CSE 333 24wi Final Exam 3/12/24

Name _____ UW netid _____

There are 7 questions worth a total of 110 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. However, you may have two 5x8 notecards or the equivalent with any hand-written notes you wish written on both sides.

There is a blank page at the end with extra space for your answers if you need more room. It is after all the questions but before the detachable sheet with reference information.

After the extra blank pages for answers, there are two pages of assorted reference information (much of which you probably won't need). You should remove this sheet of paper from the exam for convenience. These pages will not be scanned or graded.

Do not remove any pages from the middle of the exam.

If you do not 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 some 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 _____ / 110

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

Question 1. (20 points) A little C++/STL programming. One of our friends is taking a Music Appreciation course to fulfill some of their distribution requirements. To keep track of famous composers and the kinds of works they produced, they've decided to write a small C++ program. They've created a map data structure with composer names as the keys and the kinds of works written by each composer stored in a vector of strings as the values. A sample looks like this:

```
map<string, vector<string>> composers =
  { {"bach",      {"choral", "chamber", "keyboard"}},
    {"verdi",     {"choral", "opera", "chamber"}},
    {"beethoven", {"choral", "symphony", "opera", "chamber",
                   "keyboard"}},
    {"mahler",    {"symphony", "choral", "chamber"}}
  };
```

For this problem, implement the function `find_composers`. There are two arguments to this function: the map of composer information shown above, and a string that represents a kind of work (symphony, opera, etc.). The result of the function should be a vector of strings containing names of all of the composers who wrote that kind of work. The order of the composers in the output list does not matter. For example, if `composers` is the above data, then the function should produce these results:

```
find_composers(composers, "opera") => {"beethoven", "verdi"}
find_composers(composers, "symphony") => {"mahler", "beethoven"}
```

It could also produce lists that give the composer names in a different order.

If the specified kind of work does not appear anywhere in the composers data, then the function should return an empty vector.

You should assume that all necessary headers have already been `#included` (i.e., you do not need to worry about header files), and you can also assume that a `using namespace std;` directive has been written at the top of the file.

Reminder: there is some possibly useful reference information at the end of the exam.

Hint: you might, or might not, find it useful to write an auxiliary (helper) function that reports whether a particular string does or does not appear in a vector. Entirely up to you if you think it might be useful.

Write your answer on the next page…

Question 1. (cont.) Write your C++ code below. The function header is provided for you…

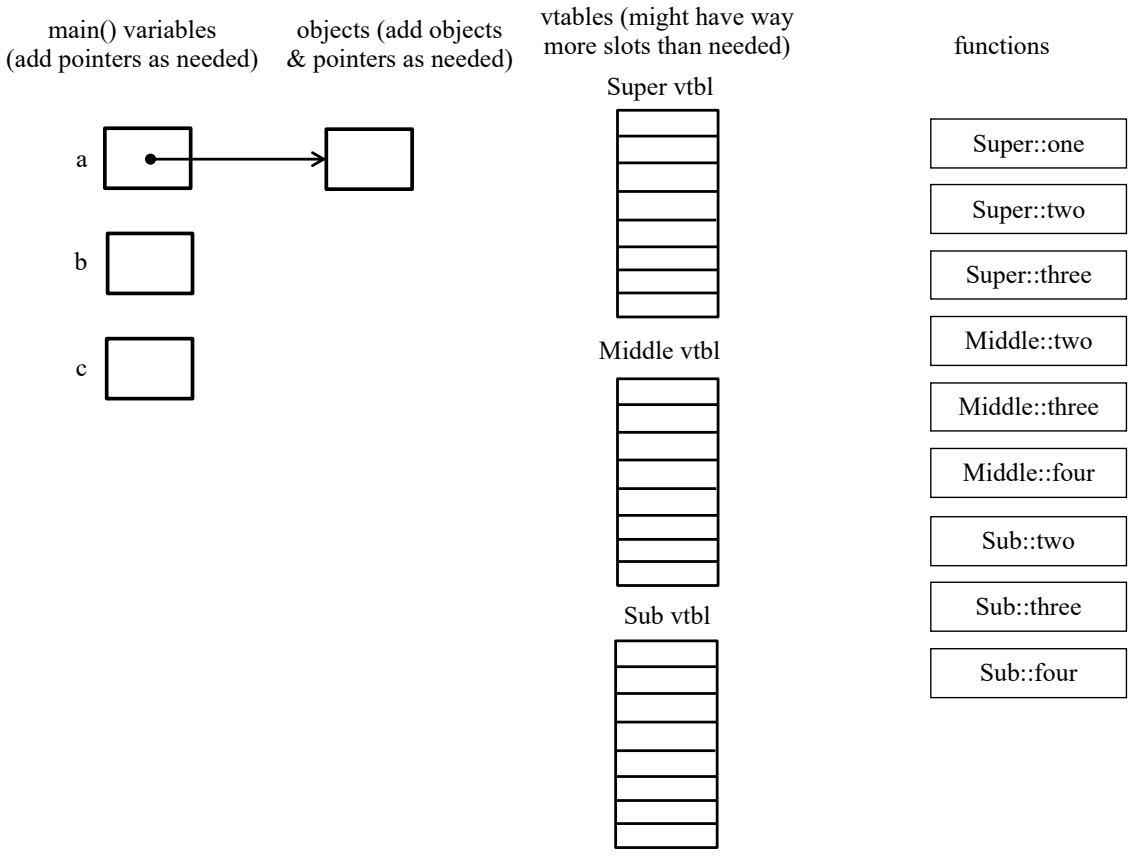
```
// return a vector containing names of composers whose
// compositions include works that match kind
vector<string> find_composers(
            map<string, vector<string>> composers, string kind) {




}
```

Question 2. (20 points) Yet another bonkers inheritance question. As usual this program compiles and executes with no errors. Headers and `using namespace std;` omitted to save space.

```
class Super {
public:
  virtual void one()   {        cout << "Super::one" << endl; }
          void two()   {        cout << "Super::two" << endl; }
  virtual void three() { two(); cout << "Super::three" << endl; }
};

class Middle: public Super {
public:
  virtual void three() {          cout << "Middle::three" << endl; }
          void four()  { two();   cout << "Middle::four" << endl; }
          void two()   { three(); cout << "Middle::two" << endl; }
};

class Sub: public Middle {
public:
          void two()   {        cout << "Sub::two" << endl; }
          void four()  { one(); cout << "Sub::four" << endl; }
          void three() {        cout << "Sub::three" << endl; }
};

int main() {
  Super *a = new Middle();
  a->one();
  a->two();
  a->three();
  cout << "---(1)---" << endl;
  Super *b = new Sub();
  b->one();
  b->two();
  b->three();
  cout << "---(2)---" << endl;
  Middle *c = new Sub();
  c->one();
  c->two();
  c->three();
  c->four();
  return 0;
}
```

Continue with the problem on the next pages. **Do not remove this page from the exam.**

Question 2. (cont.) (a) (6 points) Complete the diagram below to show all of the variables, objects, virtual method tables (vtables) and functions in this program. Parts of the diagram are supplied for you.



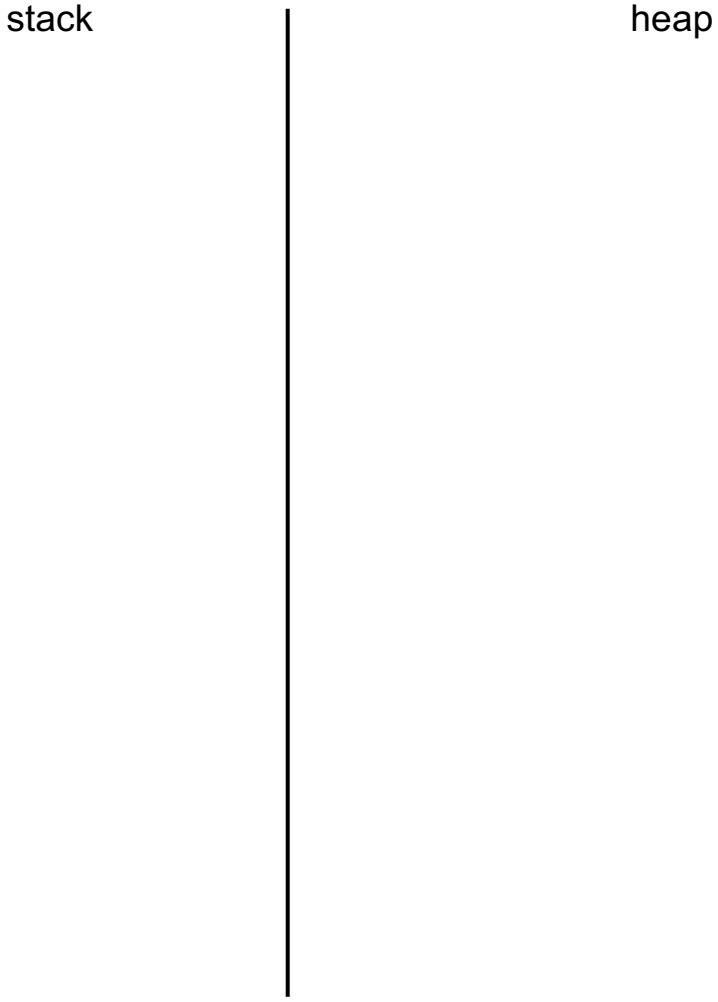
(b) (14 points) What does this program print when it executes? (write your answer in multiple columns if needed)

Question 3. (20 points) A trip down memory lane. A colleague who is new to C++ is learning how classes work by building a small class that has an integer instance variable (somewhat like the `Integer` wrapper class in Java). Here is the code:

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

// wrapped integer on the heap
class Int {
public:
  Int() {
    nptr_ = new int(0);
  }

  Int(int val) {
    nptr_ = new int(val);
  }

  virtual ~Int() {
    delete nptr_;
  }

  Int& operator=(const Int& rhs) {
    if (this != &rhs) {
      *nptr_ = *rhs.nptr_;
    }
    return *this;
  }

  int get() { return *nptr_; }

  void set(int newval) { *nptr_ = newval; }

private:
  int *nptr_;
};

int main() {
  Int x(17);
  Int y;
  Int z(y);
  y = x;
  z.set(42);
  //  Draw memory diagram when execution reaches here //
  cout << x.get() << " " << y.get() << " " << z.get() << endl;
  return 0;
}
```

(continued on next page)

Question 3. (cont.) (a) (10 points) This program does compile and executes successfully, at least for a while. Draw a diagram showing the contents of memory when execution reaches the line in the `main` function with the comment that says "draw memory when execution reaches **here**". Your diagram should clearly separate data on the stack from data allocated on the heap. Be sure to indicate clearly which data on the stack is part of the stack frame for function `main` by drawing a box labeled "main" with appropriate variables and data inside.



(b) (2 points) What output is produced when the next line is executed to print the values of integers x, y, and z?

(continued on next page)

Question 3. (cont.) (c) (3 points) There is some sort of error in this program and it does not finish execution successfully. What, precisely, is the error and what goes wrong?

(d) (3 points) How should we fix the error that you described above in part (c) that causes the program to crash? You should describe what to change in the code and give any specific replacement C++ code needed to fix the problem.

(e) (2 points) After fixing the error, what output does the program print when it is executed?

Question 4. (12 points) Templates. The code from the previous question is a wrapper for a single `int`(eger) value. There might be other cases where a simple wrapper like this would be useful for objects of other types, not just `int`.

Write your changes in the code below to modify the class **and** the main program that uses it so the class has a generic type parameter `T` for the type of the wrapped variable, and then modify the main program to use that template to create instances of the class with the actual type parameter `int`. Also, you should rename the class to be `Wrapper` instead of `Int`. The resulting program should work exactly the same as the original program, and will contain exactly the same bugs – don't fix those – but it should have a generic type for the wrapped value.

```
class Int {
public:
  Int() {
    nptr_ = new int(0);
  }

  Int(int val) {
    nptr_ = new int(val);
  }

  virtual ~Int() {
    delete nptr_;
  }

  Int& operator=(const Int& rhs) {
    if (this != &rhs) {
      *nptr_ = *rhs.nptr_;
    }
    return *this;
  }

  int get() { return *nptr_; }

  void set(int newval) { *nptr_ = newval; }

private:
  int *nptr_;
};

int main() {
  Int x(17);
  Int y;
  Int z(y);
  y = x;
  z.set(42);
  cout << x.get() << " " << y.get() << " " << z.get() << endl;
  return 0;
}
```

Question 5. (16 points) Yet another broken web server. Here is another attempt at creating a small web server, but, alas, it is buggy, like many of the ones we've been debugging over the last couple of weeks. But, like the ones we've been working on, with some fixes, this one can be repaired.

Write in corrections below to show how the code should be rearranged, changed, added to, or otherwise fixed to create a properly working server that accepts a connection from a client and then reads data from the client and writes it back until the socket is closed. Feel free to draw arrows showing how to move code around if needed, but be sure it is clear to the reader what you mean.

You should assume that all functions always succeed – ignore error handling for this question. Further, assume that the `hints` data structure is set up correctly for the call to `getaddrinfo` and the first address returned by `getaddrinfo` works and we don't need to search that linked list to find one that does work. Also, ignore the details of parameter lists – assume that all the "…" or missing parameters are valid and appropriate.

Also assume that `WrappedRead` and `WrappedWrite` are like the `read()` and `write()` POSIX functions, except that they automatically handle error conditions, including looping to retry the read or write if `EINTR` or `EAGAIN` errors occur.

Reminder: there is some potentially useful reference information at the end of the exam.

```
int main(int argc, char** argv) {

 struct addrinfo hints, *rp;

 memset(&hints, 0, sizeof(hints));

 hints.ai_... = ...;  // specify values for desired options

 getaddrinfo(NULL, argv[1], &hints, &rp);  // assume *rp is ok

 int fd = socket(rp->ai_family, rp->ai_socktype,

                                          rp->ai_protocol);

 setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, ...);

 freeaddrinfo(rp);

 accept(fd, SOMAXCONN);
```

(Continued on next page)

Question 5. (cont.) Continue writing in your corrections below as needed.

```
 char buf[BUFSIZE]  // BUFSIZE is a predefined constant
 // read data from the client and write it back until the
 // socket is closed
 while (1) {
   int r_res = WrappedRead(fd, buf, BUFSIZE);
   if (r_res == 0) {
     break;
   } else if (r_res < 0) {
     close(fd)
     return EXIT_FAILURE;
   }
   int w_res = WrappedWrite(fd, buf, r_res);
   if (w_res != r_res) {
     close(fd)
     return EXIT_FAILURE;
   }
 }
 close(fd);
 return EXIT_SUCCESS;
}
```

Question 6. (20 points) Too many things at once. Consider the following small program that uses pthreads. (This does compile and execute successfully.)

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

void *worker(void *ignore) {
  for (int k = 1; k <= 3; k++) {
    x = x + k;
    y = y + 1;
  }
  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\n", x);
  printf("final y = %d\n", y);
  return EXIT_SUCCESS;
}
```

When we run this program, it starts three threads that each assign values to `x` and `y` and then print the values of those variables. Then the program waits for all threads to finish, and prints the final values of the variables `x` and `y`.

(a) (4 points) What output would this program print if the threads are executed sequentially, not concurrently? In other words, what output is produced if thread 1 executes first and then thread 2 executes after thread 1 terminates and then thread 3 executes last?

(continued on next page)

Question 6. (cont.) (b) (6 points) When the threads run concurrently, is it possible to get different values for the numbers in the output if the program is executed multiple times? If it is, give two possible outputs that could be produced by the program that are different from the sequential output in part (a). If there is only one other possible set of output numbers besides the sequential one from part (a), write that one and indicate that it is the only possible different output.

(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 on a single line and separately from other output lines, and the variables being printed are read simultaneously with no other thread interrupting to change them while `printf` is running. If different executions lead to different output numbers, it is only because of the interaction between statements in the threads as they run concurrently.)

(c) (4 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 (you can draw a big circle around several values rather than individual small circles for each value if it helps save time ☺):

Possible final values for x:

0	1	2	3	4	5	6	7	8	9	10
	11	12	13	14	15	16	17	18	19	20 or more

Possible final values for y:

0	1	2	3	4	5	6	7	8	9	10
	11	12	13	14	15	16	17	18	19	20 or more

(continued next page)

Question 6 (cont.) (d) (6 points) Now we would like to add synchronization to the initial code so that the program still has three concurrent threads, but properly synchronized this time. In the copy of the original code below, insert appropriate locking using a single `pthread_mutex_t` lock so the threads do not interfere with each other as they update shared variables `x` and `y` but also so there is as much concurrency as possible. i.e., each thread should acquire the lock just long enough to avoid synchronization problems, but should not, for example, acquire the lock and hold on to it until it has completely finished, which would effectively cause the threads to execute one at a time rather than concurrently.

Hints: `pthread_mutex_t` and related mutex init, lock, and unlock functions are useful. (See the section at the end of the exam for additional reference information.)

```
int x = 0;
int y = 0;
void *worker(void *ignore) {
  for (int k = 1; k <= 3; k++) {
    x = x + k;
    y = y + 1;
  }
  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\n", x);
  printf("final y = %d\n", y);
  return EXIT_SUCCESS;
```

Question 7. (2 free points – all answers get the free points) Draw a picture of something you're planning to do over spring break!

Congratulations on lots of great work this quarter!!
Have a great spring break and best wishes for the future!
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. `s.find(`*search_string, start_pos* = 0`)` returns the location of the first occurrence of *search_string* starting no earlier than *start_pos* or returns `string::npos` if not found. 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.find`(*item*) – iterator pointing to element with key that matches *item* if found, or `m.end()` if not found.
  - `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(pthread_t *thread, attr, start_routine, arg)
- pthread_exit(void *status_ptr)
- pthread_join(pthread_t thread, void **value_ptr)
- pthread_cancel (pthread_t thread)
- pthread_detach(pthread_t 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)