Technical Design Summary
SFArchiver

Team - Seg Fault
Members - Simon Fong, Tony Wu, Shashankar Sudarsan

Initially, we implemented feature 1 – command line processing. For this we mapped all the possible commands to integers and used switch command to invoke different functions. The default option threw “Invalid argument” to the terminal. Each of the switch cases called corresponding methods from the StorageEngine class.

The main design idea we had to tackle the second part of this project was to separate the archive file into two separate portions: a header and data portion. The header portion includes all of the metadata – file name, file size, date added, and byte offset within the archive file – for all of the files within the archive. The data portion holds all of the binary data for all the files. In order to manage these two partitions easily, we used two separate files. This way we could modify each portion without affecting the other; it also helped a lot in the debugging process as we could check what was exactly in each file at any point. The biggest challenge that we faced using this design was reliably reading and modifying the header portion. We came up with a solution by using an internal list to keep track of the files inside of our archive as the program is running. To do this, we utilized a map data structure to map a filename to an object which holds the file size, date added, and byte offset. This allows us to easily store and modify the information needed for the header file. The implementations for each method is shown below:

1. **Add file to Archive**
    a. We do not let the user add two files with the same name. If they try, we report an error to the user.
    b. We extract the file properties from the files such as filename, size in bytes, file type, and date added and store this in the header section of our archive file.
    c. When we add the file content, we also track the offset of the data and store that in the archive header.
2. **Delete file from Archive**
    a. In order to delete a file from the archive, we first, read the archive. Then, we store the header information into a custom File object in memory and store the content of all the files into a temporary file. To delete, we simply remove the header data

for the specified file, adjust all other header data for other files, and don't copy the data for the specified file into the final archive file.

3. **List files in Archive**
    a. For listing all files, we read in the header data into an object. Then, we iterate through each file and print out its properties.
    b. For listing a specified file, we iterate through the files and only print out the file properties if the filename matches. If no file matches, we print our message saying that file does not exist in our archive.
4. **Find file containing "string" within Archive**
    a. To find the file that contains the given string, we iterate through each file and iterate through each byte in the file until we find a sequence that matches our given string.
5. **Extract a file from Archive**
    a. In order to extract a file, we search through our header object and find the file that matches the given filename. If we find it, copy all its data into a new file with that filename.
6. **Show program version**
    a. We print out the string we have stored.