19 points) The following header file defines a class that holds a pair of integers and includes a constructor and functions for accessing the values.

```
#ifndef _Pair_h_
#define _Pair_h_


class Pair {
 public:
  // Construct a Pair with given first and second values
  Pair(int first, int second)
      : first_(first), second_(second) { }


  // accessors: return first and second items from Pair
  int first()  const { return first_;  }
  int second() const { return second_; }


 private:
  // instance variables
  int first_;
  int second_;
};


#endif  // _Pair_h_
```

(a) (6 points) We would like to generalize this class so it can be used to store any pairs of values as long as the values have the same type (i.e., pairs of `int`s or pairs of `string`s, etc.)

Show the changes needed to make this a generic class where the element type is a type parameter instead of `int`. You should write your changes and additions in the above code. Hint: you'll need to start by adding `template <class T>` (or `template <typename T>`, which is equivalent) at the beginning of the class definition.

(continued on the next page)

Question 2. (cont.) We would now like to add an addition (+) operator to the generic `Pair` class on the previous page. If (a,b) and (c,d) are `Pair` values, then (a,b)+(c,d) should yield a new `Pair` containing (a+c, b+d). Neither of the original `Pair` objects should be modified. You do not need to check that addition (+) is defined on the items stored in a `Pair` – that is handled for you by the compiler when the addition operator is used.

(b) (5 points) Write the function declaration (not the implementation) to be added to the header file `Pair.h` for the new `operator+`.

(c) (8 points) Give the code to implement this new addition operator as it would appear in a separate file `Pair.cc` containing definitions of functions not implemented in `Pair.h`. Hint: the implementation needs to begin with `template <class T>`.

Question 3. (18 points) Is it real or virtual? Consider the following program. Function names are in **bold** text to make them easier to find. (You may remove this page to use while answering the questions on the next page.)

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

class Book {
 public:
  Book(string title) { title_ = new string(title); }
  virtual ~Book() { delete title_; }
  virtual string get_title() const { return *title_; }
          string get_description() const {
                   return "book: " + get_title(); }
  virtual string get_info() const {
                   return get_description() + " on main floor"; }
 private:
  string *title_;   // book title (on heap)
};

class Textbook : public Book {
 public:
  Textbook(string title, string subject) : Book(title) {
                                subject_ = new string(subject); }
  virtual ~Textbook() { delete subject_; }
          string get_description() const {
           return "textbook for "+ *subject_ +": "+ get_title();}
  virtual string get_info() const {
           return get_description() + " in the basement"; }
 private:
  string *subject_; // book subject area (on heap)
};
int main() {
  Book *b = new Book("C++ Reference");
  Book *t = new Textbook("C++ Primer", "CSE");
  cout << b->get_description() << endl;
  cout << b->get_info() << endl;
  cout << t->get_description() << endl;
  cout << t->get_info() << endl;

  delete b;
  delete t;
  return 0;
}
```

Question 3. (cont.) (a) (6 points) What output does this program produce when it is executed? (And it does execute and produce output.) Hint: be sure you look at what the code actually does, not what you think it should do. ☺

(b) (3 points) Are there any C++ errors when this program executes? Examples might include memory management bugs, object slicing on assignment, incorrect casts to the wrong type, etc. Give a brief, concise description of the problems or indicate that there are none.

(c) (6 points) Now suppose we edit this program and remove *all* occurrences of the word `virtual` from the code. When we recompile and execute it, what output does it produce? If it is the same as before you can just say so; if it is different, either write all of the output, or indicate *clearly* exactly how it is different.

(d) (3 points) When we execute this modified program with all of the `virtual`s removed, are there any hidden C++ errors this time? Again, give a very brief description.

Question 4. (12 points) A walk down memory lane. We have a small mystery program that prints out the addresses contained in four of its pointer variables. We tried running this program several times and each time it produced somewhat different output!

p = 0x1e21010, q = 0x000000, r = 0x7fff47d7f83c, s = 0x601060
p = 0x9b9010, q = 0x000000, r = 0x7fffe417e5fc, s = 0x601060
p = 0x11bc010, q = 0x000000, r = 0x7fff59b03dcc, s = 0x601060
p = 0x10c0010, q = 0x000000, r = 0x7fff6f61a75c, s = 0x601060
p = 0xd47010, q = 0x000000, r = 0x7fffca4adb3c, s = 0x601060

(a) (4 points) What's going on here? Why do we get different results when we run the same program repeatedly? It is exactly the same executable binary program run on the same system with no code changes, recompiling or relinking. So why the differences? (be brief, please)

(b) (8 points) We don't know for sure what the program does, but we do know that all of the pointers whose values are printed have type `int*`, and all of them have been initialized to point to a different region in the process address space if they point to anything at all. For each pointer, circle the answer that best describes the part of the process address space that the pointer value (address) references. Circle "can't tell" if there really isn't enough information to make a decision or at least a highly educated guess.

`p:`	stack	heap	global/static	NULL	can't tell
`q:`	stack	heap	global/static	NULL	can't tell
`r:`	stack	heap	global/static	NULL	can't tell
`s:`	stack	heap	global/static	NULL	can't tell

Question 5. (10 points) Oh, what tangled threads we weave. Consider the following small program, which compiles and executes without any errors. (`#includes` omitted to save space.)

```
int x = 0;

void * worker(void * ignore) {
  int y = 0;
  y = y + 1;
  x = x + y;
  printf("x = %d, y = %d\n", x, y);
  return NULL;
}
int main() {
  pthread_t t1, t2; int ignore;
  ignore = pthread_create(&t1, NULL, &worker, NULL);
  ignore = pthread_create(&t2, NULL, &worker, NULL);
  x = x + 1;
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  printf("final x = %d\n", x);
  return 0;
}
```

(a) (7 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.

(b) (3 points) Circle ***all*** of the possible values that variable `x` could have at the end of different executions, as printed in the final line of output:

0 1 2 3 4 5 6 or greater

Question 6. (10 points) A fork in the road. Now consider the following program that is basically the same as the previous one, except that it uses fork instead of pthreads for concurrency. (`#includes` omitted again to save space.)

```
int x = 0;

void worker() {
  int y = 0;
  y = y + 1;
  x = x + y;
  printf("x = %d, y = %d\n", x, y);
}
int main() {
  pid_t p1, p2; int ignore;
  p1 = fork();
  if (p1 == 0) {
    worker();
  } else {
    p2 = fork();
    if (p2 == 0) {
      worker();
    } else {
      x = x + 1;
      waitpid(p1, &ignore, 0);
      waitpid(p2, &ignore, 0);
      printf("final x = %d\n", x);
    }
  }
  return 0;
}
```

(a) (7 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.

(b) (3 points) Circle ***all*** of the possible values that variable `x` could have at the end of different executions, as printed in the final line of output:

0 1 2 3 4 5 6 or greater

Question 7. (12 points) Sockets and packets, oh my! Circle true or false for each of the following statements.

true false TCP guarantees reliable delivery of the packets that make up a stream, assuming that the socket doesn't fail because of an I/O error.

true false IP guarantees reliable delivery of packets, assuming that the socket doesn't fail because of an I/O error.

true false Given a particular hostname (like www.amazon.com), `getaddrinfo()` will return a single IP address corresponding to that name.

true false A single server machine can handle connection requests sent to multiple IP addresses.

true false The `listen` function returns a file descriptor number for a socket that a server can use to exchange data with the connected client machine.

true false A router stores information about the complete path needed to send a IP packet to its destination.

Question 8. (1 free point) What are you planning to do during Spring Vacation? (Circle all that apply)

a) Sleep

b) Do laundry

c) Clear out biology experiments that have grown in the refrigerator

d) Write code for projects that didn't get done because of classes this quarter

e) Party

f) Travel to a warm location and sit on the beach

g) Travel to a cold location and go skiing, snowboarding, or something similar

h) Catch up on missed episodes of _____

i) Read a book not related to computing, technology, or work

j) Put in extra hours at work

k) Spring break??? You're kidding!!!! I'm already behind in my spring classes!!

l) Other _____

m) I'm not telling you, but please give me my free point

n) I don't know yet, but please give me my free point

Whatever you're planning to do, have a great break!

The CSE 333 staff