Question 1. (16 points) C++ STL. We have a C++ map that contains information about available flights between airports. Each key in the map is a string with a 3-letter airport code. The corresponding value is a vector of strings containing the codes of all other airports reachable directly from this airport. We will assume that if airport A appears in the destinations list for airport B then B will also appear in the destinations list for A. For example, the following map includes information that there is a flight from SEA to SFO and vice versa, but not from SEA to MSP or vice versa.

```
map<string, vector<string>> flights =
  { {"SEA", {"SFO", "JFK"}},
    {"JFK", {"SEA", "MSP", "SFO"}},
    {"MSP", {"JFK"}},
    {"SFO", {"SEA", "JFK"}}
  };
```

Complete the definition of function `path_exists` below. The arguments to `path_exists` are a flights table like the one above, and a vector of strings with a sequence of airport codes. The function should return true if it is possible to get from the first airport in the list to the last by taking direct flights between each pair of adjacent airports in the list. For example (abusing notation a little to write a vector value using curly braces), `path_exists(flights, {"SEA", "SFO", "JFK", "MSP"})` should return `true` since there are flights from SEA to SFO, from SFO to JFK, and from JFK to MSP, while `path_exists(flights, {"SEA", "MSP"})` should return false. You should assume there are at least two airport codes in the route list.

```
bool path_exists(map<string, vector<string>> flights,
                 vector<string> route) {
  auto src = route.begin(), dst = route.begin() + 1;
  for (; dst != route.end(); ++src, ++dst) {
    bool found = false;
    for (const auto &airport : flights[*src]) {
      if (airport == *dst) {
        found = true;
        break;
      }
    }
    if (!found)
      return false;
  }
  return true;
}
```

Question 2. (18 points) A somewhat(?) straightforward exercise involving classes. Consider the following program, which, as is customary, does compile and execute without errors. (Headers and `using namespace std;` omitted)

```
// point on 2-d plane
class Point {
 public:
  Point(): x_(0.0), y_(0.0) { cout << "Point() ctr" << endl; }
  Point(double x, double y): x_(x), y_(y)
                             { cout << "Point(x,y) ctr" << endl; }
  Point(const Point &other): x_(other.x_), y_(other.y_)
                             { cout << "Point copy ctr" << endl; }
  ~Point()
     { cout << "~Point(" << x_ << "," << y_ << ") dtr" << endl; }
  Point &operator=(const Point &other) {
    cout << "Point op=" << endl;
    if (this != &other) {
      x_ = other.x_; y_ = other.y_;
    }
    return *this;
  }
 private:
  double x_, y_;
};

// rectangle defined by upper-left and lower-right corners
class Rectangle {
 public:
  Rectangle()        { cout << "Rectangle() ctr" << endl; }
  Rectangle(const Point &ul, const Point &lr): ul_(ul), lr_(lr)
                     { cout << "Rectangle(ul,lr) ctr" << endl; }
  Rectangle(const Rectangle &other)
                                 : ul_(other.ul_), lr_(other.lr_)
                     { cout << "Rectangle copy ctr" << endl; }
  ~Rectangle() { cout << "~Rectangle dtr" << endl; }
  Rectangle &operator=(const Rectangle &other) {
    cout << "Rectangle op=" << endl;
    if (this != &other) {
      ul_ = other.ul_; lr_ = other.lr_;
    }
    return *this;
  }
 private:
  Point ul_, lr_;  // upper-left and lower-right corners
};
```

Remove this page from the exam, then continue with the problem on the next pages.
Do not write anything on this page. It will not be scanned for grading.

Question 2. (cont.) Now, here is a main program that uses these classes. At the bottom of the page, write the output produced when this program is executed.

```
int main() {
  Point p1;
  Point p2(1,4);
  cout << "---(1)---" << endl;
  Point p3 = p2;
  cout << "---(2)---" << endl;
  Rectangle r1;
  cout << "---(3)---" << endl;
  Point p4(2,3);
  Rectangle r2(p2,p4);
  cout << "---(4)---" << endl;
  r1 = r2;
  cout << "---(5)---" << endl;
  return 0;
}
```

Output:

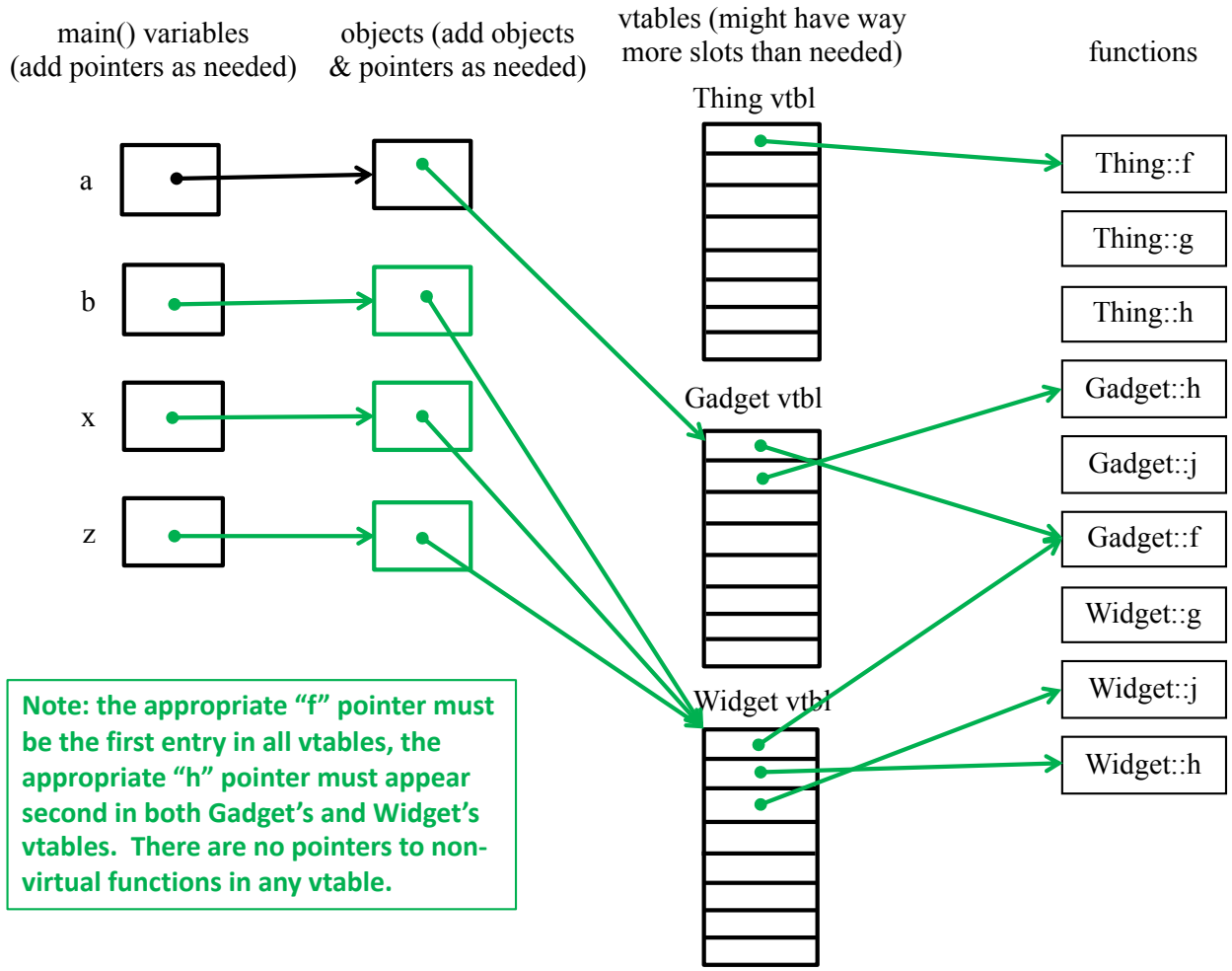
```
Point() ctr
Point(x,y) ctr
---(1)---
Point copy ctr
---(2)---
Point() ctr
Point() ctr
Rectangle() ctr
---(3)---
Point(x,y) ctr
Point copy ctr
Point copy ctr
Rectangle(ul,lr) ctr
---(4)---
Rectangle op=
Point op=
Point op=
---(5)---
~Rectangle dtr
~Point(2,3) dtr
~Point(1,4) dtr
~Point(2,3) dtr
~Rectangle dtr
~Point(2,3) dtr
~Point(1,4) dtr
~Point(1,4) dtr
~Point(1,4) dtr
~Point(0,0) dtr
```

Question 3. (24 points) Here we have another of those bizzare programs that seem to keep appearing on these exams with subclasses and virtual functions. As usual it does compile and execute with no errors.

```
class Thing {
public:
  virtual void f() {      cout << "Thing::f" << endl; }
          void g() {      cout << "Thing::g" << endl; }
          void h() { g(); cout << "Thing::h" << endl; }
};

class Gadget: public Thing {
public:
  virtual void h() {      cout << "Gadget::h" << endl; }
          void j() { g(); cout << "Gadget::j" << endl; }
          void f() { h(); cout << "Gadget::f" << endl; }
};

class Widget: public Gadget {
public:
          void g() {      cout << "Widget::g" << endl; }
  virtual void j() { f(); cout << "Widget::j" << endl; }
          void h() {      cout << "Widget::h" << endl; }
};

int main() {
  Thing *a = new Gadget();
  a->f();
  a->g();
  a->h();
  cout << "---(1)---" << endl;
  Thing *b = new Widget();
  b->f();
  b->g();
  b->h();
  cout << "---(2)---" << endl;
  Gadget *x = new Widget();
  x->f();
  x->g();
  x->h();
  x->j();
  cout << "---(3)---" << endl;
  Widget *z = new Widget();
  z->f();
  z->g();
  z->h();
  z->j();
  return 0;
}
```

Remove this page from the exam, then continue with the problem 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 all of the variables, objects, virtual method tables (vtables) and functions in this program. Parts of the diagram are supplied for you.



(b) (6 points) What does this program print when it executes? (use multiple columns if needed)

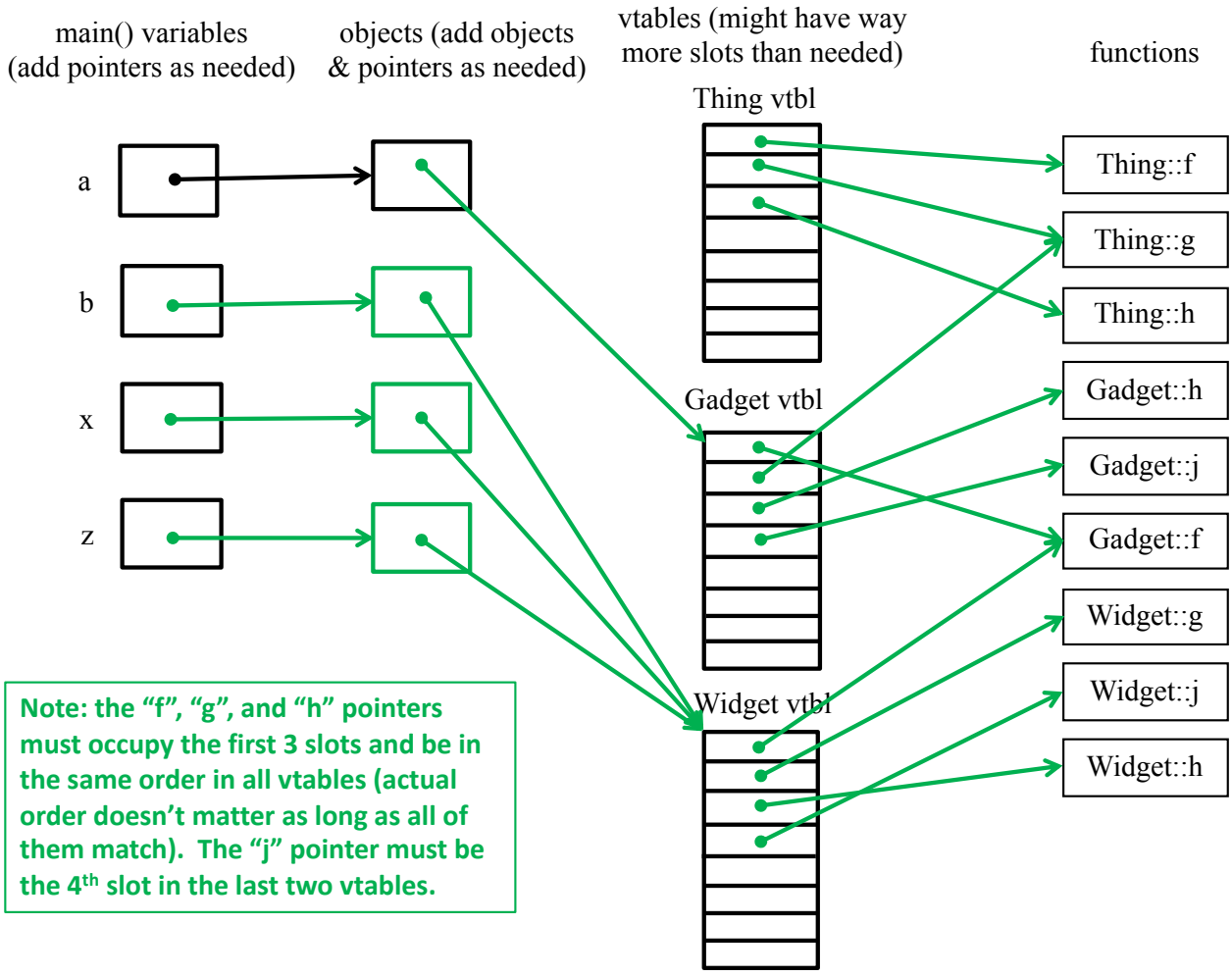
```
Gadget::h
Gadget::f
Thing::g
Thing::g
Thing::h
---(1)---
Widget::h
Gadget::f
Thing::g
```

```
Thing::g
Thing::h
---(2)---
Widget::h
Gadget::f
Thing::g
Widget::h
Thing::g
Gadget::j
```

```
---(3)---
Widget::h
Gadget::f
Widget::g
Widget::h
Widget::h
Gadget::f
Widget::j
```

(continued on the next page)

Question 3. (cont.) (c) (6 points) Now suppose we take the original program and place the word "virtual" in front of every function definition in every class. Complete the diagram below to show all of the variables, objects, virtual method tables (vtables) and functions in this altered program.



Note: the "f", "g", and "h" pointers must occupy the first 3 slots and be in the same order in all vtables (actual order doesn't matter as long as all of them match). The "j" pointer must be the 4th slot in the last two vtables.

(d) (6 points) What output does this altered program produce when it is executed? Again, use multiple columns if you need more room.

```
Gadget::h
Gadget::f
Thing::g
Gadget::h
---(1)---
Widget::h
Gadget::f
Widget::g
Widget::h
```

```
---(2)---
Widget::h
Gadget::f
Widget::g
Widget::h
Widget::h
Gadget::f
Widget::j
---(3)---
```

```
Widget::h
Gadget::f
Widget::g
Widget::h
Widget::h
Gadget::f
Widget::j
```

Question 4. (20 points, 4 each) Smart pointers. Below we have several small programs, each of which calls a function `foo`, and each of which uses smart pointers. Some of them are buggy, and some of them work correctly. Your job is to determine, for each program, the following:

(1) Does it compile?
(2) Is its behavior correct (i.e. no memory leaks, run-time errors, or undefined behavior)? Choose n/a if it did not compile.
(3) If you answered "no" to either of the above, explain concisely what the problem is and how to fix it.

You should assume that none of the `foo` functions will attempt to free, delete, or otherwise modify their argument. You can assume that the actual value returned by `main` is not relevant.

(a)
```
int foo(int *n);  // defined elsewhere

int main() {
  std::unique_ptr<int> best_course(new int(333));
  int ans = foo(best_course);
  return ans;
}
```

Does it compile? (circle) yes **(no)** not sure

Does it execute correctly (circle) yes no **(n/a (didn't compile)**

What is the problem (if any) and how do we fix it? (leave blank if not applicable)

`foo`'s parameter type is `int*` but the argument type is `unique_ptr<int>`. A fix is to use `best_course.get()` as the parameter.

(b)
```
int foo(std::shared_ptr<int> p);

int main() {
  std::unique_ptr<int> best_course(new int(333));
  int ans = foo(std::shared_ptr<int>(best_course.release()));
  return ans;
}
```

Does it compile? (circle) **(yes)** no not sure

Does it execute correctly (circle) **(yes)** no n/a (didn't compile)

What is the problem (if any) and how do we fix it? (leave blank if not applicable)

(continued next page)

Question 4. (cont.)

(c)
```
int foo(int, std::shared_ptr<int> p);

int main() {
  std::shared_ptr<int> best_course(new int[10]);
  int ans = foo(10, best_course);
  return ans;
}
```

Does it compile? (circle) (yes) no not sure

Does it execute correctly (circle) yes (no) n/a (didn't compile)

What is the problem (if any) and how do we fix it? (leave blank if not applicable)

The parameter type for the `shared_ptrs`, `<int>`, is not correct for an array. The `shared_ptrs` will do an ordinary `delete` on the array, but it must be a `delete[]`. The fix is to use `shared_ptr<int[]>` for the `shared_ptr` types.

(d)
```
int foo(std::unique_ptr<int> n);

int main() {
  std::unique_ptr<int> best_course(new int(333));
  int ans = foo(best_course);
  return ans;
}
```

Does it compile? (circle) yes (no) not sure

Does it execute correctly (circle) yes no (n/a (didn't compile)

What is the problem (if any) and how do we fix it? (leave blank if not applicable)

Function `foo` has a call-by-value parameter, and `unique_ptrs` cannot be copied. The solution is to use `shared_ptrs` or to use a reference parameter (&) for `foo`.

(e)
```
int foo(const int *p);

int main() {
  std::unique_ptr<int> best_course(new int(333));
  int ans = foo(best_course.release());
  return ans;
}
```

Does it compile? (circle) (yes) no not sure

Does it execute correctly (circle) yes (no) n/a (didn't compile)

What is the problem (if any) and how do we fix it? (leave blank if not applicable)

There is a memory leak. `best_course.release()` returns the raw pointer and the smart pointer is then no longer responsible for deleting the data. Either we need to explicitly delete the data, or use `get()` instead of `release()` to make a copy of the pointer while still letting the shared pointer delete the heap data later.

Question 5. (20 points) Networking – an insecure password server. We've been working on the prototype of a network authentication server. We're not ready to do any of that fancy encryption or security stuff yet – first we need to be able to read plain text usernames and passwords over the network and validate them.

The main part of the server code that accepts a connection from the client and closes the socket when done has already been implemented. The missing part is the logic to handle the client request. For this problem, implement the function `handle_client(cfd)` (on the next page), which reads a request from the client file descriptor, looks up the user name in a C++ map, and checks to see if the password in the network request matches the one found in the username / password data structure.

Specifically, your code needs to perform the following steps:

1. Read a request from the client socket. The request will have a maximum length of `REQ_MAX_LEN` bytes, although it might be less. You can allocate a buffer of this size for the input and not worry about excessive amounts of input. You *do* need to handle `EINTR` and `EAGAIN` results from `read` as usual.
2. The client request will have the form "*username*`\r\n`*password*`\r\n\r\n`" with no leading blanks. You need to parse the request and extract the *username* and *password* strings from it.
3. Check the password map to determine if the username and password match properly (i.e. the password is valid for that username). This is simple string matching.
4. Send a response back to the user. It should be either "`OK\r\n\r\n`" if the username and password match, or "`DENIED\r\n\r\n`" if the username is known but the password does not match the username, or, if the userame is not found, the response should be "`BADUSER\r\n\r\n`". Since you are using `write` to send the data to the client, you also need to worry about `EINTR` and `EAGAIN` results and retry until all of the data is written.

The username/password data is stored in a global variable named `passwd` that has the following declaration:

```
extern std::map<std::string, std::string> passwd;
```

The map keys are usernames and the associated values are the passwords. You should use this variable in your code without needing to declare it further.

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

Reminder: there are several possibly useful reference pages at the end of the exam.

(continued next page)

Question 5. (cont.) Give an implementation of the `handle_client` function below. The parameter `cfd` is the client socket file descriptor returned from the server's accept function, and this socket should be used to communicate with the client. Assume all necessary header files are already `#included`.

```
static const int REQ_MAX_LEN = 255; // max input request length
extern std::map<std::string, std::string> passwd;  // passwords

void handle_client(int cfd) {
    // Write your solution below

    // allocate enough space for the request + null terminator
    char reqbuf[REQ_MAX_LEN+1];
    std::string req;

    // step 1 - read from client with a POSIX socket read loop
    int to_read = REQ_MAX_LEN;
    int num_read = 0;
    while (to_read > 0) {
        int res = read(cfd, reqbuf + num_read, to_read);
        if (res == -1 && (errno == EINTR || errno == EAGAIN))
            continue;
        // Note: we return when res == 0, because the client has
        // disconnected and there is no longer any point in
        // handling their request
        if (res == 0 || res == -1)
            return;
        num_read += res;
        to_read -= res;
        // break when we've read the entire request
        reqbuf[num_read] = '\0';
        req = std::string(reqbuf);
        if (req.find("\r\n\r\n") != std::string::npos)
            break;
    }
    // step 2 - parse the request
    size_t b1 = req.find("\r\n");
    size_t b2 = b1 + 2;
    size_t b3 = req.find("\r\n", b1+2);
    // username\r\npassword\r\n\r\n
    //         ^   ^       ^
    //         b1  b2      b3
    std::string username = req.substr(0, b1);
    std::string password = req.substr(b2, b3 - b2);
    std::string response;
```

(continued on the next page)

Question 5. (cont.) Additional space for the answer. Do not remove this page.

```
    // step 3 - check the qdatabase
    auto result = passwd.find(username);
    if (result == passwd.end())
        response = std::string("BADUSER\r\n\r\n");
    else if (password != result->second)
        response = std::string("DENIED\r\n\r\n");
    else
        response = std::string("OK\r\n\r\n");

    // step 4 - write the response back
    int to_write = response.length();
    int written = 0;
    while (to_write > 0) {
        int res = write(cfd, response.c_str() + written,
                                                    to_write);
        if (res == -1 && (errno == EINTR || errno == EAGAIN))
            continue;
        if (res == 0 || res == -1)
            return;
        written += res;
        to_write -= res;
    }

}
```

Question 6. (20 points) Threads. Consider the following simple C++ program, which creates three threads. Each thread increments two global variables, then prints their values. After all three threads finish, the final values of the global variables are printed.

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

int x = 0;
int y = 0;

void * worker(void * ignore) {
  x = x + 1;
  y = y + x;
  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;
}
```

(a) (10 points) What output is produced by this program when it executes? If it is possible to get different output each time the program executes, write down two possible outputs. If the program always produces the same output, write that output here and indicate it is the only possible one. (Assume that `printf` executes atomically so that printing of output lines will not produce garbled output.)

Here are two possible outputs (obtained by running the program, no less). Other, even stranger output values are possible.

```
x = 1, y = 1
x = 2, y = 3
x = 3, y = 6
final x = 3, y = 6
```

```
x = 3, y = 6
x = 3, y = 9
x = 3, y = 3
final x = 3, y = 9
```

(b) (10 points) Circle *all* of the possible values that variables `x` and `y` could have at the end of any possible executions of this program, as printed in the final line of output:

x: 0 1 2 3 4 5 6 7 8 9 ≥10

y: 0 1 2 3 4 5 6 7 8 9 ≥10

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

Congratulations on lots of great work this quarter!!
Have a great spring break and say hello when you get back!
The CSE 333 staff