# SFArchiver: A File Archiver in C++

ECE 180 WI18 Final Project - March 11th, 2018

Team 0: Ankitesh K. Singh, Girish Bathala, Jing Liang, Srinath Narayanan

---

## Problem Definition:

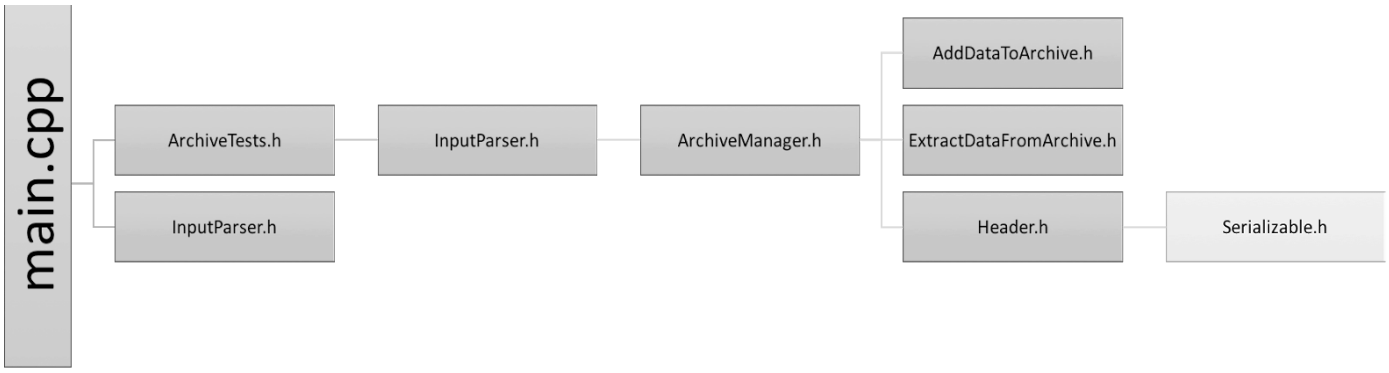
We are asked to design a storage engine which supports the command-line interface. Inside the storage engine, we can create several archives to store different kind of files. We can apply the following operations for each archive in our storage engine: add file to archive, delete file from archive, list file in archive, find file containing "string" within archive, extract a file.

- For **adding** file to archive, the valid command should be: "sfarchiver add archivename filename (or) filePath". If the archivename doesn't exist in the storage engine, it is automatically created otherwise emit "File already exists" to the terminal.
- To check **version**, use command: "sfarchiver version" (or) "sfarchiver -v".
- For **deleting** file from archive, the valid command should be: "sfarchiver del archivename filename". If the archivename doesn't exist or the filename doesn't exist, we throw an error.
- For **listing** files in archive, "sfarchiver list(-l) archivename" displays the information of all the files in that archive in the terminal. "sfarchiver list(-l) archivename filename" displays only the information pertaining to the given "filename".
- For **extracting** a file from archive, the valid command should be: "sfarchiver extract archivename filename", and a copy of the named files will be emitted to the terminal.
- For **finding** a file from a given archive which matches a given "string": "sfarchiver find archivename filename", and the details of file containing the string is pushed to screen.

We implemented two additional commands:

- With **test** command, the executable runs several tests which tests all the above functionality and returns info about failure. Command : "sfarchiver test"

Figure 1: File Hierarchy




- With **-c** or **clean** command we can force clean the left over space upon delete(when below the threshold criteria). Command : "sfarchiver clean (or) -c ArchiveName

## Design Description

We used the following classes in this program :

**InputParser** class deals with command-line interface. The main.cpp will only call InputParser class when the program runs. If the command is valid, InputParser class will call the corresponding member function in ArchiveManager class, otherwise the system will emit a clear and appropriate error message to the terminal.

**Header** class derives from **Serializable** abstract base class. In Serializable class, we defined two pure virtual functions: serialize() and deserialize() and overrode these two functions in Header class. Basically, Header class is a class which can get newly added file's information (name, type, added date , file exists boolean and file size) and store it to the corresponding header file. The file's information(name, type, added date, file exists boolean and filesize) will write to some header file when serialize() is

called , whereas deserialize() can read from some header file to get the information of all the files stored in that header file.

**ArchiveManager** class deals with all the functions of an archive. It has five member functions: addToArchive(), extractFromArchive(), delFromArchive(), listArchive(), find() and one utility function findinArchive(). **AddDataToArchive** class and **ExtractDataFromArchive**

class will be called in addToArchive() and extractFromArchive() to do the exact adding and deleting job.

**Delete/ Space Management mechanism : To improve storage efficiency through a thresholding approach**

A key goal of this project is storage efficiency and our space management mechanism kicks in when the total space occupied by the deleted files in the archive is greater than or equal to 30%. This calculation is performed swiftly by our algorithm as it iterates only through the header information of files and the "exits" flag in the header metadata. "Exists" if set to false conveys that the file is deleted from the archive. When the threshold limit is breached, all files in the archive "archName" with file "exists" == true, in the header metadata, are copied into a "temp" archive. The old archive "archName" with required and deleted files is now cleared from the disk and "temp" is renamed to "archName".

**Extract: How chunking helps?**

The data pertaining to each file in the archive is written/read into/from "chunks" aka char buffers of fixed size (512 Bytes). This mechanism provides an easy interface to read data, write data and move around chunks while calling the delete method. Also, reading a large file into memory will cause the program to crash during runtime and chunking is a more practical approach to execute the same.

**Find: How we are merging chunks and searching for faster access**

The find method returns details of all files which contain the given input string. Each file's header is deserialized and checks are performed to verify if it is a valid text file ( type == "txt") and if the file is not already deleted(exists == true). The file data is deserialized using the "chunking" mechanism described above and there might be a possibility where a word in the file is cut off at the intersection of chunks. std::string.find() on a single buffer will not be able to match the cut-off word. Therefore, chunks are concatenated two at a time and the std::string.find() is implemented on the combined string to find the required match.