

# CSE 333 Section 3 Solutions - POSIX I/O Functions

Welcome back to section! We're glad that you're here :)

## Exercise 1:

A common use of the POSIX I/O function is to **write** to a file; fill in the code skeleton below that writes all of the contents of a string `buf` to the file `333.txt`.

```
// **NOTE: This is one way to solve this exercise.
// There exist other correct solutions to this exercise.

int fd = open("333.txt", O_WRONLY); // open 333.txt
int n = ....;
char *buf = ..... ; // Assume buf initialized with size n
int result;

char *ptr = buf; // initialize variable for loop

... // code that populates buf happens here

while (ptr < buf + n) {
    result = write(fd, ptr, buf + n - ptr);

    if (result == -1) {
        if (errno != EINTR && errno != EAGAIN) {
            // a real error happened, return an error result
            close(fd); // cleanup
            perror("Write failed");
            return -1;
        }
        continue; // EINTR or EAGAIN happened, so loop and try again
    }
    ptr += result; // update loop variable
}
close(fd); // cleanup
```

### Follow-up Questions:

Why is it important to store the return value from the `write()` function? Why do we not check for a return value of 0 like we do for `read()`?

**write() may not actually write all the bytes specified in count.**

**Writing adds length to your file, so you don't need to check for end of file.**

Why is it important to remember to call the `close()` function once you have finished working on a file?

**In order to free resources i.e. other processes can acquire locks on those files.**

## Exercise 2:

Given the name of a directory, write a C program that is analogous to `ls`, i.e. prints the names of the entries of the directory to `stdout`. Be sure to handle any errors!

Example usage: `./dirdump <path>` where `<path>` can be absolute or relative.

```
int main(int argc, char** argv) {
    /* 1. Check to make sure we have valid command line arguments */
    if (argc != 2) {
        fprintf(stderr, "Usage: ./dirdump <path>\n");
        return EXIT_FAILURE;
    }

    /* 2. Open the directory, look at opendir() */
    DIR* dirp = opendir(argv[1]);
    if (dirp == NULL) {
        fprintf(stderr, "Could not open directory\n");
        return EXIT_FAILURE;
    }

    /* 3. Read through/parse the directory and print out file names
       Look at readdir() and struct dirent */
    struct dirent* entry;

    entry = readdir(dirp);
    while (entry != NULL) {
        printf("%s\n", entry->d_name);
        entry = readdir(dirp);
    }

    /* 4. Clean up */
    closedir(dirp);
    return EXIT_SUCCESS;
}
```

## Bonus Exercise

Given the name of a file as a command-line argument, write a C program that is analogous to `cat`, *i.e.* one that prints the contents of the file to `stdout`. Handle any errors!

Example usage: `./filedump <path>` where `<path>` can be absolute or relative.

```
int main(int argc, char** argv) {
    /* 1. Check to make sure we have a valid command line arguments */
    if (argc != 2) {
        fprintf(stderr, "Usage: ./filedump <filename>\n");
        return EXIT_FAILURE;
    }
    /* 2. Open the file, use O_RDONLY flag */
    int fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        fprintf(stderr, "Could not open file for reading\n");
        return EXIT_FAILURE;
    }
    /* 3. Read from the file and write it to standard out.*/
    char buf[SIZE];
    ssize_t len;
    do {
        len = read(fd, buf, SIZE);
        if (len == -1) {
            if (errno != EINTR && errno != EAGAIN) {
                close(fd);
                perror(NULL);
                return EXIT_FAILURE;
            }
            continue;
        }
        size_t total = 0;
        ssize_t wlen;
        while (total < len) {
            wlen = write(1, buf + total, len - total);
            if (wlen == -1) {
                if (errno != EINTR && errno != EAGAIN) {
                    close(fd);
                    perror(NULL);
                    return EXIT_FAILURE;
                }
                continue;
            }
            total += wlen;
        }
    } while (len != 0);
    /*4. Clean up */
    close(fd);
    return EXIT_SUCCESS;
}
```