ECE 180 - SOFTWARE FOUNDATIONS - I
GROUP PROJECT - SLEEPLESS IN SAN DIEGO

TECHNICAL DESIGN SUMMARY OF FINAL PROJECT

INTRODUCTION

This program is to implement a storage engine and manage a collection of resources in a single binary file that is considered to be an archive. This archive is similar to how a compressed file such as a .zip or .tar file works. The entire program should run as a standalone command line tool. The storage engine must be able to manage archives which have a collection of files. The functionality expected here is to able to add/delete files to or from a given archive, list the files present in an archive or list the information about a given file such as date added and size of the file, extract a given file from an archive and copy it to a local directory, and return the version number of the entire program.

Memory management is also an important consideration here since when we delete files, the resulting free memory blocks must be properly accounted for by shifting other file blocks or any other suitable method.

APPROACH

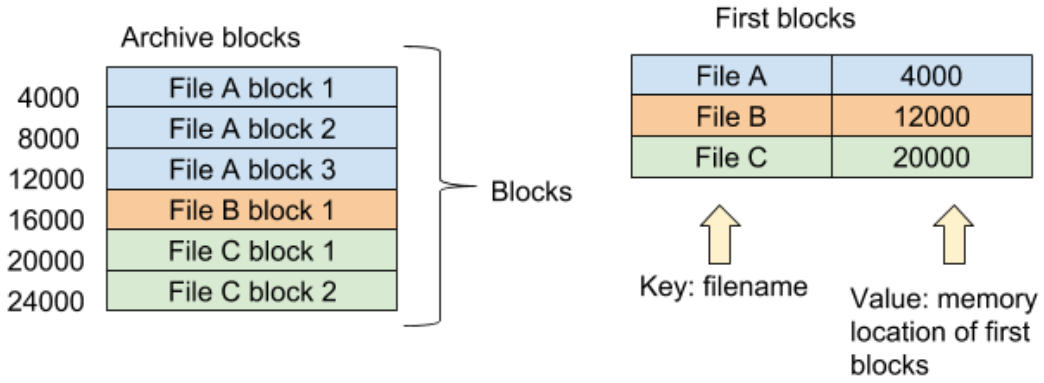
Our approach involves using uniformly sized blocks of data, such that the files added to the archive are added as separate blocks. We have the size of one block as 4000, where the header is of size 500 and the data is of size 3500. Initially, when we add a file to the archive, we calculate the size of the file, and based on that we also calculate the number of blocks required in order to store the file. This approach is easier because it enables us to deal with individual blocks of data, instead of referencing memory which might make the archive operations more complex and prone to failure.

All the blocks in each file are implemented as a linked list; the traversal operation of linked list ensure that the program has access to all the contents in the file. Therefore, the program only needs to keep track of the first block in each file. In memory, the blocks from the same file are stored contiguously and the file are ordered by the time added. In the delete method, the program will write the old contents into a temporary file in disk, then read back from the temporary file and write to old file but skipping the contents of the file to be deleted. This enables efficient memory management without having any empty blocks in the archive.

In the list method, we store the information about each file, the date added and the size of the files, and display them when it is called. Also, all the file names in an archive are stored and displayed when it is required as well.

In order to extract the files, we locate the first block of the file to be extracted. An output file is opened using fstream, and the first block of the file is written to a buffer. From the buffer, it is written to the output file. After each block is written, it moves on to the next block using the next pointer if the same file spans multiple blocks, and writes each of them to the output file till all the file blocks are written. Thus, a file is extracted to the current directory from the archive by this method.

For the find method, we first examine whether the file in question within which the given string is to be found is first a file containing text. If so, the file is searched line by line to see if the string to be found is located within the file. If found, the properties of the file in which the string is found is printed out.



BLOCK DIAGRAM OF FILE STORAGE ENGINE

CHALLENGES

Every time the program closes, the data structures that were used to keep track of the archive and block contents in memory disappear. Therefore, we need to reconstruct the data structure by reading the files that are saved to the disk when the program is ran the next time.

It is also imperative to keep track of which files are stored in which blocks in order to facilitate easy deletion and extraction of the files. Thus, an efficient mechanism to keep track is necessary to ensure this.