12sp

Problem 1: Multiple Choice Madness (24 points)

Circle exactly one answer for each of the following questions:

i. It is possible for C code to determine the endian-ness of the underlying CPU.

a) true b) false

ii. In C, a pointer is a variable that contains an address. If you add 2 to a pointer, then:

a) the resulting value is the address plus 2

b) the resulting value depends on what value the pointer points to

c) the resulting value depends on the type of the pointer

d) a segmentation fault is thrown

iii. When you pass a struct as an argument to a C function, then:

a) the struct is passed by value (i.e., a copy of the struct is made, including copying each field in the struct))

b) the struct is passed by reference (i.e., a pointer to the struct is passed)

c) a compiler error is thrown, since you cannot pass structs as arguments

d) what happens depends on the type of fields in the struct

iv. When you pass an array as an argument to a C function, then:

a) the array elements are passed by value (i.e., a copy of the array is made, including copying each element of the array)

b) since arrays are really just pointers, a pointer to the first element of the array is passed and no array elements are copied

c) a compiler error is thrown, since you cannot pass arrays as arguments

d) what happens depends on the type of the array

v. The purpose of a header guard is to:

a) prevent more than one .c file from including a particular .h file

b) prevent the header file from being included indirectly, as a side-effect of including some other .h file that includes it

c) document the contents and purpose of the header file

d) prevent the header file from being included twice, directly or indirectly

vi. A C++ reference:

a) serves as an alternative name for an object or variable (i.e., is an alias)

b) serves as a pointer to an object or variable

c) cannot be used as a parameter of a function

d) cannot be passed as an argument to a function

vii. What does "const" in the following code imply?

```
void foo (const int *x) {   …   }
```

a) the value of the pointer "x" cannot be changed inside the function foo

b) the function foo cannot have any side-effects

c) nothing; const in this case has no effect

d) the value that the pointer "x" points to cannot be changed inside the function foo

viii. What does "const" in the following code imply?

```
void Foo::bar (int *x) {   …   } const;
```

a) the method bar( ) cannot mutate any of its parameters

b) the method bar( ) cannot have any side-effects at all

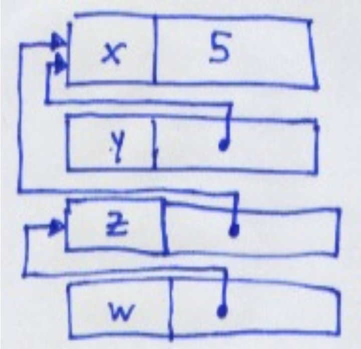
c) the method bar( ) cannot mutate any of Foo's state

d) the method bar( ) can only invoke const-y functions and methods

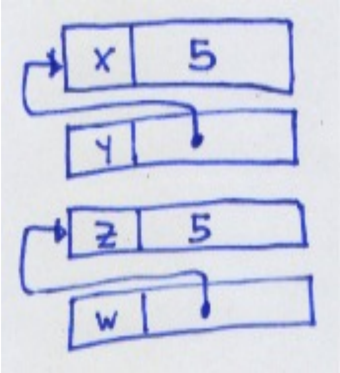
ix. Which of the following box and arrow diagrams correctly represents the following code?

```
int x = 5;
int *y = &x;
int &z = x;
int *w = &z;
```

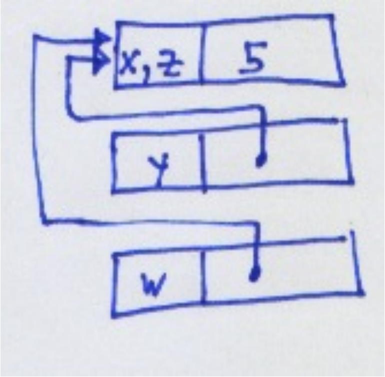
a)



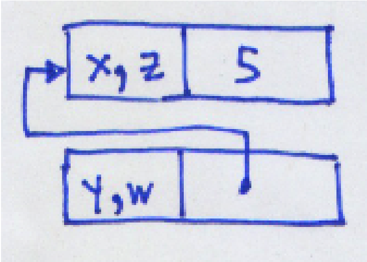
b)



c)



d)



x. The destructor of an object that is heap-allocated:

a) is invoked when the function in which it is allocated returns

b) is never invoked

c) must be invoked manually

d) is invoked when somebody uses "delete" to deallocate the object

xi. A vtable:

a) exists for each class, and contains a function pointer for each method in the class

b) exists for each class, and contains a function pointer for each virtual method in the class

c) exists for each object instance, and contains a function pointer for each method in the object's class

d) exists for each object instance, and contains a function pointer for each virtual method in the object's class

xii. Slicing occurs when:

a) the value of a derived class is assigned to an instance of a base class

b) a pointer to a derived class is cast to, and assigned to, a pointer to a base class

c) an N-element array is assigned to an M-element array, where M<N

d) an element is deleted from a std::vector

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 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.

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).

Question 7. (12 points) Sockets and packets, oh my! Circle true or false for each of the following statements.

true false TCP guarantees reliable delivery of the packets that make up a stream, assuming that the socket doesn't fail because of an I/O error.

true false IP guarantees reliable delivery of packets, assuming that the socket doesn't fail because of an I/O error.

true false Given a particular hostname (like www.amazon.com), `getaddrinfo()` will return a single IP address corresponding to that name.

true false A single server machine can handle connection requests sent to multiple IP addresses.

true false The `listen` function returns a file descriptor number for a socket that a server can use to exchange data with the connected client machine.

true false A router stores information about the complete path needed to send a IP packet to its destination.

Question 2. (16 points) Smart pointers. The following program creates a short linked list. But it leaks memory because it never deletes any of the heap-allocated data.

Fix this program so it has no memory leaks. However, you may not alter what the program does, you may not replace pointers to data with copies of the data, and you may not insert any `delete` statements. Instead, you should fix the leaks by changing the code to use smart pointers appropriately instead of regular pointers. Cross out existing code and write new code as needed. Legibility is a big help – please write clearly.

```
#include <iostream>     // needed only for printing list
#include <memory>       // smart pointers (for solution)
using namespace std;


struct Node {

  int* val_;          // ptr to node's data on heap

  Node *next_;        // next node in list or nullptr if none

};

int main() {

  // create list

  Node *list = new Node();

  list->val_ = new int(17);

  Node *p = new Node();

  p->val_ = new int(42);

  p->next_ = nullptr;

  list->next_ = p;

  // print list

  for (auto n = list; n != nullptr; n = n->next_)

    cout << *(n->val_) << "  ";

  cout << endl;

  return 0;

}
```

Question 2. (19 points) 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) (6 points) We would like to generalize this class so it can be used to store any pairs of values as long as the values have the same type (i.e., pairs of ints or 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. Hint: you'll need to start by adding template <class T> (or template <typename T>, which is equivalent) at the beginning of the class definition.

(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 that addition (+) is defined on the items stored in a `Pair` – that is handled for you by the compiler when the addition operator is used.

(b) (5 points) Write the function declaration (not the implementation) to be added to the header file `Pair.h` for the new `operator+`.

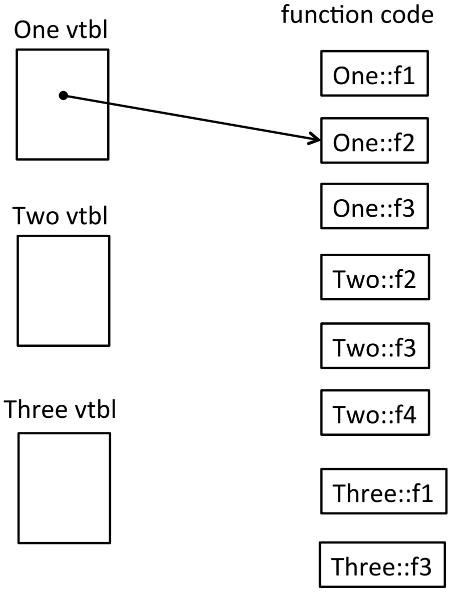
(c) (8 points) Give the code to implement this new addition operator as it would appear in a separate file `Pair.cc` containing definitions of functions not implemented in `Pair.h`. Hint: the implementation needs to begin with `template <class T>`.

Question 3. (32 points) The usual, demented, dreaded virtual function madness. Consider the following program, which, when appropriate code is inserted in the blank space in `main`, does compile and execute with no errors.

```
#include <iostream>
using namespace std;

class One {
public:
          void f1() { f3(); cout << "One::f1" << endl; }
  virtual void f2() {       cout << "One::f2" << endl; }
          void f3() {       cout << "One::f3" << endl; }
};

class Two: public One {
public:
          void f4() {       cout << "Two::f4" << endl; }
          void f2() { f1(); cout << "Two::f2" << endl; }
  virtual void f3() { f4(); cout << "Two::f3" << endl; }
};

class Three: public Two {
public:
  void f3() { f2(); cout << "Three::f3" << endl; }
  void f1() {       cout << "Three::f1" << endl; }
public:
};

int main() {

  return 0;
}
```

Remove this page from the exam, then answer questions about this code 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 the layout of the virtual function tables for the classes given on the previous page. Be sure that the order of pointers in the virtual function tables is clear (i.e., which one is first, then next, etc.). One of the function pointers is already included to help you get started.



(b) (10 points) Now, for each of the following sequences of code, assume that we try to run the program with the given lines of code replacing the empty box in `main`. Either write the output that is produced when that program is executed, or, if an error occurs, give a concise description of the problem.

(i)
```
One *x = new Two();
x->f1();
```

(ii)
```
One *x = new Two();
x->f3();
```

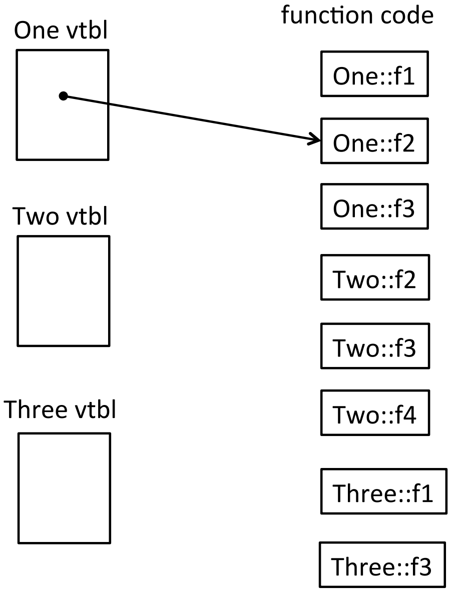
(iii)
```
Two *x = new Two();
x->f3();
```

(iv)
```
One *x = new Three();
x->f4();
```

(v)
```
Three *x = new Three();
x->f3();
```

(continued on next page)

Question 3. (cont.) (c) (6 points) Now, assume that we change the original code by writing the keyword `virtual` in front of every function declared in classes `One`, `Two`, and `Three`. Complete the vtable diagram below to show the vtable layouts for the three classes with this change. As before, be sure the order of pointers in each vtable is clear.



(d) (10 points) As before, show the output produced after changing all of the member functions in the classes to be `virtual` and then replacing the empty box in `main` with each of the following sequences of code. Either write the output that is produced when the program is executed, or, if an error occurs, give a concise description of the problem.

(i)
```
One *x = new Two();
x->f1();
```

(ii)
```
One *x = new Two();
x->f3();
```

(iii)
```
Two *x = new Two();
x->f3();
```

(iv)
```
One *x = new Three();
x->f4();
```

(v)
```
Three *x = new Three();
x->f3();
```

Question 4. (18 points) A bit of concurrency. Here is yet another small program that creates a handful of threads and lets them run concurrently. (Header files and `usings` omitted to save space, but this does compile and execute without crashing.)

```
const int NUM_THREADS = 3;
static int result = 0;         // global var. changed by threads
static pthread_mutex_t lock;   // unused variable (for now)

struct t_args {  // arguments for one thread:
  int amount;    //   amount to add to global result
  int count;     //   number of times to add amount to result
};

// Given a t_args struct, add amount to result,
// repeated count times.
void *update_result(void *arg) {
  struct t_args *p_arg = (struct t_args *) arg;
  for (int i = 0; i < p_arg->count; i++) {
    result += p_arg->amount;
  }
  return NULL;
}

int main(int argc, char** argv) {
  struct t_args args[NUM_THREADS] = {{5, 4}, {20, 6}, {-5, 2}};
  pthread_t thrds[NUM_THREADS];
  // create threads
  for (int i = 0; i < NUM_THREADS; i++) {
    if (pthread_create(&thrds[i], NULL, &update_result, &args[i])
                                                          != 0) {
      cerr << "pthread_create failed" << endl;
    }
  }
  // wait for threads to finish
  for (int i = 0; i < NUM_THREADS; i++) {
    if (pthread_join(thrds[i], NULL) != 0) {
      cerr << "pthread_join failed" << endl;
    }
  }
  // print final value of result and exit
  cout << "Total: " << result << endl;
  return 0;
}
```

(continued on the next page, but leave this page in the exam. You may need to write some changes in the above code.)

Question 4. (cont) When we run this program it starts three threads, waits for them all to finish, and then prints the final value of the variable `result`.

(a) (2 points) What value should this program print if the threads do not interfere with each other (i.e., if, for example, the three threads were executed sequentially, one after the other, rather than running concurrently):

Expected value if no interference:

(b) (6 points) When the threads run concurrently, what is the possible range of values that the program could print? (i.e., what are the possible final values for variable `result`?) Give the minimum and maximum possible values. If concurrent execution always produces the same output, just list that value as both the minimum and maximum.

(Hint: to answer this part of the question, it helps to assume that if we execute the program repeatedly, the concurrent thread scheduling will occur in as many different, unpredictable ways as possible.)

Minimum possible value:

Maximum possible value:

(c) (10 points) If it is possible for concurrent execution to produce different results, show the minimal changes needed to guarantee that the global variable `result` has the expected final value that it would have if the threads executed sequentially (i.e., the value given in your answer to part (a)). Your solution should, however, still allow for as much concurrent execution of the different threads as possible. For example, it is not reasonable to rewrite the code so the threads run sequentially, one after the other or otherwise rewrite the existing code so that it is substantially different.

Show your changes by writing them on the code on the previous page.