ECE 26400 Spring 2025 Practica 1

Section 2 - March 11th, 9:00-9:50am

1 Overview

For the programming practica, we will be working with binary files. Specifically, we will be working with a binary format that is inspired by how zip files work.

Your task is to modify `practica.c` to fill in the required functions. You will have 45 minutes to complete this task and make your final submission to Gradescope. During that time, you are free to make as many submissions as you wish to run tests on Gradescope.

1.1 Grade

You will get credit for submitting the correct file (`practica.c`), your code compiling, your code having no memory leaks or memory errors, and your code passing our tests. Any modifications made to files other than `practica.c` will be ignored; it is only necessary to submit `practica.c`.

Note that we reserve the right to change specific test inputs after the practica. While you are not expected to write your own tests, hardcoding to the specific provided test cases may cause your grade to drop after the practica. You should make an effort to solve the general problem.

1.2 Editor

You will edit `practica.c` on your local machine. You may use any text editor that does not contain AI features (such as Copilot). You should not use a remote server for editing (such as eceprog/thinlinc).

1.3 Compiler

You do not need a compiler to complete the practica; simply uploading your code to Gradescope will show both compiler errors and the results of the tests in `main.c`. However, should you already have a compiler installed at the time of the practica, you are free to use the provided Makefile to compile and run code on your local machine. You are responsible for your compiler working if you choose to compile with your local machine.

1.4 Resources

The practica is open book, open notes. You are free to use any physical resources, such as textbooks or paper notes. You are also free to use digital notes, provided they may be accessed offline.

The only online resources permitted are Brightspace (to download the assignment), Gradescope (to submit and test your code), and `http://man.he.net/`, which is an online version of the C man pages.

1.5 Tips

Prioritize writing code to pass simpler test cases first, and verify that they pass before moving onto complex test cases. If you run out of time, it is better to have a complete solution for some tests than an incomplete solution for all tests.

2 Problem Description - Read Index

Your task is to implement `read_index` in `practica.c`:

```
Index read_index(const char* filename);
```

This function reads data from a binary file into an index. The index in C is defined by the following struct in `practica.h`:

```
typedef struct {
    Entry* entries;
    int size;
} Index;
```

The index consists of an array of entries, with size determined by `size`. Entries are defined by the following struct:

```
typedef struct {
    char key[100];
    int value;
} Entry;
```

`key` is a string stored directly in the entry as an array; since it is null terminated. `value` is a 4 byte integer which the index associates with the key (its exact usage is beyond the scope of this function; you may assume any signed 4 byte integer is a possible input).

The purpose of `read_index` is to write all data from the index into the file located at `filename`. If the file cannot be opened, the function returns an index with `size == 0` and `entries == NULL`. If the file can be opened, the entries are read from the file and returned in the index.

The format of the binary file you are to read is as follows:

- 4 byte little endian integer determining the number of entries.
- For each entry:
  - 4 byte little endian integer representing the entry `value`.
  - 4 byte little endian integer representing the size of the entry key (excluding the null character).
  - A number of characters up to the key size, representing the key contents (excluding the null character).

Since the format of the entry in binary contains variable length strings, it is not possible to read the entries using a single `fread` call. Rather, you will need multiple `fread` to read each field individually.

Additionally, since `entries` is a pointer, it will be necessary to allocate it using information from the binary file to determine the size. The entry key is not a pointer, so does not need to be allocated directly.