

# Contents

<b>1 Overall System Design</b>	<b>1</b>
1.1 Overall Layout / theory . . . . .	1
1.2 Command Line Parsing . . . . .	1
1.2.1 Problems . . . . .	1
1.3 Listing files . . . . .	2
1.4 Extracting Files . . . . .	2
1.4.1 Caveats . . . . .	2
1.5 Adding Files . . . . .	2
<b>2 Work Log</b>	<b>3</b>
<b>3 Challenges</b>	<b>3</b>
<b>4 Questions And Answers</b>	<b>3</b>
4.1 what do you think the main point of this assignment is? . . . . .	3
4.2 how did you ensure your solution was correct? Testing details, for instance. . . . .	3
4.3 what did you learn? . . . . .	4

## 1 Overall System Design

### 1.1 Overall Layout / theory

The overall design of the myar application is fairly straightforward. When a user runs the program, the command line parser parses it, and determines what sub-sections of the program to run.

It then opens the archive, after verifying that it begins with the proper ARMAG, and then executes the portion of the program that should deal with whatever action you wanted it to run.

## 1.2 Command Line Parsing

The parser here is fairly simple. Initially, I was going to use getopt() but after re-reading the requirements, it appeared you wanted all the arguments first, with the files passed on the end, not right after the argument, as per standards.

- It starts with your first argument, (the flags, or options) and passes them into a variable to deal with later.
- Next it takes your second argument, (the path to the archive) and stores it into a variable to deal with later.
- Next, It will iterate through up to 7 more arguments, saving them into an array to be dealt with later.
- Then it double checks that you have provided enough / correct arguments. If not, it dies with an error.
- Finally, It iterates through a list of acceptable arguments, and executes the proper portion of the program, based on whichever one it encounters first.
- it will check arguments in the following order:

-t

-v

-q

-x

-d

-A

errors out if nothing was found.

### 1.2.1 Problems

There were major problems with passing invalid arguments (mainly not enough arguments) and the program crashing and burning. However, Recently, most of these cases have been trapped, and should no longer occur.

## 1.3 Listing files

After the archive has been verified, and opened for editing / reading, all archives and read modes are initially parsed the same way. It starts by calling getList(), whether you wanted a quicklist, or the detailed list.

getList() fetches the address of the end of the archive, initializes variables, and then begins looping through the file. It starts immediately after ARMAG, and will read the first header, then convert the length of that header's file to an integer.

It will then lseek that far through the file, to the next file header, and read the next one. It continues through the file in this manner, until it finally seeks to or past the end of the file.

Each time, it writes the file header into the global array of structs that stores all this data.

Once that has finished, it will call `fixList()` which simply iterates through the list, removing trailing slashes, invalid files, and prettying up the list. It then falls back into whichever form of list you asked for. The simple list pretty much just outputs what is stored in the fixed list, while the detailed list is more complex.

## 1.4 Extracting Files

Extracting files with `myar` is simple. Simply use the command:

- `myar -x [name of archive] [file to extract] [another file to extract]...`

Internally, the commandline parser will parse through your arguments, and put all arguments after the name of the archive into an array to be used by this extraction process.

It then iterates through the list of files in the archive, and in a nested loop, iterates through the arguments you passed it. For all matches, it will call the function to actually extract said file to its file.

### 1.4.1 Caveats

- I have thoroughly tested with `ar`, and it does not seem to preserve directories, at least not absolute directories. Hence, `myar` will also discard absolute directories when extracting.
- If you specify the same file name twice, `myar` will sometimes extract the file twice to the same file, duplicating it twice in a single file. Avoid this.
- If the file you specified is not in the archive, No error message will be displayed at this time. Simply nothing will happen.

## 1.5 Adding Files

Sadly due to time constraints, `myar` does not yet support inserting files into an archive, or creating a new one.

Much of the code for this is in the program, and using `myar -q` will invoke the partially finished code, however, headers are not written properly, and **YOU WILL CORRUPT YOUR ARCHIVE FILE!**

## 2 Work Log

Didn't do much except for planning and prototyping before the 15th.

on the 16th, got general outline of code up and running.

Had `-t` and `-v` working on the 19th

Added command line support on 20th

Extraction added on 21st

Finished up on 22nd

Full outline here: <https://github.com/1n5aN1aC/myar/commits/master>

### 3 Challenges

I learned that strings in C are super annoying, but using some builtin methods, and making a few of your own is a lifesaver.

I had major problems with working with the struct, and I am not entirely satisfied with my solution, but the parts that are done work correctly, so...

see "what did you learn" below for more info.

### 4 Questions And Answers

#### 4.1 what do you think the main point of this assignment is?

- Make sure we know how to use all the tools of the course (C, tex, etc.)
- Get us used to basic system calls, and moving around inside of and reading/writing files.

#### 4.2 how did you ensure your solution was correct? Testing details, for instance.

Through testing was done using archives made with UNIX's ar. Following is a list of all the test cases Performed that I can remember:

- myar with no arguments
- myar with one argument (flag) but no files specified
- myar -t ex.a
- myar -v ex.a
- myar -tv ex.a (will handle first flag only)
- myar -t ex.a any amount of junk arguments
- myar -v ex.a any amount of junk arguments
- myar -t/v ex.a (where ex.a is empty) [file is corrupt]

- `myar -t/v ex.a` (where `ex.a` has a corrupted `armag`) [file is corrupt]
- `myar -x ex.a` (no files to extract)
- `myar -x ex.a derp herp` (two files not in archive)
- `myar -x ex.a thi1` (one file in archive)
- `myar -x ex.a thi1 tha2` (two files in archive)
- `myar -x ex.a thi1 derp` (one file in archive, one not)

all the above command worked properly on files created with UNIXs `ar`. There were several variations of options shown in this list that I did not list here, as they are very similar.

### 4.3 what did you learn?

- Although I previously knew some C++ this assignment has been a rude awakening on the front of c-strings, character arrays and the like
- Overall, I would say I strongly dislike C from this course so far. I like my objects in Java / Python too much.
- Although, I have mostly learned how to work with structs and character arrays in C, even though I greatly dislike it.
- C is very frustrating to code in, and takes considerably longer than Java / Python (at least for me.)

See "Challenges" above for more info.