New York University Abu Dhabi

CS-UH 3010 - Spring 2023

Programming Assignment 3

Due: **May 12^{th}, 2023**

Preamble:

In this assignment, you will write an archiving application termed adzip –it stands of abu dhabi zip– whose objectives are to *1)* to "flatten" a logical hierarchy(-ies) of a filesystem and *2)* store this hierarchy(-ies) in a single file having a postfix `.ad`. Your program should provide the capability to "re-create" the flattened logical hierarchy(-ies) possibly elsewhere in the filesystem; it should be also able to selectively retrieve files/directories stored in a `.ad` file.

adzip may accept a number of input parameters including files and/or directories that all have to be ultimately incorporated into a designated `.ad` archive. The way files and directories are stored in the archive should be such that elements could be **readily recreated** along with all their basic file-system properties (characteristics).

In general, adzip should demonstrate similar usage and behavior with programs you have used so far including `zip`, `rar` and `7z`.

Procedural Matters:

◇ Your program is to be written in `C/C++` and *must run* on `NYUAD's Ubuntu` server `bled.abudhabi.nyu.edu`.

◇ You have to first submit your project via `brightspace.nyu.edu` and subsequently, *demonstrate* your work.

◇ Dena Ahmed (`daa4-AT+nyu.edu`) will be responsible for answering questions as well as reviewing and marking the assignment in coordination and collaboration with the instructor.

Invocation of your Archiving Application:

Your program named `adzip` has to make use of at least the following flags (and respective inline arguments):

```
prompt >>  adzip {-c|-a|-x|-m|-p} <archive-file> <file/directory list>
```

- `-c` store in the archive file `<archive-file>` (appropriately designated with postfix `.ad`), all files and/or directories provided by the list `<file/directory-list>`. If other files/directories exist within in `<directory-list>`, then all this content is recursively stored in the `<archive-file>`.
- `-a` append filesystem entities (files and directories) indicated in the `<file/directory list>` in the archived file `<archive-file>` that already exists. If additional files and directories exist in `<file/directory list>`, they are also recursively appended along with their content in the designated `<archive-file>`.
- `-x` extract all files and catalogs that have been archived in file `<archive-file>` by accurately recreating the original hierarchy in the *current working directory*.
- `-m` print out the meta-data `(owner, group, rights)` for all files/directories that have been archived in `<archive-file>`.
- `-p` display the hierarchy(-ies) of the files and directories stored in the `<archive-file>`. Do this in a way that can be readily understood by the user.

A few points to consider:

- In contrast to the usual behavior of programs such as `rar/zip`, `adtar` is not required to use any additional inline arguments. More specifically, your arguments should not involve the *Standard Input/Output/Error*.
- The retrieval involves only the files/directories that are provided in the respective list. If no such list is provided, then all files/directories must be retrieved and respective hierarchies must be established on the current working directory.
- To conserve space in the `.ad` archive file, you may elect to store the content of a data-file in a compressed form. When the `adzip` uses the flag `-x` all files are retrieved and established in the hierarchy(-ies) in uncompressed form. Compression of data-files is an option you may consider while your create `adzip`. Although it is a desired feature for obvious reasons, it is not a must for the assignment.
- Your program should also try to deal with *hard links* in any reasonable/feasible manner.

Designing your Program:
The archiving of the files and directories must be done in a way that their retrieval can be accomplished easily. Besides the content of the filesystem entities the archive should maintain respective properties including the owner, the group as well as the access rights of each entity.

In general, you may adopt your own way of organizing the `adzip` archive files. Figure 1 depicts such a *possible* organization. You can use this organization, you may extend and/or modify as you see it fit or even finally, you may adopt a vastly different design for your archives. You will have to justify your design choices by offering arguments for the design you will finally follow.

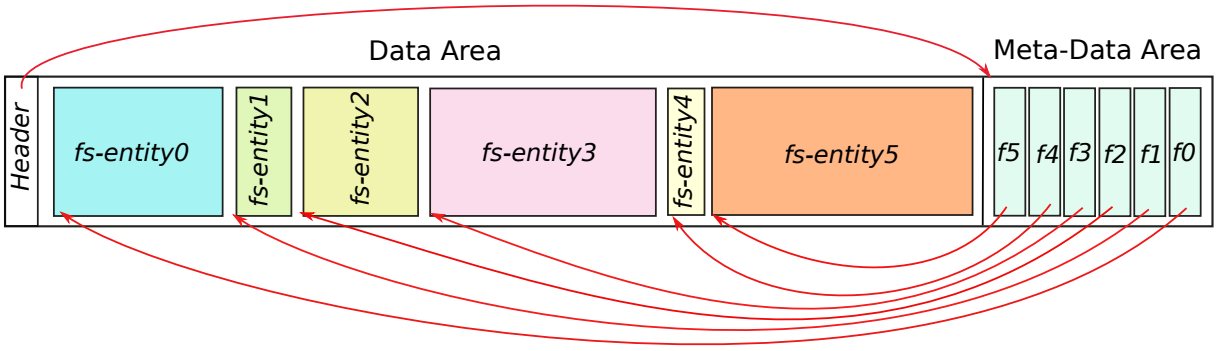

Figure 1: A *Possible* Organization of a `.ad` Archive containing 6 filesystem entities (*fs-entities*).

The organization of Figure 1 initially features all stored filesystem entities including files and directories. File system entities are archived in sequence. At the end of the stored entities, there is a "dictionary" (meta-data area or structure) containing a set of records; each record describes a file or a directory in the archive and may contain the file/directory characteristics including the size of the size, the position a file starts in the `.ad` archive, etc. There is also a fixed-length header in the archive that may contain any information you may want along with the position that the "dictionary" of the archive starts from.

The rationale for the above design is that it can more effectively facilitate the appending of new filesystem elements into an existing `.ad` archive file. Note that the size of the meta-data area is often expected to be much less voluminous from the first data-area of the archive where the "content" of filesystem entities is stored. Hence, the meta-data area can be easily "re-positioned" and re-written further "down" the `.ad` file, should be want to append more entities at the end of

the data-area.

We should note that archived directories may contain elements that will have to be organized on their own. If for instance, a directory features another directory, the latter has to be organized appropriately. This organization should be carried out in a recursive fashion. When a sub-directory is to be retrieved from the archive, then the entire `adzip` structure has to be properly "walked over" so as we can accurately locate where the sub-directory is.

What you Need to Submit:

1. In this assignment, you can select 1 partner to work as a group. Alternatively, you can also work alone if you like.
2. A directory that contains all your work including source, header, Makefile, a readme file, etc.
3. A short write-up about the design choices you have taken in order to design your program(s); a pdf document that is 2-3 pages long would be sufficient.
4. All the above should be submitted in the form of "flat" `zip` or `7z` file bearing your name: For instance: `AlexDelis-Proj4.7z`.

   If you work with a partner, then both your names have to be used in the submission `zip` file; for example: `DenaAhmedAlexDelis-Proj4.zip`. In this case, only 1 parner has to submit the zip-ball.
5. Submit the above 7z/zip-ball to `brightspace.nyu.edu`

Grading Scheme:

Aspect of Programming Assignment Marked	Percentage of Grade (0–100 + 10)
Quality in Code Organization & Modularity	15%
Basic File/Directory Archiving Requirements	35%
Append Operation	10%
Displaying Meta-data	10%
Printing Archiving Hierarchy	05%
Archiving directories/files in multiple level	10%
Use of Makefile & Separate Compilation	05%
Well Commented Code	05%
Design Clarifications Provided in Write-Up	05%
Use of Compression with Files	05% (bonus)
Archiving Links	05% (bonus)

Miscellaneous & Noteworthy Points:

1. You have to use *separate compilation* in the development of your program.
2. If you decide to use `C++` instead of plain `C`, you *should not* use STL/templates.
3. We would like to see this project be done in **groups of two students**. If you very strongly wish, you could also develop this assignment by working individually. Regardless, your code should run on the LINUX server: `bled.abudhabi.nyu.edu`

4. Pieces of code that are developed by groups should display the full names of both group members in all source files as part of a commented-out-line.

5. You can access the above server through `ssh` using port `4410`; for example at prompt, you can issue: "`ssh yourNetID@bled.abudhabi.nyu.edu -p 4410`"

6. Although it is understood that you may exchange ideas on how to make things work and seek advice from fellow students/groups, sharing of code is *not allowed*.

7. If you use code that is not your own, you will have to provide appropriate citation (i.e., explicitly state where you found the code). Otherwise, plagiarism questions may ensue. Regardless, you have to fully understand what such pieces of code do and how they work.

8. The project should run on the LINUX server: `bled.abudhabi.nyu.edu`

9. You can access the above server through `ssh` using port `4410`; for example at prompt, you can issue: "`ssh yourNetID@bled.abudhabi.nyu.edu -p 4410`"

10. There is ongoing Unix support through the *Unix Lab* Saadiyat [Uni23].

References

[Uni23] NYUAD UnixLab. `https://unixlabnyuad.github.io/`. C2 Building, 2023.