CSE 333 22sp Final Exam 6/8/22

Name _____ ID # _____

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. 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 pages 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 those pages 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 _____ / 100

1. _____ / 15
2. _____ / 18
3. _____ / 20
4. _____ / 18
5. _____ / 14
6. _____ / 14
7. _____ / 1

Question 1. (15 points) C++ STL. In these days of product shortages, it's hard to keep track of what is currently available where. We have a C++ `map` that contains information about stores and products currently available at those stores. The keys in the map are strings giving store names. The associated values are lists of items available in those stores. For example:

```
map<string, vector<string>> inventory =
  { {"qfc", {"bread", "milk", "bananas", "soap"}},
    {"bartells", {"soap", "toothpaste"}},
    {"safeway", {"milk", "bread", "bananas", "hotdogs"}}
  };
```

We would like to write a function that uses this information to produce a map whose keys are the items (values) from the above data and whose values are a set of stores where those items are available. The first few entries in the result produced from the above data might be the following:

```
{ {"bread",   {"qfc", "safeway"}},
  {"soap",    {"bartells", "qfc"}},
  {"hotdogs", {"safeway"}},
  ... }
```

The keys in the maps and the store names in the result sets can be in any order – your code does not need to guarantee any particular ordering of these.

Complete the definition of function `find_stores` on the next page so it produces a map of products and stores where they are available from an input map of stores and products that are available in those stores. Although the code is fairly short, please write it on the next page to ensure there is plenty of space for your answer, rather than trying to fit it at the bottom of this page.

(Note: since the store names are keys in the original map, we are guaranteed that they are unique, so we do not have to worry about duplicate values in the result sets.)

(Reminder: reference information about STL containers is included in the tear-off pages at the end of the exam.)

Write your

Answer

On the

Next

Page…

Question 1. (cont.) Complete the definition of function `find_stores` below. You should assume all necessary headers have already been `#include`d, and that a `using namespace std;` directive has been supplied so you do not need to write `std::` in front of all standard library names. You almost certainly will not need all of this space.

```
// return a map from product names to sets of stores where those
// products are available
map<string, set<string>> find_stores(
                    map<string, vector<string>> inventory) {
// write your implementation below




}
```

Question 2. (18 points) Templates and things. The following header file defines a class that holds a pair of integers and includes a constructor and functions for accessing the values.

```
#ifndef PAIR_H_
#define PAIR_H _


class Pair {
 public:
  // Construct a Pair with given first and second values
  Pair(int first, int second)
      : first_(first), second_(second) { }

  // accessors: return first and second items from Pair
  int first()  const { return first_;  }
  int second() const { return second_; }

 private:
  // instance variables
  int first_;
  int second_;
};

#endif  // PAIR_H _
```

(a) (5 points) We would like to generalize this class so it can be used to store any pairs of values as long as both values have the same type (i.e., pairs of `int`s, pairs of strings, etc.)

Show the changes needed to make this a generic class where the element type is a type parameter instead of `int`. You should write your changes and additions in the above code.

(continued on the next page)

Question 2. (cont.) We would now like to add an addition (+) operator to the generic `Pair` class on the previous page. If (a,b) and (c,d) are `Pair` values, then (a,b)+(c,d) should yield a new `Pair` containing (a+c, b+d). Neither of the original `Pair` objects should be modified. You do not need to check whether addition (+) is defined on the items stored in a `Pair` – that is handled for you by the compiler when the addition operator is used. The compiler will check that the actual type used when the template is instantiated supports addition and produce appropriate error messages if it does not.

(b) (5 points) Write the function declaration (not the implementation) to be added to the header file `Pair.h` for the new `operator+`. If it is possible to add `operator+` as either a member or non-member function of the `Pair` class, you can use whichever one you prefer.

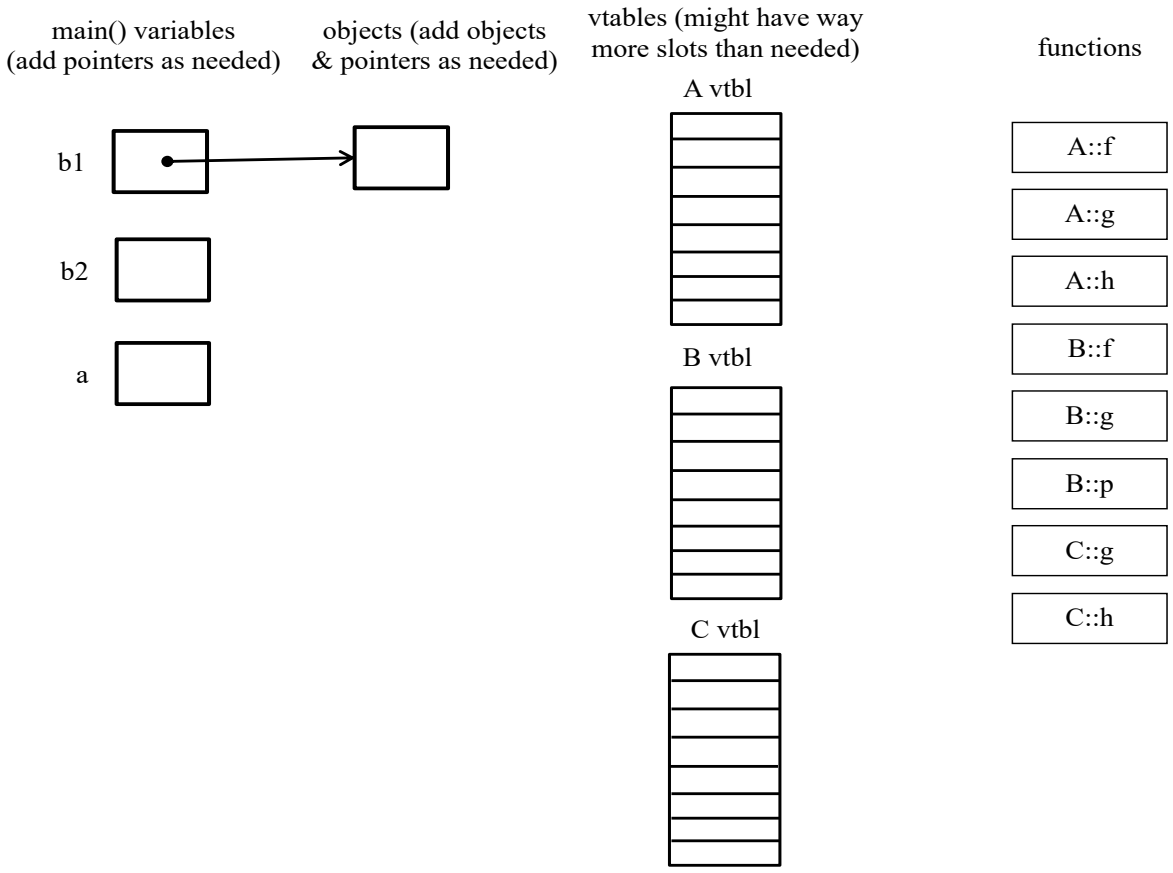
(c) (8 points) Give the code to implement this new addition operator as it would appear in a separate file `Pair.cc` containing definitions of functions not implemented in the header `Pair.h`.

Question 3. (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:
          void f() { h(); cout << "A::f" << endl; }
          void g() {      cout << "A::g" << endl; }
  virtual void h() {      cout << "A::h" << endl; }
};

class B: public A {
 public:
          void f() {      cout << "B::f" << endl; }
  virtual void g() { h(); cout << "B::g" << endl; }
          void p() { g(); cout << "B::p" << endl; }
};

class C: public B {
 public:
          void g() { f(); cout << "C::g" << endl; }
  virtual void h() { p(); cout << "C::h" << endl; }
};

int main() {
  cout << "--part 1--" << endl;
  B* b1 = new B();
  b1->f();
  cout << "---" << endl;
  b1->g();
  cout << "---" << endl;
  b1->h();
  cout << "---" << endl;
  b1->p();
  cout << "--part 2--" << endl;
  B* b2 = new C();
  A* a = b2;
  a->f();
  cout << "---" << endl;
  a->g();
  cout << "--part 3--" << endl;
  b2->f();
  cout << "---" << endl;
  b2->g();
  return 0;
}
```

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

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

Question 4. (18 points) More C++ classes. Consider the following program, which contains a simple integer wrapper class `Int` and a main function that uses it. Header `#includes` and `using namespace std;` omitted to save space. In the box on the right, write the output that is produced when this program is run. It compiles and executes with no errors.

```
class Int {
 public:
  // Constructors & destructor
  Int()                 { cout<<"Int()"  <<endl; val_=17; }
  Int(int n)            { cout<<"Int(n)" <<endl; val_=n;  }
  Int(const Int &other){cout<<"copy ctr"<<endl; val_=other.val_;}
  ~Int()                { cout<<"dtr"<<endl; }

  // accessor function
  int get() { cout<<"Int.get"<<endl; return val_; }

  // assignment
  Int &operator=(const Int& rhs) {
    cout<<"Int.op="<<endl;
    if (this == &rhs) return *this;
    this->val_ = rhs.val_;
    return *this;
  }
 private:
  int val_;   // instance variable
};  // end of class Int

int IntSum(Int v[], int sz) {
  int result = 0;
  for (int k = 0; k < sz; k++) {
    result += v[k].get();
  }
  return result;
}

int main() {
  Int x = 10;
  Int y = x;
  Int a[2];
  cout << "---" << endl;
  a[0] = 42;
  cout << "---" << endl;
  int sum = IntSum(a, 2);
  cout << "sum = " << sum << endl;
  cout << "---" << endl;
  Int* p;
  p = &x;
  int n = p->get();
  cout << "p->get() = " << n << endl;
  return EXIT_SUCCESS;
}
```

Program output:

Question 5. (14 points) Networking. Below is the pseudo-code for a very simple TCP server that accepts connections from clients and exchanges data with them. However, this code doesn't work because it has structural errors. In particular, some functions are called at the wrong time or in the wrong place, and there may be other problems. Write in corrections below to show how the pseudo-code should be rearranged, changed, or fixed to have the proper structure for a simple server. Feel free to draw arrows showing how to move code around, but be sure it is clear to the grader what you mean.

You should assume that all functions always succeed – ignore error handling for this question. Further, assume that 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 "…" parameters are valid and appropriate.

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 options
  getaddrinfo(NULL, argv[1], &hints, &rp);

  // ok to assume *rp is a valid address and will work here
  int fd = socket(rp->ai_family,
                  rp->ai_socktype, rp->ai_protocol);
  setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, ...);
  freeaddrinfo(rp);
  while (1) {

    bind(fd, rp->ai_addr, rp->ai_addrlen);

    fd = accept(fd, ...);

    listen(fd, SOMAXCONN);

    // talk to client as needed
      read(fd, ...);
      write(fd, ...);
    close(fd);
  }
  return EXIT_SUCCESS;
}
```

Question 6. (14 points, 2 each) Too many things at once… Consider the following C program using pthreads, which does compile and execute without errors (headers omitted to save space):

```
int x = 0;

void * thread_worker(void * ignore) {
  x = x + 1;
  printf("x=%d\n", x);
  return NULL;
}

int main() {
  pthread_t t1, t2;
  int ignore;
  ignore = pthread_create(&t1, NULL, &thread_worker, NULL);
  ignore = pthread_create(&t2, NULL, &thread_worker, NULL);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  printf("final x=%d\n", x);
  return EXIT_SUCCESS;
}
```

For each of the following output sequences, circle "yes" if it could be produced by some possible execution of the above program and circle "no" if it could never happen under any circumstances. (Hint: at least one of these sequences is possible.)

YES NO `x=0 x=1 final x=1`

YES NO `x=1 x=1 final x=1`

YES NO `x=1 x=2 final x=1`

YES NO `x=1 x=1 final x=2`

YES NO `x=1 x=2 final x=2`

YES NO `x=2 x=1 final x=2`

YES NO `x=2 x=2 final x=2`

Question 7. (1 free point – all answers get the free point) Draw a picture of something you plan to do this summer!

Congratulations on lots of great work this quarter!!
Have a great summer!
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 `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(thread, attr, start_routine, arg)
- pthread_exit(status)
- pthread_join(thread, value_ptr)
- pthread_cancel (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)