CSE 333 18su ~~Final~~ 2nd Exam August 17, 2018

Name _____ ID # _____

There are 6 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. _____ / 22
2. _____ / 25
3. _____ / 20
4. _____ / 20
5. _____ / 12
6. _____ / 1

Note: Please write your answers only on the specified pages. Reference pages and pages with only questions and explanations will not be scanned for grading, and you should feel free to remove them from the exam.

There is an extra blank page after the last question at the end of the exam if you need additional space for one or more answers. That page will be graded if it contains answers.

There are two pages of reference information following the blank page at the end. You may remove these pages. They will not be scanned or graded.

Question 1. (22 points) STL and C++ classes. Our friends who run the SnackOverflow concession are writing a small program to keep track of the number of items sold. A `Sales` object contains a `<string, vector<int>>` map, where each string is a unique product name. The vector associated with each product name is an ordered list of the number of copies of that item sold each time someone purchases that item. For example, if `things` is an initially empty `Sales` object and we execute the following statements,

```
things.Add("skittles", 1);
things.Add("chips", 2);
things.Add("skittles", 2);
things.Add("skittles", 1);
things.Add("bananas", 17);
things.Add("chips", 1);
```

then the `Sales` object `things` should contain the following map entries in some order:

```
<"skittles", {1, 2, 1}>
<"chips", {2, 1}>
<"bananas", {17}>
```

Here is the definition of class `Sales`. Your job is to implement various functions on the next page.

```
class Sales {
public:
  Sales() = default;
  Sales(const Sales &) = delete;
  Sales &operator=(const Sales &) = delete;

  // record a sale of num items named s
  void Add(string s, int num);

  // print product names in alphabetical order with total
  // number sold of each product using format name: total
  void PrintTotals();

private:
  map<string, vector<int>> items;

  // return sum of elements in int vector v
  int VecSum(const vector<int> &v);
};   // class Sales
```

Please write your answers on the next page and **remove this page from the exam. This page will not be scanned for grading.**

Question 1. (cont) Provide implementations of the member functions of class Sales below. You should assume that all headers are provided and you can assume that a `using namespace std;` directive has been written already. Hints: Remember there are two pages of reference information at the end of the exam that might be useful. Also, the answers can be quite short – don't be alarmed if you don't need all this space.

(a) (7 points)

```
// record a sale of num items named s
void Sales::Add(string s, int num) {




}
```

(b) (7 points) (This is a helper function to be used in part (c). It is a private member of the `Sales` class.)

```
// return sum of elements in int vector v
int Sales::VecSum(const vector<int> &v) {




}
```

(continued on next page)

Question 1. (cont.) (c) (8 points) The output of this next function should be a single line for each product in `items` consisting of product name followed by a colon and a space followed by the total number of items sold. Output should be written to `cout`. Items should be printed in alphabetical order. For the sample data at the beginning of the question, the output would be

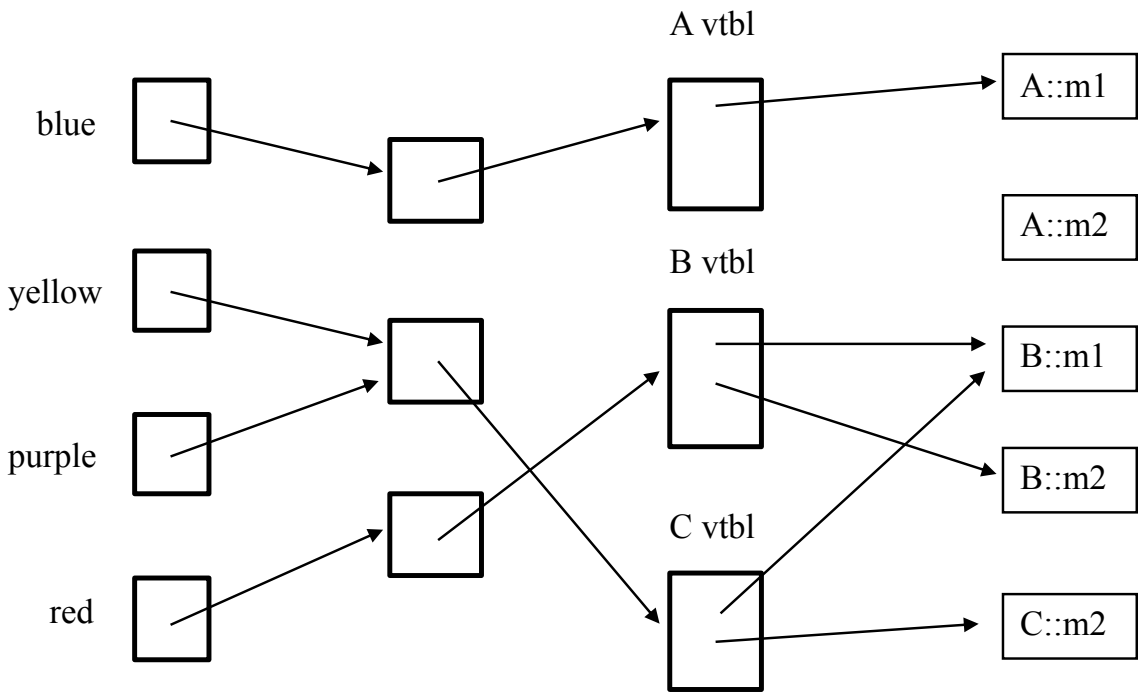
```
bananas: 17
chips: 2
skittles: 4
```

Your solution *must* use the function `VecSum` (from part (b)) to add up the contents of each vector when computing the total item sales for each product name.

```
void Sales::PrintTotals() {









}
```

Question 2. (25 points) *Not* the usual, demented, dreaded virtual function madness, but demented, dreaded, and madness in a different way. When doing a bit of software archeology, we've run across a C++ program and a diagram of the program's memory, including objects, vtables, and methods (functions). For this question we want to recreate the code that produced this execution diagram.

Here is the memory diagram:



From left to right, the first column of boxes are program variables, the next column shows C++ objects allocated on the heap (i.e., by `new`), the third column shows the vtables for the various classes, and the boxes in the last column, with labels like A::m1 represent the code for individual functions (methods).

(a) (12 points) Given the diagram above, complete the source code and on the next page. Write the necessary functions for each class and finish the initializations for each variable shown in `main`.

The body of each function should print to `cout` its class name, followed by two colons, followed by the function name. For example, class `B`'s function `m1()` should output `B::m1`. Each variable in main should point to the appropriate object on the heap. Write each function on a single line (e.g., `void f7() {cout<<"X::f7"<< endl;}`, and include `virtual` when appropriate.)

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 2. (cont.) (a) (12 points) Complete the code below so it will produce the diagram on the previous page when it is executed.

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

class A {
public:




};

class B: public A {
public:




};

class C: public B {
public:



};

int main() { // don't forget to fill in the following lines

  A *blue = _____________________________;

  B *yellow = _____________________________;

  C *purple = _____________________________;

  B *red = _____________________________;

  /// see next part of the question for additional statements that appear here ///

  return EXIT_SUCCESS;
}
```

(continued on next page)

Question 2. (cont.) (b) (7 points) Now, using the code you wrote in the previous part of the question, determine the output produced by each of the following function calls if that function call is written in the main function by itself right before the "`return EXIT_SUCCESS;`" line at the end. If there is some sort of an error and the call would not produce any output, write either "compile error" or "runtime error" as appropriate to indicate the problem.

(i) `blue->m1();`

(ii) `blue->m2();`

(iii) `yellow->m2();`

(iv) `blue->m3();`

(v) `purple->m1();`

(vi) `purple->m2();`

(vii) `red->m1();`

(c) (6 points) Now suppose we go back through the code in part (a) and remove the keyword `virtual` everywhere it appears in the code. How would your answers to part (b) change, if at all? If there are any changes, which function call(s) would produce a different result, and what would the new result(s) be? If there are no changes, say so.

Question 3. (20 points) Networking. The following code is supposed to set up a small network server, listen for a client to connect, and then call `HandleClient` (not provided) to communicate with the client over the socket. The argument to the server program is the port number to use. Take a look at the code below and on the next pages, then answer the questions about it on the following page. There are bugs!

```
// Local function declarations
void Usage(char *progname);
int  Listen(char *portnum, int *sock_family);
void HandleClient(int c_fd, struct sockaddr *addr,size_t addrlen,
                  int sock_family);

int main(int argc, char **argv) {
  // Expect the port number as a command line argument.
  if (argc != 2) {
    Usage(argv[0]);
  }

  int sock_family;
  int SOME_FD = Listen(argv[1], &sock_family);
  if (SOME_FD <= 0) {
    return EXIT_FAILURE;
  }

  while (1) {
    struct sockaddr_storage caddr;
    socklen_t caddr_len = sizeof(caddr);
    int ANOTHER_FD = accept(SOME_FD,
                    reinterpret_cast<struct sockaddr*>(&caddr),
                    &caddr_len);
    if (ANOTHER_FD < 0) {
      if ((errno == EINTR) || (errno == EAGAIN) ||
          (errno == EWOULDBLOCK))
        continue;
      break;
    }

    HandleClient(SOME_FD,
           reinterpret_cast<struct sockaddr *>(&caddr),
           caddr_len,
           sock_family);
  }

  // Close up shop.
  close(SOME_FD);
  return EXIT_SUCCESS;
}
```

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

Question 3. (cont.) More code for this problem, continued from previous page.

```
int Listen(char *portnum, int *sock_family) {
  // Populate the "hints" addrinfo structure for getaddrinfo().
  struct addrinfo hints;
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET6;       // IPv6 (also handles IPv4)
  hints.ai_socktype = SOCK_STREAM;  // stream
  hints.ai_flags = AI_PASSIVE;      // use wildcard "in6addr_any"
  hints.ai_flags |= AI_V4MAPPED;    // use v4-mapped v6 if no v6
  hints.ai_protocol = IPPROTO_TCP;  // tcp protocol
  hints.ai_canonname = nullptr;
  hints.ai_addr = nullptr;
  hints.ai_next = nullptr;

  struct addrinfo *result;
  int res = getaddrinfo(nullptr, portnum, &hints, &result);

  // Did addrinfo() fail?
  if (res != 0) {
    return -1;
  }

  // Loop through the returned address structures until we are
  // able to create a socket and bind to one.  The address
  // structures are linked in a list through the "ai_next" field
  // of result.
  int listen_fd = -1;
  struct addrinfo *rp;
  for (rp = result; rp != nullptr; rp = rp->ai_next) {
    listen_fd = socket(rp->ai_family,
            rp->ai_socktype,
            rp->ai_protocol);
    if (listen_fd == -1) {
      listen_fd = -1;
      continue;
    }

    // Configure the socket;
    int optval = 1;
    setsockopt(listen_fd, SOL_SOCKET, SO_REUSEADDR,
          &optval, sizeof(optval));
```

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

Question 3. (cont.) Last code for this problem, continued from previous page.

```
    // Success. Tell the OS that we want this to be a listening
    // socket.
    if (listen(listen_fd, SOMAXCONN) != 0) {
      close(listen_fd);
      break;
    }

    // It failed.  Close the socket, then loop back around and
    // try the next address/port returned by getaddrinfo().
    close(listen_fd);
    listen_fd = -1;
  }

  // Free the structure returned by getaddrinfo().
  freeaddrinfo(result);

  // If we failed to bind, return failure.
  if (listen_fd == -1)
    return listen_fd;

  // Try binding the socket to the address and port number
  // returned by getaddrinfo().
  if (bind(listen_fd, rp->ai_addr, rp->ai_addrlen) == 0) {
    // Return to the caller the address family.
    *sock_family = rp->ai_family;
  }

  // Return to the client the listening file descriptor.
  return listen_fd;
}
```

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

Question 3. (cont.) And now for the networking question, at last. There are some bugs in this code for you to diagnose. Definitely refer to specific parts of the code as needed in your answers, but what you write on this page should be self-contained.

(a) (10 points) When we try to run this program using port 3333 (which is not currently in use by any other process), we got this message:

```
Couldn't bind to any addresses.
```

What is causing this specific problem and how should we fix it?

(b) (10 points) After fixing the problem in part (a) we are able to run this server code and connect to it using `nc 127.0.0.1 3333`. But as soon as the connection is established, we get the following message:

```
[Error on client socket: Transport endpoint is not
connected]
Failure on accept: Bad file descriptor
```

What is causing this problem and how should we fix it?

Question 4. (20 points) Too many things at once – a question that looks a lot like a previous one, but isn't exactly. Consider the following small program that uses pthreads. This does compile and execute.

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

int g = 0;

void * worker(void * ignore) {
  int x = 0;
  for (int k = 1; k <= 3; k++) {
    x = x + 1;
    g = g + x;
  }
  printf("x = %d, g = %d\n", x, g);
  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 g = %d\n", g);
  return 0;
}
```

When we run this program it starts two threads and waits for them both to finish, and then prints the final value of the variable `g`. Each thread also prints the values of variables `x` and `g` right before it terminates.

(a) (6 points) What output would this program print if instead of running the threads concurrently, we ran thread 1 first, waited for it to terminate, then ran thread 2?

(continued on next page)

Question 4. (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 is only one or two possible outputs, write those and indicate that they are the only possible results.

(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 and separately from other output lines. If different executions lead to different outputs it is only because of the interaction between the threads as they run concurrently.)

(c) (6 points) Assuming that the threads are executed concurrently, as in part (b), what are the possible final values of global variable `g`? Circle all that could possibly happen on some possible execution:

`0` `1` `2` `3` `4` `5` `6` `7` `8` `9` `10` `11` `12` `13` `14` `15` or more

Question 5. (12 points) We know that C++ smart pointers can handle some memory management tasks for us automatically and avoid memory leaks. However, they have to be used properly to avoid problems.

(a) (6 points) Suppose we create a double-linked list with the following `Node` definition, which uses `shared_ptr`s for the links.

```
struct Node {
  int data;
  shared_ptr<Node> next;
  shared_ptr<Node> prev;
};
```

We know that this will not always prevent memory leaks. Give a brief explanation or example of why it is still possible to leak memory if we build a double-linked list out of these nodes, assuming that we use a `shared_ptr` to point to the first node in a list created from these `Node` structs.

(continued on next page)

Question 5. (cont.) (b) (6 points) One solution to the problem identified in part (a) is to use `weak_ptr`s instead of `shared_ptr`s for the backward links:

```
struct Node {
  int data;
  shared_ptr<Node> next;
  weak_ptr<Node> prev;
};
```

Are we *always* guaranteed that any program using this `Node` struct will *never* leak memory occupied by these `Node`s provided that a `shared_ptr` is used to point to the first node in a list created from these `Node` structs? Give a brief technical justification for your answer

Question 6. (1 free point – all answers get the free point) Draw a picture of something (hopefully fun) that you plan to do now that summer classes are over.

Congratulations on lots of great work this summer !!
Have a great break and say hello when you get back !
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. 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 `lists` and `sets` 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 `vectors`:
  - `c.push_back(x)` – copy x to end of c
- Some additional operations on `maps`:
  - `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[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 `sets`
  - `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)