Design Choices

Structure of the Zip File: Our zip file stores the file content first, then the metadata for the files, and then a footer containing the total number of files stored and the total size of all the files (not directories) being stored.

Our project has a helper functions header along with the main program. Command line input validation is done in the main file.

We are using 3 structs

- Footer struct-

```
struct footer
{
   int num_headers;      // total number of files/headers we have
   int total_file_size; // sum of all the file sizes
};
```

    - Used to store info that helps navigating the zip file - added at the very end (after metadata array) of the file so it is easy to find

- Metadata struct -

```
struct header
{
   char *file_name;
   int file_size;
   mode_t file_mode;
   uid_t file_uid;
   gid_t file_gid;
   time_t file_mtime;
   int fileOrDirectory; // flag --> 1, file. folder --> 0
};
```

    - Stores the metadata for all the files- array of this struct is added in one go after the all the file contents are added to the zip file

- AppendResult struct-

```
struct AppendResult
{
   struct footer data;
   struct header *head;
};
```

- Just wraps together the updated footer and metadata together after appending the file so it is easier to return this struct when we want a function to have both header and footer as the return value (C does not allow return multiple values from a function)

-c : we create the new zip file, write all the file content to it after recursively going through the directories and add the metadata to the array of metadata structs. At the end we write this array in the file and then the footer.

-a : gets all the metadata and footer info from the current file, navigates to the position where the file content ended(using total_file_size in footer) and appends the new content there and remove the metadata and footer from the end of the file, updates the metadata and the footer info and adds both of them at the end of the file (that is file with the new content added)

-x : uses the metadata of the files to get their paths and then go through the path creating new directories if necessary and then using “chdir” to go in them and creating/going to new files/directories

-m : loops over the metadata array of files and displays it

-p : prints the hierarchy info. Recursively traverses the directories if they are found and prints the file names within and keeps doing the same Basically it prints the hierarchy data in the same order it was stored rather than traversing each level of the file system.

- We were confused about this and implemented two functions. One prints the hierarchy data in the same order it was stored, and the second one uses the other approach. However, we think the first approach is correct and hence, the second function has been commented out.