Question 1. (20 points) A little C++/STL programming. It's the time of year when many packages are being sent across the country and around the world. We'd like to help our friends at the corner grocery-post-office-minimart-cafe keep track of the destinations of the many packages they are collecting that need to be sent somewhere. Every time somebody shows up with new packages to be shipped, our friends at the store add a line to the file `boxes.txt` listing the destination and number of new packages for that destination. A sample input file might look something like this:

```
Seattle     3
Redmond     1
Seattle     1
Omaha       4
Boston      3
```

(You should assume that each destination is a single word like "Seattle" or "Los_Angeles", not words with embedded blanks like "San Francisco".)

Write a small C++ program that accepts the name of a file formatted above as its argument (`argv[1]`), opens the file, and reads its contents. Then, after reading the file, the program should print a sorted list of the destinations and total number of packages being sent to each destination. For the above sample input file, the output would be

```
Boston      3
Omaha       4
Redmond     1
Seattle     4
```

(Do not worry about the exact spacing of the output as long as each line contains the proper information.)

You do need to check that the file can be opened successfully, but, to keep the code simple for this exam problem, you can assume that when you read the destination strings and integer numbers using the simple C++ >> stream input operator that it will work, and you do not need to check for whether that succeeds or not.

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. It's also fine to write the whole program as a single `main` function if that makes sense, because it should be pretty short.

Hint: STL containers, particularly maps, might be very useful here….

Hint: remember that `ifstream f(`*filename*`, ifstream::in)` can be used to open the named file for reading, and you can test the stream variable `f` afterwards to see if the open succeeded by writing things like `if(f)...` or `if(!f)...`.

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

Write your answer on the next page…

Question 1. (cont.) Write your C++ code below.

Here is one possible solution. The error checking messages here are a bit more elaborate than needed or expected for an exam problem, but the program does need to check that it can successfully open the file.

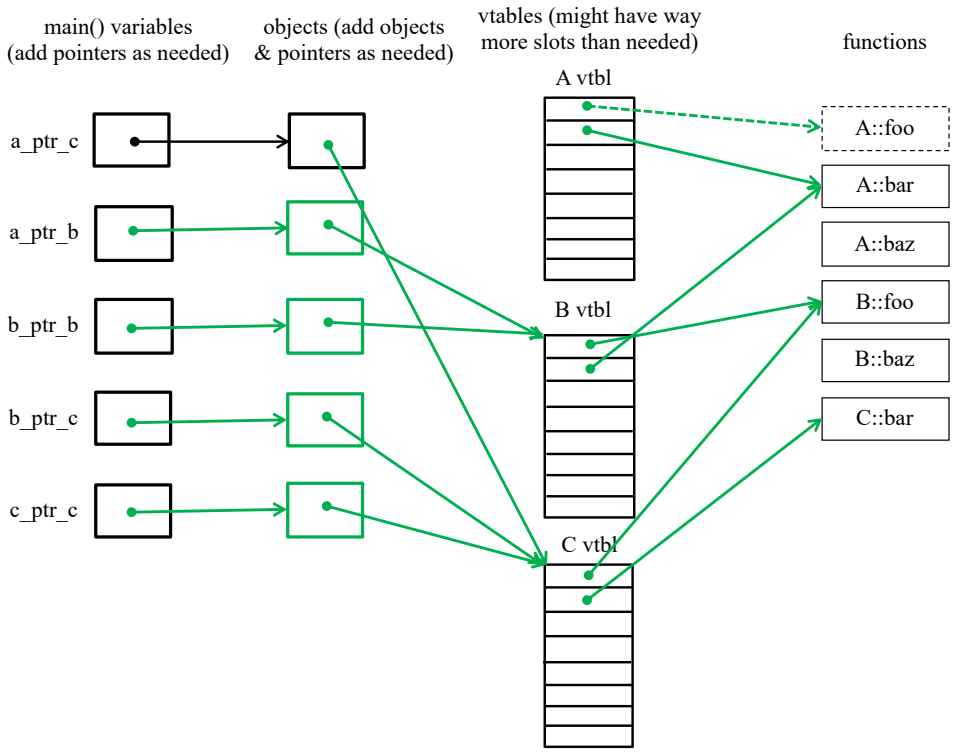
```
int main(int argc, char **argv) {
  // check that an argument is supplied
  if (argc != 2) {
    cerr << "usage: " << argv[0] << " filename" << endl;
return EXIT_FAILURE;
  }

  // Create stream and open for reading.  Exit if unable.
  std::ifstream f(argv[1], ifstream::in);
  if (!f) {
    cerr << "unable to open file: " << argv[1] << endl;
    return EXIT_FAILURE;
  }

  // read cities and numbers of packages from input file and
  // accumulate totals by city
  string city;
  int npackages;
  map<string, int> packages;   // map of <city, #packages> pairs
  while (f >> city) {
    f >> npackages;
    packages[city] += npackages;
  }

  // if input failed for some reason other than eof, report error
  // (not required for exam question, but included as an example)
  if (!f.eof()) {
    cerr << "error reading file: " << argv[1] << endl;
    return EXIT_FAILURE;
  }

  // print cities and number of packages sorted by city
  for (const auto &info : packages) {
    cout << info.first << " " << info.second << endl;
  }

  return EXIT_SUCCESS;
}
```

Question 2. (20 points) Here we go again – dynamic dispatch and friends. As usual this program compiles and executes with no errors. Headers and `using namespace std` omitted to save space.

```
class A {
public:
  virtual void foo() = 0;
  virtual void bar() { baz(); cout << "A::bar" << endl; }
          void baz() { cout << "A::baz" << endl; }
};

class B : public A {
public:
  void foo() { bar(); cout << "B::foo" << endl; }
  void baz() { cout << "B::baz" << endl; }
};

class C : public B {
public:
  void bar() { cout << "C::bar" << endl; }
};

int main() {
  A* a_ptr_c = new C();
  A* a_ptr_b = new B();
  B* b_ptr_b = new B();
  B* b_ptr_c = new C();
  C* c_ptr_c = new C();

  a_ptr_b->foo();
  b_ptr_b->baz();
  c_ptr_c->bar();

  cout << "---" << endl;

  a_ptr_c->bar();
  c_ptr_c->baz();
  b_ptr_c->bar();

  cout << "---" << endl;

  c_ptr_c->foo();
  b_ptr_b->foo();
  a_ptr_c->foo();
  return EXIT_SUCCESS;
}
```

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.



Notes: The original diagram was a bit misleading because A::foo is a pure virtual function, which means it will not be present in the program. It *is* assigned an offset in the vtable though so it can be properly called as a virtual function.

Functions foo and bar can be assigned in either order in the A vtable slots. They must appear in the same order in the subclass vtables.

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

```
A::baz
A::bar
B::foo
B::baz
C::bar
---
C::bar
B::baz
C::bar
---
```

```
C::bar
B::foo
A::baz
A::bar
B::foo
C::bar
B::foo
```

Question 3. (20 points) Memories…. of CSE 333 and other things. A colleague is trying to build a small C++ class that holds an array of integers plus the length of the array. Here is the code they've created so far.

//// Ray.h ////

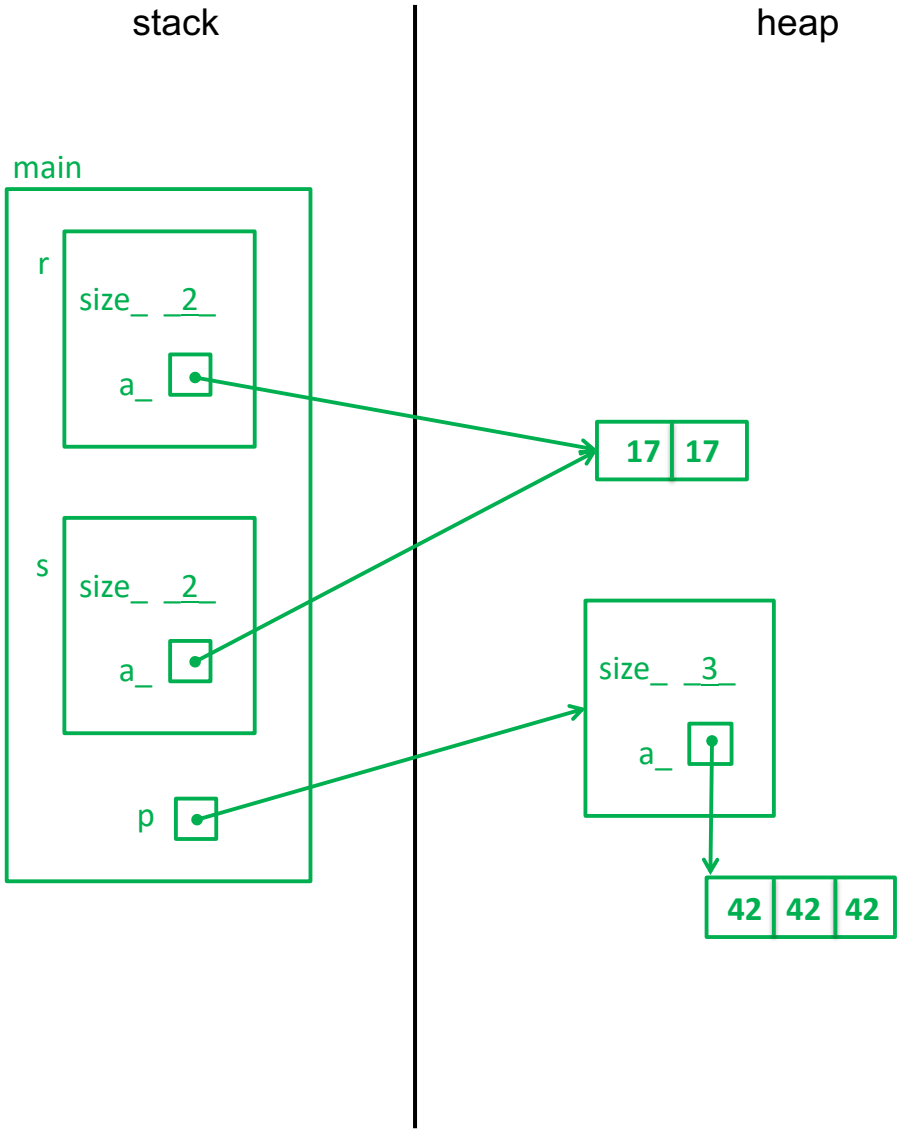
```
#ifndef RAY_H_
#define RAY_H_

#include <iostream>

class Ray {
 public:
  // construct new Ray with n copies of val
  Ray(int n, int val): size_(n), a_(new int[n]) {
    for (int i = 0; i < size_; i++) {
      a_[i] = val;
    }
  }
  // copy constructor and destructor
  Ray(const Ray &other) : size_(other.size_), a_(other.a_) { }
  ~Ray() { delete [] a_; }

  // print contents to stdout
  void Pr() {
    std::cout << "(";
    for (int k = 0; k < size_; k++) {
      std::cout << ((k > 0) ? "," : "") << a_[k];
    }
    std::cout << ")" << std::endl;
  }
 private:
  int size_;
  int *a_;  // elements stored in a_[0..size-1]
};
#endif // RAY_H_
```

//// main.cc ////

```
#include <cstdlib>
#include "Ray.h"

int main() {
  Ray r(2,17);
  Ray *p = new Ray(3,42);
  Ray s(r);
  // draw memory when execution reaches here //
  r.Pr();
  s.Pr();
  p->Pr();
  delete p;
  return EXIT_SUCCESS;
}
```

(continued on next page)

Question 3. (cont.) (a) (12 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.



Note: In diagrams like this it is important to indicate how related data is grouped into objects and into local variables belonging to some function on the stack. A number of solutions only showed free-floating data fields without using boxes to indicate how the data was grouped into objects, and that is not enough.

(continued on next page)

Question 3. (cont.) (b) (4 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?

When the program reaches the end of `main`, we get a double free error because the destructors for both `r` and `s` attempt to delete the same array allocated on the heap.

The cause is that the copy constructor for `Ray` does a shallow copy of the instance variables of the original object. When we use the copy constructor to create a new `Ray` object, both objects will point to the same array rather than correctly having their own private heap-allocated array containing the data.

(Note: the question had a typo in the destructor. `delete a_;` should have been `delete [] a_`. Solutions that noticed this but missed the double-free problem did receive some credit, but the major bug is the double free.)

(c) (4 points) How can we fix the error that you described above in part (b) that causes the program to crash? You should describe what to change in the code and give the specific replacement C++ code needed to fix the problem.

Fix the copy constructor for `Ray` so it allocates a separate array for the new object and initializes it with a copy of the data stored in the other object. One way to do this is to replace the original copy constructor with the following:

```
Ray(const Ray &other) : size_(other.size_),
                        a_(new int[other.size_]) {
  for (int i=0; i < size_; i++) {
    a_[i] = other.a_[i];
  }
}
```

Question 4. (12 points) Templates. The code from the previous question holds an array of `int`s and keeps track of the length of the array. Obviously, something like this could be useful for arrays of any type, not just `int`s.

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 `E` for the type of the array elements, and the main program uses that template to create instances of the class whose elements are type `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 array elements.

Necessary changes shown in bold green text.

```
// Ray.h //
#ifndef RAY_H_
#define RAY_H_
#include <iostream>
template <typename E>
class Ray {
 public:
  // construct new Ray with n copies of val
  Ray(int n, int E val): size_(n), a_(new int E[n]) {
    for (int i = 0; i < size_; i++) {
      a_[i] = val;
    }
  }
  // copy constructor and destructor
  Ray(const Ray &other) : size_(other.size_), a_(other.a_) { }
  ~Ray() { delete [] a_; }
  // print contents to stdout
  void Pr() {
    std::cout << "(";
    for (int k = 0; k < size_; k++) {
      std::cout << ((k > 0) ? "," : "") << a_[k];
    }
    std::cout << ")" << std::endl;
  }
```

(remainder of `Ray.h` and other code continued on next page)

Question 4. (cont.) Write the template changes needed on the rest of the code below.

```
//// remainder of Ray.h ////
 private:
  int size_;
  int E *a_;  // elements stored in a[0..size-1]
};

#endif // RAY_H_
```

```
//// main.cc ////
#include <cstdlib>
#include "Ray.h"
int main() {
  Ray r(2,17);
  Ray<int> *p = new Ray(3,42);
  Ray s(r);
  r.Pr();
  s.Pr();
  p->Pr();
  delete p;
  return EXIT_SUCCESS;
}
```

The `<int>` type is actually needed in the declaration of the pointer variable `p` in `main`. Without it, there is not enough type information available for C++ to figure out an appropriate type to substitute the `E` type parameter to create an correct type for the pointer variable `p`. This is a pretty subtle quirk of the C++ template type system, so we did not make a major deduction if it was missed.

It is possible, also, to include specific type parameters for variables `r` and `s` (`Ray<int> r(2,17)`), but the template type inference algorithm can determine this type from the constructor parameters used to create these variables, so the type does not need to be given explicitly.

Question 5. (15 points) Networking. One of our friends is teaching themselves network programming and is using CSE 333 examples to learn how to build a network server. Unfortunately, they are having some problems and their code doesn't quite work right. The core parts of the code are shown below.

Write in corrections below to show how the code should be rearranged, changed, or fixed to create a properly working server. 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.

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

Corrections shown in bold green text.

```
int main(int argc, char** argv) {
  struct addrinfo hints, *rp;
  memset(&hints, 0, sizeof(hints));
  // Set up the hints...
  hints.ai_... = ...;    // specify values for options

  // Get local socket and create it
  getaddrinfo(NULL, argv[1], &hints, &rp); // assume *rp works
  // Bug 1: need to retain fd returned by socket
  int fd_1 = 0;
  int fd_1 = socket(rp->ai_family, rp->ai_socktype,
                                     rp->ai_protocol);

  // Bind socket and set to listen
  int optval = 1;
  int sock_fam = rp->ai_family;
  // Bug 2: undeclared (wrong) fd used in setsockopt
  setsockopt(listen_fd  fd_1, SOL_SOCKET, SO_REUSEADDR,
                                    &optval, sizeof(optval));

  // Bug 3: need to bind before listen - swap those statements
  bind(fd_1, rp->ai_addr, rp->ai_addrlen);
  listen(fd_1, SOMAXCONN);

  // Bug 4: memory leak.  Need to free getaddrinfo result
  freeaddrinfo(rp);
```

(code continued on next page)

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

```
  // Accept clients and interact with them
  while (1) {
    int fd_2 = accept(fd_1, ...);
    // talk to client as needed
      // Bug 5: need to use fd from accept to talk to client
      read(~~fd_1~~ fd_2);
      write(~~fd_1~~ fd_2);
    // Bug 6: closing listener fd after handling only one client
    // (should be done on server shutdown only).  This probably
    // should be close(fd_2) to close the client fd before
    // accepting the next client connection reusing the same fd
    // variable.
    close(~~fd_1~~ fd_2);
  }
  return EXIT_SUCCESS;
}
```

Question 6. (18 points) Too many things at once. Consider the following small program that uses pthreads. (This does compile and execute successfully.)

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

int x = 0;
int y = 0;

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

When we run this program, it starts two threads that each assign values to `x` and `y` and prints the values of those variables, then waits for both threads to finish, and then 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?

```
x = 0, y = 1
x = 1, y = 2
final x = 1, y = 2
```

(continued on next page)

Question 6. (cont.) (b) (8 points) When the threads run concurrently, is it possible to get different output when the program is executed repeatedly? If it is, give three possible outputs that could be produced by the program. If there are only one or two possible outputs, write those and indicate that they are the only possible results. If one of the outputs is the same as in your answer to part (a) when the program runs sequentially, you can include that answer again here.

(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 outputs it is only because of the interaction between statements in the threads the threads as they run concurrently.)

Yes, there are many possible outputs. The simplest case is when the thread printf statements execute in a different order by switching between threads after updating the variables but before printing the formatted output strings:

```
x = 0, y = 1                x = 1, y = 2
x = 1, y = 2                x = 0, y = 1
final x = 1, y = 2          final x = 1, y = 2
```

More interesting cases happen when the system switches between threads when they are in the middle of updating variables. Here are a couple of possibilities:

```
x = 0, y = 1                x = 0, y = 1
x = 0, y = 2                x = 0, y = 1
final x = 0, y = 2          final x = 0, y = 1
```

(c) (6 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:

Possible final values for x:

(0) (1) 2 3 4 5 6 7 8 9 10 or more

Possible final values for y:

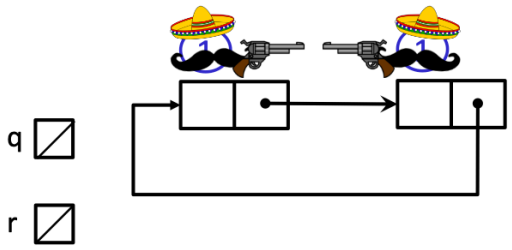
0 (1) (2) 3 4 5 6 7 8 9 10 or more

A couple of short-answer questions to finish up. Keep your answers brief and to the point. It should be possible to answer these questions with a few sentences each.

Question 7. (4 points) The C++ standard libraries include smart pointers which automatically delete owned ("pointed-to") objects when the smart pointer is deleted. If used extensively these can help avoid all kinds of memory management problems, particularly memory leaks and dangling pointers. Could we use C++ smart pointers to automate all of our memory management, the same as is done by the automatic garbage collector in Java? Give short technical explanation of why or why not.

No. We want to be able to build complex data structures and for that we will need `shared_ptrs`. These use reference counting to determine when to delete the "pointed-to" objects that they own.

But if there are cycles in the data structures, we can wind up with heap items all of which have non-zero reference counts, so they cannot be reclaimed, but which are not reachable or usable from the active parts of the program. These represent memory leaks.



Question 8. (4 points) When we were exploring strategies for implementing concurrent servers, we looked at the relative costs of concurrent threads compared to processes. On most systems, processes seem to be at least an order of magnitude more expensive to create than threads. Why is this so? What is it about creating a process that is significantly more expensive than creating a new thread? (Be brief!)

When we create a new thread, all we are doing is adding a new stack and execution context (mostly another set of registers) to an existing process. The new thread uses the existing process resources, including open files, memory, and operating system data structures describing the process, among other things.

To create a new process, we need to clone the existing process, create a new address space, and duplicate the process resources (code, open file descriptors, other process description information inside the operating system, contents of virtual memory, and so forth). Although good engineering can reduce the overhead needed to do this, forking a new process requires significantly more work than just adding a new thread to an existing one.

Question 9. (2 free points – all answers get the free points) Draw a picture of something you're planning to do over winter break!

Congratulations on lots of great work this quarter!!
Have a great holiday break and best wishes for the new year!
The CSE 333 staff