

CSE 333 Final Exam June 10₁₀, 11111011111₂

Name _____

There are 7 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. _____ / 20

5. _____ / 20

2. _____ / 13

6. _____ / 15

3. _____ / 16

7. _____ / 1

4. _____ / 15

CSE 333 Final Exam June 10, 111101111, 2

Question 1. (20 points) A bit of C++ hacking – STL version The question that is a lot longer than the answer. It's a social media world and Twitter wants you to digest some data that they have. Write a program that reads input from stdin. Each line in the input contains two strings: first, the name of a follower and second the name of a person they follow. So, for instance, an input line of text containing

```
bob  alice
```

means that “bob” follows “alice”. The output should contain one line for each person who has one or more followers and contain the person's name and number of followers. Example: if alice has 12 followers and ed has 3,415, the output should be:

```
alice  12
ed     3415
```

The names in the output should be sorted (which is the default order in which an iterator for maps access the data).

You should use a C++ STL map container to accumulate the data. Here is some brief reference information about maps:

- 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.
- As with any STL container, if `m` is a map, `m.begin()` and `m.end()` return iterator values that might be useful. 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 all STL containers, including map:
 - `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
- Some additional operations on maps:
 - `m.insert(x)` – add copy of `x` to `m` (a key-value pair for a map)
 - `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).
- 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 for reference while working if you wish.

CSE 333 Final Exam June 10, 1111011111₂

Question 1. (cont.) Write your program to read and summarize the Twitter follower data below. You do not need to write `#includes` – assume that these are already provided. You also may assume that input operations succeed until there is no more data. You do not need to do error checking beyond that. To simplify things, use standard C++ input and output (`cin` and `cout`).

CSE 333 Final Exam June 10₁₀, 11111011111₂

Question 2. (13 points) A template for your thoughts. One of our new interns, who knows a lot of Java but is only getting the hang of C++, has written a small program that uses templates. In good C++ fashion, the program has been broken into header, implementation, and client source files as follows:

Compare.h:

```
#ifndef _COMPARE_H_
#define _COMPARE_H_
template <class T>
int comp(const T& a, const T& b);
#endif // COMPARE_H_
```

Compare.cc:

```
#include "compare.h"
template <class T>
int comp(const T& a, const T& b) {
    if (a < b) return -1;
    if (b < a) return 1;
    return 0;
}
```

main.cc:

```
#include <iostream>
#include "compare.h"
int main(int argc, char **argv) {
    std::cout << comp(10, 20) << std::endl;
    return 0;
}
```

Unfortunately the code doesn't compile. (a) (5 points) what's wrong? (1 sentence)

(b) (8 points) Describe two different ways to fix the problem (1 sentence each)

CSE 333 Final Exam June 10, 111101111, 2

Question 3. (16 points) Classes ‘n things. Consider the following files on this page and the next, which model some household pets:

Pet.h:

```
#ifndef _PET_H_
#define _PET_H_

#include <string>
using namespace std;

class Pet {
public:
    Pet(string name): name_(new string(name)) { }
    string name() { return *name_; }
    string sound() { return "glorp"; }
    ~Pet() { delete name_; }
private:
    string *name_;
};

class Dog: public Pet {
public:
    Dog(string name): Pet(name) { }
    string sound() { return "Woof!"; }
    ~Dog() { }
};

class Cat: public Pet {
public:
    Cat(string name): Pet(name) { }
    string sound() { return "Meow!"; }
    ~Cat() { }
};

#endif // _PET_H_
```

(continued on the next page. You can remove this page if it is convenient while working on the problem)

CSE 333 Final Exam June 10, 11111011111₂

Question 3. (cont.) Here is the main program (main.cc) that uses the header file from the previous page:

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

void Print_Pet(Pet* critter) {
    cout << critter->name() << " says " <<
        critter->sound() << endl;
}

int main() {
    Pet* canine = new Dog("Buddy");
    Pet* feline = new Cat("Whiskers");
    Print_Pet(canine);
    Print_Pet(feline);
    delete canine;
    delete feline;
    return 0;
}
```

(a) (5 points) What does this program print when it is executed?

(b) (6 points) Is the output what the programmer probably intended? If so, explain why. If not, explain in a sentence or two what's wrong and how to fix it.

(c) (5 points) When we run this program, are there any crashes, memory leaks, or other problems except (possibly) what is printed as output? If so, how do we fix the problem? (Be brief. Hint: this is (almost) a trick question.)

CSE 333 Final Exam June 10₁₀, 11111011111₂

Question 4. (15 points) Pointy Things. Ben Bitdiddler is a Java programmer who really doesn't like C++ memory management (who does?). But he's discovered smart pointers and figures that they will be almost as pleasant to use as Java's garbage collection. He's put together a small test program using smart pointers to build a linked list. Here it is:

```
struct Node {
    string data;
    shared_ptr<Node> next;
    shared_ptr<Node> prev;
    // convenience constructor
    Node(string val, shared_ptr<Node> next, shared_ptr<Node>
          prev) : data(val), next(next), prev(prev) { }
};

int main() {
    shared_ptr<Node> head(new Node("bar", nullptr, nullptr));
    head = shared_ptr<Node>(new Node("foo", head, nullptr));
    head->next->prev = head;
    cout << head->data << endl;
    cout << head->next->data << endl;
    return 0;
}
```

(a) (5 points) What does this program print when it is run?

(b) (5 points) In spite of Ben's optimism, this program still leaks memory. Why?

(c) (5 points) What would be the right C++ way using smart pointers to fix the memory problem and why does it work?

CSE 333 Final Exam June 10₁₀, 11111011111₂

The next multi-part question concerns networking. Here is some reference information about some TCP, IP, and POSIX I/O functions and data structures that may be useful. You will not need to use all of these functions, however the ones you use should be used correctly. You may remove this page for reference while working.

- `int accept(int sockfd, struct sockaddr *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 for second argument
- `int listen(int sockfd, int backlog)`
 - Use SOMAXCONN for backlog
- `ssize_t read(int fd, void *buf, size_t count)`
- `int socket(int domain, int type, int protocol)`
 - Use SOCK_STREAM for type (TCP), 0 for protocol, get domain from address info struct
- `ssize_t write(int fd, const void *buf, size_t count)`

```
struct addrinfo {
    int          ai_flags;        // AI_PASSIVE, AI_CANONNAME, etc.
    int          ai_family;      // AF_INET, AF_INET6, AF_UNSPEC
    int          ai_socktype;    // SOCK_STREAM, SOCK_DGRAM
    int          ai_protocol;    // use 0 for "any"
    size_t       ai_addrlen;     // size of ai_addr in bytes
    struct sockaddr *ai_addr;    // struct sockaddr_in or _in6
    char         *ai_canonname;  // full canonical hostname
    struct addrinfo *ai_next;    // linked list, next node
};
```

```
struct sockaddr_storage {
    sa_family_t ss_family; // address family
    // Remaining bytes are padding that can be ignored
};
```

CSE 333 Final Exam June 10, 111101111, 2

Question 5. (20 points) One of the spring interns was working on a TCP client program and created most of the code. Unfortunately he left before he could figure out the actual TCP/IP code needed in a client to connect to a server.

(a) (12 points) Fill in the 4 blanks labeled MISSING in the client code below to complete the program so it connects to a server and exchanges data with it. The code uses asserts to check for errors to keep things short(er). The `TalkToServer` function that actually exchanges data with the server is covered in the next part of the question. In this part, just leave the call to `TalkToServer` in place, and don't analyze it here. Assume that all necessary header files are already `#included`. Hint: this code is not exactly the same as the socket code from class, but it does use network functions in the same way.

```
// local declarations
void Usage(char *progname);
void TalkToServer(int fd, std::string host, std::string path);

int main(int argc, char **argv) {
    unsigned short port = 0;
    int res, socket_fd;

    if (argc != 4 || sscanf(argv[2], "%hu", &port) != 1)
        Usage(argv[0]);
    char *name = argv[1];

    // Get an appropriate sockaddr structure.
    struct addrinfo hints, *results;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    // MISSING 1:

    res = _____;
    assert(res == 0 && results != NULL);
    // Set the port in the first result.
    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);
    } else {
        exit(EXIT_FAILURE);
    }
}
```

CSE 333 Final Exam June 10₁₀, 1111101111₂

Question 5. (cont.) Program code and part (a) continued. Fill in the blanks on this page as describe above.

```
// Store the first result.
struct sockaddr_storage addr;
memcpy(&addr, results->ai_addr, results->ai_addrlen);
size_t addrlen = results->ai_addrlen;

// Free the results
freeaddrinfo(results);

// MISSING 2: Create the socket.

socket_fd = _____;
assert(socket_fd != -1);

// MISSING 3:

res = _____;
assert(res != -1);

// Talk back and forth to the server
TalkToServer(socket_fd, argv[1], argv[3]);

// MISSING 4: clean up

res = _____;
assert(res != -1);
return EXIT_SUCCESS;
}
```

(remaining parts of the question on the next page)

CSE 333 Final Exam June 10, 111101111, 2

Question 5. (cont.) Here is the function used in the first part of the problem to communicate with the server.

```
void TalkToServer(int socket_fd, std::string host,
                  std::string path) {
    std::string str("GET " + path + " HTTP/1.1\r\n");
    str += "Host: " + host + "\r\n\r\n";
    int len = str.length();
    char buf[len + 1];
    strcpy(buf, str.c_str()); // copies str into buf

    // Really should be in a loop in case of EAGAIN, EINTR.
    // But less to read this way!
    int res = write(socket_fd, buf, len);
    assert(res > 0);

    while (1) {
        int res = read(socket_fd, buf, len);
        if (res == -1) {
            if (errno == EINTR || errno == EAGAIN)
                continue;
            close(socket_fd);
            exit(EXIT_FAILURE);
        }
        buf[res] = '\0';
        std::cout << buf;
    }
}
```

(b) (4 points) Examine the code for this function and in a couple of sentences describe what it does. Your description should be the sort of thing that would be included in an appropriate heading comment for `TalkToServer` that would tell someone what it does when called, including information about the significance of the function parameters.

(c) (4 points) Are there any programming/logic bugs in this code or does it work as intended? If there are problems, briefly describe what they are.

CSE 333 Final Exam June 10, 111101111,2

Question 6. (15 points) Concurrency. Consider the following program (#includes omitted to save space): (and *don't* detach this page)

```
// Global variables
static unsigned int total_count = 0;
static int counts[100];
static const unsigned int kNumThreads = 100;

// Increment total_count arg times. Executed by each thread.
void *thread_start(void *arg) {
    unsigned int thread_count = 0;
    unsigned int *loops = reinterpret_cast<unsigned int *>(arg);
    for (unsigned int i = 0; i < *loops; i++) {
        thread_count++;
    }
    total_count += thread_count;
    counts[*loops] = thread_count;
    return NULL;
}

int main(int argc, char **argv) {
    // Create kNumThreads threads, each executing thread_start
    pthread_t thr_array[kNumThreads];
    for (unsigned int i = 0; i < kNumThreads; i++) {
        unsigned int argument = i;
        if (pthread_create(&thr_array[i],
                          NULL,
                          &thread_start,
                          reinterpret_cast<void *>(&argument)) != 0) {
            std::cerr << "pthread_create() failed." << std::endl;
            return EXIT_FAILURE;
        }
    }
    // Join with all the threads we created
    for (unsigned int i = 0; i < kNumThreads; i++) {
        void *res;
        if (pthread_join(thr_array[i], &res) != 0) {
            std::cerr << "pthread_join() failed." << std::endl;
        }
    }
    // Print out the final counts and total count.
    for (unsigned int i = 0; i < kNumThreads; i++) {
        std::cout << "counts[" << i << "]=" << counts[i] <<std::endl;
    }
    std::cout << "total count is: " << total_count << std::endl;
    return EXIT_SUCCESS;
}
```

Answer questions about this code on the next page. You will want to leave this page in the exam to hand in as part of your answer to part (a).

CSE 333 Final Exam June 10₁₀, 11111011111₂

Question 6. (cont.) (a) (5 points) This program is not thread safe because of a missing lock or locks. In the code on the previous page, circle each line or set of lines of code where there should be a mutual exclusion lock, and write a brief explanation of why locking is needed there.

(b) (4 points) Even after adding locks in appropriate places, the above code is still not thread-safe. Explain what the problem is and suggest a way to resolve it.

(c) (6 points) Circle T (true) or F (false) for each of the following general statements about threads and processes. (These are not specific to the code examined in the previous parts of the problem.)

T F `fork()` is the only system call (that we've covered) that returns twice.

T F Creating threads is more expensive than forking processes.

T F When a process terminates, all of its child processes are terminated immediately.

T F Threads in a program share the same global variables.

T F When a process calls `fork()` the function value returned to the original (parent) process is 0.

T F If a process with an open file descriptor does a `fork()` and then does a `close()` on that file descriptor, that will close the file in both the original and child processes.

CSE 333 Final Exam June 10₁₀, 11111011111₂

Question 7. (1 free point) My favorite way of procrastinating instead of working on CSE 333 assignments is:

- a) Email / text messaging
- b) Talking on the phone
- c) Facebook
- d) YouTube
- e) General web surfing
- f) Homework for other classes
- g) Searching for answers on StackOverflow
- h) Exercise / spending time at the IMA
- i) Hiking, running, bicycling, or other outdoors activities
- j) Participating in team sports (which one? _____)
- k) Watching team sports
- l) Swimming in the Drumheller fountain
- m) Reading an ebook
- n) Reading a paper book
- o) Writing, but not for classes
- p) Arts and crafts
- q) Listening to music, either live or recorded (kind of music: _____)
- r) Participating in or producing music (kind: _____)
- s) Going to the movies
- t) Binge watching TV or web (name of series: _____)
- u) Unsupervised activities
- v) You missed it! My favorite way to procrastinate is _____
- w) I'm not going to tell you, but please give me my free point anyway
- x) Other:

*Have a great summer and best wishes for the future !!
The CSE 333 staff*