COMPUTER SCIENCE & ENGINEERING
UNIVERSITY of WASHINGTON

CSE 333 Winter 2015

Final

You have 110 minutes to answer the questions in this quiz. In order to receive credit you must answer the question as precisely as possible.

Some questions are harder than others, and some questions earn more points than others. You may want to skim them all through first, and attack them in the order that allows you to make the most progress.

If you find a question ambiguous, be sure to write down any assumptions you make. Be neat and legible. If we can't understand your answer, we can't give you credit!

Write your name and email address on this cover sheet.

This is an open book, open notes, open laptop exam.

NO INTERNET ACCESS OR OTHER COMMUNICATION.

Name:

Email:

Question:	I	II	III	IV	V	VI	Total
Points:	13	20	10	20	30	7	100
Score:							

I. C/C++

(a) (10 points) Circle true or false for each statement (no need to justify your answers here).

True **False** STL containers always create a deep copy of inserted data.

True **False** Variables that are declared but not initialized contain zeros.

True **False** Using smart pointers such as `unique_ptr` and `shared_ptr` can achieve the same memory safety level as in Java (i.e., no buffer overflows or memory leaks).

True **False** The following is a correct way to fill a struct with zeros:

```
struct Pair { int a, b; };
struct Pair *p = ...;
memset(p, 0, sizeof(p));
```

True **False** The following is a correct and efficient way to modify a C++ string:

```
std::string s("csa333");
char *p = &s;
p[2] = 'e';
```

(b) (3 points) In "The Night Watch" the author mentions that programmers may encounter a "virtual static friend protected volatile templated function pointer" in C++. Do you think this is actually feasible? If yes, give an example. If no, explain why not.

II. (20 points) Memory nightmare

Assume the following `struct`.

```
typedef struct _labeled {
  void *data;
  uint32_t datasize;
  char **labels;
  uint32_t nlabels;
} labeled_struct, *Labeled;
```

For each function below, check whether it has: (1) a memory access error, (2) a memory leak, and (3) no errors.

If you believe there is a memory bug as in (1) or (2), circle the line number where the bug is and explain using one sentence. You only need to indicate one bug per program (if you believe there are more than one).

(a) Answer: ☐ 1 ☐ 2 ☐ 3

```
1  Labeled
2  MakeNewLabeled(void *data, uint32_t datasize, uint32_t numlabels) {
3    Labeled lbl = malloc(sizeof(labeled_struct));
4    if (lbl == NULL)
5      return NULL;
6    lbl->data = malloc(datasize);
7    if (lbl->data == NULL)
8      return NULL;
9    memcpy(lbl->data, data, datasize);
10   lbl->nlabels = numlabels;
11   if (numlabels == 0)
12     return NULL;
13   lbl->labels = malloc(sizeof(char*) * lbl->nlabels);
14   if (lbl->labels == NULL)
15     return NULL;
16   for (uint32_t i = 0; i < numlabels; i++) {
17     lbl->labels[i] = NULL;
18   }
19   return lbl;
20 }
```

(b) Answer: ☐ 1 ☐ 2 ☐ 3

```
bool StoreLabel(Labeled lbl, char *label, uint32_t indx) {
  if (lbl == NULL || indx >= lbl->nlabels) return false;
  if (lbl->labels[indx] != NULL) free(lbl->labels[indx]);
  lbl->labels[indx] = malloc(strlen(label));
  strcpy(lbl->labels[indx], label);
  return true;
}
```

(c) Answer: ☐ 1 ☐ 2 ☐ 3

```
bool GetData(Labeled lbl, void **data_out, uint32_t *datasize_out) {
  if (lbl == NULL) return false;
  void *tmp = malloc(lbl->datasize);
  if (tmp == NULL) return NULL;
  memcpy(tmp, lbl->data, lbl->datasize);
  *data_out = tmp;
  *datasize_out = lbl->datasize;
  free(tmp);
  return true;
}
```

(d) Answer: ☐ 1 ☐ 2 ☐ 3

```
void FreeLabeled(Labeled lbl) {
  if (lbl == NULL) return;
  for (uint32_t i = 0; i < lbl->nlabels; i++) free(lbl->labels[i]);
  free(lbl);
}
```

III. (10 points) Sockets

Circle true or false for each statement (no need to justify your answers here).

True **False** `getaddrinfo()` will return a single IP address corresponding to a given hostname (e.g., `uw.edu`).

True **False** Both TCP and UDP guarantee reliable delivery of the packets that make up a stream.

True **False** One can use `read(sockfd, ...)` to read data from a socket.

True **False** One can use `fsync(sockfd)` to force the data out of a socket and ensure the delivery of the data to another machine.

True **False** When a browser fetches webpages from a server (as in Homework #4) using the HTTP protocol, if a single IP packet is lost, the browser will have to resend the HTTP request and try to fetch the data again.

IV. Threads

Ben Bitdiddle is writing a C++ program, using 1000 threads to increase the value of the global variable x to 1000.

```
1  #include <stdio.h>
2  #include <mutex>
3  #include <thread>
4  #include <vector>
5
6  int x = 0;
7
8  void worker() {
9    int y = x;
10   x = y + 1;
11   printf("x = %d, y = %d\n", x, y);
12 }
13
14 int main() {
15   std::vector<std::thread> ws;
16   for (size_t i = 0; i < 1000; ++i)
17     ws.push_back(std::thread(worker));
18   for (auto &t : ws)
19     t.join();
20   printf("final x = %d\n", x);
21   return 0;
22 }
```

(a) (5 points) Describe one data-race bug in the program (there may be more than one). Be specific about which line(s) the bug is in the source code and what threads are involved.

(b) (5 points) Briefly explain why the bug you described in the previous question is a data-race bug, using either happens-before or lockset as described in the Eraser paper.

(c) (10 points) Ben Bitdiddle decides to eliminate data races using locks. In particular, he adds a global mutex `m`, and wraps every line of code in `worker()` with `lock()/unlock()`, as follows:

```
...
std::mutex m;

void worker() {
  m.lock();
  int y = x;
  m.unlock();

  m.lock();
  x = y + 1;
  m.unlock();

  m.lock();
  printf("x = %d, y = %d\n", x, y);
  m.unlock();
}
...
```

With Ben's fix, will the program correctly print "final x = 1000" in every run? Explain why or why not.

V. Terra Nova

Consider the following C function, `copyfd()`, which copies the content of a file (referred by the file descriptor `infd`) to another (referred by the file descriptor `outfd`).

Note that `err()` prints an error message and terminates the current program.

```c
#include <err.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>

void copyfd(int infd, int outfd) {
  char buffer[512];
  for (;;) {
    ssize_t ret = read(infd, buffer, sizeof(buffer));
    if (ret == 0) break;         /* done */
    if (ret < 0) {
      if (errno == EINTR) continue;
      err(1, "read");            /* exit with an error */
    }

    char *p = buffer;
    do {
      ssize_t written = write(outfd, p, ret);
      if (written <= 0) {
        if (errno == EINTR) continue;
        err(1, "write");        /* exit with an error */
      }
      p += written;
      ret -= written;
    } while (ret);
  }
}
```

(a) (10 points) Describe how data is transferred from `infd` to `outfd`. Be specific about what system calls are involved and how many times the data is copied from the kernel to user space (e.g., `buffer`) and copied from user space to the kernel.

(b) (20 points) Alyssa P. Hacker wants to improve the performance of the program. In particular, she wants to reduce the number of data transferring between user space and the kernel. She learns that it's possible to create a kernel-space "buffer," using a system call called `pipe`. Below is part of the manpage.

```
int pipe(int pipefd[2]);
```

"`pipe()` creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array `pipefd` is used to return two file descriptors referring to the ends of the pipe. `pipefd[0]` refers to the read end of the pipe. `pipefd[1]` refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe."

In other words, if one puts data into `pipefd[1]`, one can get the content from `pipefd[0]`. Ignore the return value from `pipe()`.

To transfer data from a file to a pipe, and from a pipe to a file, one can use another Linux system call called `splice`. Below is part of the manpage.

```
ssize_t splice(int fd_in, loff_t *off_in, int fd_out,
               loff_t *off_out, size_t len, unsigned int flags);
```

"`splice()` moves data between two file descriptors without copying between kernel address space and user address space. It transfers up to `len` bytes of data from the file descriptor `fd_in` to the file descriptor `fd_out`, where one of the descriptors must refer to a pipe.

The following semantics apply for `fd_in` and `off_in`:

- If `fd_in` refers to a pipe, then `off_in` must be `NULL`.
- If `fd_in` does not refer to a pipe and `off_in` is `NULL`, then bytes are read from `fd_in` starting from the current file offset, and the current file offset is adjusted appropriately.
- If `fd_in` does not refer to a pipe and `off_in` is not `NULL`, then `off_in` must point to a buffer which specifies the starting offset from which bytes will be read from `fd_in`; in this case, the current file offset of `fd_in` is not changed.

Analogous statements apply for `fd_out` and `off_out`.

Upon successful completion, `splice()` returns the number of bytes spliced to or from the pipe. A return value of 0 means that there was no data to transfer, and it would not make sense to block, because there are no writers connected to the write end of the pipe referred to by `fd_in`.

On error, `splice()` returns -1 and `errno` is set to indicate the error. "

For this question, always use 0 for `flags`.

Now, read the above manpages again, and help Alyssa improve the program using `pipe` and `splice`. Write down a new `copyfd()` function.

Hint: you only need to change a few lines of the code—create two file descriptors of a pipe, move data from the input file `infd` to the write end of the pipe, and move data into the output file `outfd` from the read end of the pipe.

VI. CSE 333

Any answer, except no answer, will receive full credit.

(a) (3 points) The paper "The Internet Considered Harmful" describes Melissa, a groundbreaking algorithm that combines lambda calculus and journaling file systems to achieve Byzantine fault tolerance for the DNS. Will Melissa be breakable with a quantum computer, like Shor's algorithm for integer factorization? Explain why or why not.

(b) (4 points) Did you learn anything in CSE 333? Please give a score on a scale from 0 (nothing) to 10 (more than I had hoped for), and briefly explain.

End of Quiz