CSE 333 25wi Final Exam 3/18/24

Name _____ UW netid _____@uw.edu
Please print legibly

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. _____ / 18
3. _____ / 14
4. _____ / 18
5. _____ / 18
6. _____ / 20
7. _____ / 2

Question 1. (20 points) A little C++/STL programming. The Seahawks, like most American football teams, have an extensive data analytics department, but for some reason they don't have anything that can keep track of the points scored by individual players. Since you've just finished CSE 333 and should be able to come up with a suitable program, they have asked for your help.

The program should accept input to do the following operations:

1. `add` *player points* : add an entry with the given number of points for the specified player. If the player already exists in the stored data (capitalization and exact spelling must match), append the new points to their existing list of points.
2. `display` : display the list of all players and the lists of points scored. The players can be listed in any order, but the scores for each player must be displayed in the order entered into the program.
3. `exit` : terminate the program

Example: Here is a sequence of input commands and the output produced by the program. The output is shown in *underlined italics* for illustration. Your program should not use any special fonts or styles.

```
add Smith 30
add Howell 28
add Smith 35
add Howell 25
add Williams 40
display
Smith: 30 35
Howell: 28 25
Williams: 40
exit
```

Since this is an exam question, you may go ahead and assume that input operations like reading strings or numbers will succeed. You do need to handle the error case where the input line starts with something other than `add`, `display`, or `exit`, and if that happens, just continue on to read the next command after producing a suitable error message on `cerr`. You do not need to worry about resynchronizing the input – just report the error and attempt to read the next input if there is additional data.

A skeleton program (starter code) is provided on the next page and you should fill in your code in appropriate places to complete the program. You may assume all necessary system library `#include` statements are already provided, but you do need to declare appropriate variables to hold the data as you read and process it.

(continued on next page)

Question 1. (cont.) Fill in the skeleton (starter) code below with your solution:

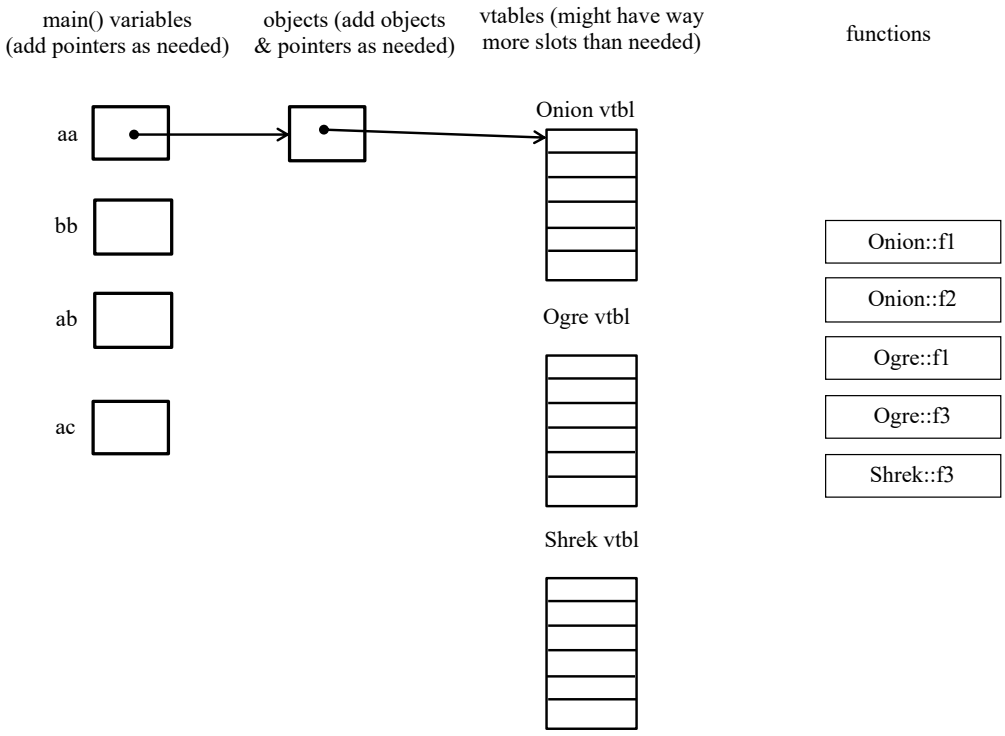
```
# assume all necessary libraries are #included
using namespace std;

int main() {
   // add variable declarations here




    while (true) {
        cin >> command;

        if (command == "add") {




        } else if (command == "display") {




        } else if (command == "exit") {



        }
        else {



        }
    }
    return _____________________;}
```

Question 2. (18 points) Yet another whacky inheritance question. As usual this program mostly compiles and mostly executes with no errors. Headers and `using namespace std;` omitted to save space.

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

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

class Shrek : public Ogre {
 public:
  virtual void f3() { cout << "Shrek::f3" << endl; }
};

int main() {
  Onion* aa = new Onion();
  Ogre* bb = new Ogre();
  Onion* ab = bb;
  Onion* ac = new Shrek();
  // HERE
  return EXIT_SUCCESS;
}
```

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

Question 2. (cont.) (a) (8 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) (10 points, 2 points each) For each of the following function calls, what happens if we replace the line HERE in the main program with this particular function call? If it compiles and executes successfully write down the output produced. If an error occurs, indicate what the error is and whether it is a compile-time or runtime error.

i) `aa->f2();`

ii) `ab->f2();`

iii) `ac->f3();`

iv) `bb->f1();`

v) `ab->f1();`

Question 3. (14 points) Smart pointers. C++ implementations cannot provide garbage collection because the language allows unrestricted casting and changing of pointer values. But the libraries do provide smart pointers, which can help with memory management when used appropriately.

Consider the following program that uses smart pointers (`#includes` and using namespace std omitted to save space)

```
class Resource {
public:
  Resource(int resourceId) : id_(resourceId) {
    std::cout << id_ << " ctor" << std::endl;
  }
  ~Resource() {
    std::cout << id_ << " dtor" << std::endl;
  }
private:
  int id_;
};  // end class Resource

int main() {
  std::unique_ptr<Resource> r1(new Resource(1));

  { //  start new scope
    std::cout << "-- 1 --" << std::endl;
    std::shared_ptr<Resource> r2(new Resource(2));
    std::shared_ptr<Resource> r3 = std::move(r1);

    { // start new scope
      std::cout << "-- 2 --" << std::endl;
      std::shared_ptr<Resource> r4 = r2;
      std::shared_ptr<Resource> r5(new Resource(3));
      r3 = nullptr;
    } // close scope

    std::cout << "-- 3 --" << std::endl;
    r2 = nullptr;
  } // close scope
  if (r1) {
    std::cout << "r1 is valid" << std::endl;
  } else {
    std::cout << "r1 is nullptr" << std::endl;
  }

  return EXIT_SUCCESS;
}
```

Write the output produced by this program on the next page.

Question 3. (cont) Below, write the output produced when we compile and execute the program on the previous page. The program compiles and executes without errors or memory leaks. (`#includes` and `using namespace std;` omitted to save space)

Question 4. (18 points) Templates. Consider the following program, which defines a template `Box` that stores a vector of items of some type `T`, then creates two of these `Box`es and stores some numbers and strings in them:

```
#include <iostream>
#include <vector>
#include <string>
#include <cstdlib>

template <typename T>
class Box {
 private:
  std::vector<T> items_;
 public:
  Box() { std::cout << "Generic box" << std::endl; }
  void add(T item) {
    items_.push_back(item);
    std::cout << "Added item" << std::endl;
  }
  T get(int i) { return items_[i]; }
  int size() const { return items_.size(); }
};

int main() {
  Box<double> b1;
  Box<std::string> bs;

  b1.add(3.14);
  b1.add(2.718);
  b1.add(-1.5);
  bs.add("hello cse333");

  return EXIT_SUCCESS;
}
```

Answer questions about this program on the following page.

Question 4. (cont.) (a) (6 points) What output is produced when we run the given program? (It does compile and execute with no errors). This answer is very simple.

(b) (12 points) The creators of the `Box` template did not provide any way to print the contents of a `Box`, although there are member functions that could be used to implement this. We would like to add a `Print_box` function, but with the constraint that we are not allowed to modify the original `Box` class template. Instead, we need to define a `Print_box<T>(b)` function template that will print the contents of a `Box b` whose elements have type `T`. You should give the definition of this function template below, and then show the output that is produced when we add `Print_box(b1);` and `Print_box(bs);` at the end of the original `main` function.

Footnotes: You should print the items to the standard `cout` stream and you may assume that the elements `x` of type `T` in the `Box` can be printed using `cout<<x`. Your output should include one space after each element printed, and should include a `endl` operation after the last item printed to end the output line.

Definition of `Box<T>` template:

Output produced by `Print_box(b1);` and `Print_box(bs);` when added to the end of `main`:

Question 5. (18 points) Client socket programming. One of your friends has been impressed with how you have been able to send and receive data over the network with relatively simple socket programming. They have decided to give it a try and have come up with the code on the next page.

Unfortunately, it doesn't quite work and they have come to you for help. For this problem, take a look at the code, find the problems, and fix them (there aren't that many). You should annotate the existing code both to explain the problem(s), and then cross out or rewrite code as needed to fix things. 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, you can assume that the `hints` data structure shown here will work with the call to `getaddrinfo`, that the first address returned by `getaddrinfo` works, and we don't need to search that linked list returned by `getaddrinfo` to find an address that does work.

Also assume that `WrappedRead()` and `WrappedWrite()`, if they appear in the code, work just like the `read()` and `write()` POSIX functions, except that they deal with error conditions, particularly looping to retry the read or write if `EINTR` or `EAGAIN` errors occur.

Also assume that all necessary `#include` headers are present for needed library files and that a `using namespace std;` declaration appears at the top of the code.

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

Fix the code on the next page. You can use the rest of this page for scratch space or for notes about your changes to the code if you wish.

If you need additional space, you can always continue work on the overflow pages at the end of the exam. Just be sure to indicate on the next page that your work is continued on the overflow pages, then label your work there so the graders can find it.

(Continued on next page)

Question 5. (cont.) Fix this code so it will connect to a server whose DNS name is given by `argv[1]` and port number given by `argv[2]`, write the string `argv[3]` to the server, then close the connection when that's done.

```
// #include headers omitted to save space
using namespace std;
int main(int argc, char **argv) {
  unsigned short port = 0;
  struct sockaddr_storage addr;
  size_t addrlen;
  struct addrinfo hints, *results;



  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;



  retval = getaddrinfo(argv[1], nullptr, &hints, &results);
  if (results->ai_family == AF_INET) {
    struct sockaddr_in *v4addr =
                    (struct sockaddr_in *)results->ai_addr;
    v4addr->sin_port = htons(port);
  } else if (results->ai_family == AF_INET6) {
    struct sockaddr_in6 *v6addr =
                     (struct sockaddr_in6 *) results->ai_addr;
    v6addr->sin6_port = htons(port);
  }

  int socket_fd = socket(AF_INET, SOCK_STREAM, 0);









  // write the string given as argv[3] to the server
  WrappedWrite(socket_fd, argv[3], strlen(argv[3])+1);

  return EXIT_SUCCESS;
}
```

Question 6. (20 points) Concurrency: bug or feature? Consider the following program that creates a pair of threads and executes them. Assume that all necessary `#include` headers are supplied – they are omitted here to save space. This program does compile and execute without errors.

```
// global variables
int arr[10] = {0}; // initialize all elements to 0
int idx = 0;

void *worker(void *arg) {
  int thread_id = *(int *)arg;
  for (int i = 0; i < 5; i++) {
    int curr_idx = idx;
    arr[curr_idx] += thread_id;
    idx = curr_idx + 1;
  }
  return NULL;
}

int main() {
  pthread_t t1, t2;
  int id1 = 1, id2 = 2;

  pthread_create(&t1, NULL, worker, &id1);
  pthread_create(&t2, NULL, worker, &id2);

  pthread_join(t1, NULL);
  pthread_join(t2, NULL);

  for (int i = 0; i < 10; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");

  return EXIT_SUCCESS;
}
```

When we run this program, it starts separate threads that each assign values to the global array and `idx` variables. Then the program waits for all threads to finish, and prints the final values of the array elements.

Answer

questions

about this code

on the next page….

Question 6. (cont.) (a) (4 points) What output would the program print if the threads were executed sequentially – first thread `t1` then thread `t2` – and not concurrently?

(b) (6 points) When the threads run concurrently, is it possible to get different output values for the array elements 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. 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) What are the possible minimum and maximum values of the individual elements of the array `arr` at the end of the concurrently executed program?

(d) (6 points) Now we would like to add synchronization to the original code so that the program still has concurrent threads, but properly synchronized this time. In the copy of the original code on the previous page, insert appropriate locking using a single `pthread_mutex_t` lock so the threads do not interfere with each other as they update shared variables, 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.)

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.

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 `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.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 `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.

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)