CSI 402 – Systems Programming

Programming Assignment II

Date given: Oct. 12, 2017 **Due date:** Oct. 26, 2017
Total grade for this assignment: 100 points **Weightage:** 3%
Note: Programs that produce compiler/linker errors will receive a grade of zero.

A. Purpose. A file archiver is a computer program that combines a number of files together into one archive file, or a series of archive files, for easier storage or portability. The output of a file archiver is called an archive. Basic archivers just take a list of files and concatenate their contents sequentially into archives. More advanced archivers store additional metadata, such as the original timestamps, file attributes or access control lists. Additionally, some file archivers employ data compression to reduce the size of the archive. The Unix `tar` command is an example of an archiving tool (archives have the extension `.tar`). Another example is the collection of tools that generate `.zip` files.

Your task is to develop a modified version of the `tar` command. Obviously, you are not allowed to use the `tar`. Specifically, your program should provide the following functionality:

- Given a list of files and the desired archive name, create the archive while leaving the original files intact.
- Given a list of files, the desired archive name, and a desired archive size, create a series of archives of up to the given size while leaving the original files intact.
- Given an archive file, create the individual files contained in that archive, while leaving the archive intact.
- Given an archive file, list statistical information about the file.
- Given an archive file and a list of individual files, verify that the archive contains those files.

B. Description. The executable version of your program must be named `archiver`. Your `makefile` must ensure this. The `archiver` program must support the following usage:

```
archiver [-a|-u|-l|-v] archivename [file1 file2 ... fileN]
```

- `archiver -a archivename file1 file2 ... fileN`: Create an archive with the specified name from the specified files. The original files should remain unchanged. The number of files can vary.
- `archiver -u archivename`: "Unpack" the specified archive and generate each file contained within. The original archive should remain unchanged.
- `archiver -l archivename`: Print to `stdout` the total size of the archive, the number of files in the archive, and each filename along with the corresponding file size (one file name and size per line)
- `archiver -v archivename file1 file2 ... fileN`: determine whether or not the specified archive is damaged. There are three possibilities:

- The archive correctly contains all N files. In this case, your program should print the message "Archive verified" to `stdout`.
- The archive is missing some data. In this case, your program should print "Archive is missing X bytes" to `stdout`, where X is the number of missing bytes.
- The archive has N files, and is the correct size to contain the specified files, but some of the data is incorrect. In this case, your program should print to `stdout` the message "Archive is corrupted".

Suppose a set of N files is provided: $f_1, f_2, \ldots, f_n$. For a file f_i, let the value l_i represent the length of the file name, n_i the actual file name, s_i the size of the file in bytes, and c_i the actual contents of the file as a sequence of bytes. The corresponding archive will have the following format: (i) 4 bytes to store N, the number of files in the archive, and (ii) N file records; each file record, r_i, will store information about the corresponding f_i according to the following format: 1 byte to store l_i, $l_i + 1$ bytes to store n_i (including the null terminating character), 4 bytes to store s_i, and s_i bytes to store c_i. The resulting archive file is a binary file, which contents will "look" like the diagram below. Note that the value under each piece of data is the size, in bytes, of that piece of data, and is not part of the contents of the binary file.

N	l_1	n_1	s_1	c_1	l_2	n_2	s_2	c_2	...	l_N	n_N	s_N	c_N
4	1	l_1+1	4	s_1	1	l_2+1	4	s_2	...	1	l_N+1	4	s_N

For a more concrete example, suppose you want to archive two files "`foo.txt`" and "`blah.mpeg`", 120 and 300,000 bytes in size respectively. The resulting archive of these two files, will "look" like:

2	7	*foo.txt*\0	120	contents	9	*blah.mpeg*\0	300000	contents
4 bytes	1 byte	8 bytes	4 bytes	120 bytes	1 byte	10 bytes	4 bytes	300,000 bytes

Remarks:

- The number of files, N, is a 4-byte unsigned integer.
- The size of each file is a 4-byte unsigned integer.
- The length of each filename is $1 \leq l_i \leq 255$ (i.e., can be stored as a 1-byte unsigned char).
- The filename takes up exactly $l_i + 1$ bytes in the archive (each character takes up 1 byte, including the null-terminating character).
- The file contents in the archive should be exactly as they were in the original file. No modifications or compression should be performed.

Error Handling: For cases not covered by this specification, you may specify and implement a reasonable behavior. Additionally, your program must detect the following fatal errors. In each case, your program should produce a suitable error message to `stderr` and stop.

- An unknown switch is provided (i.e., the flag provided is not one of `-a`, `-u`, `-l`, or `-v`).
- A combination of flags is provided (e.g., both `-a` and `-l` are provided in the command line).
- Wrong number of command line arguments:
  - When the `-a` flag is provided, there should be at least 4 arguments.

  - When the `-u` flag is provided, there should be exactly 3 arguments.
  - When the `-l` flag is provided, there should be exactly 3 arguments.
  - When the `-v` flag is provided, there should be at least 4 arguments.

- An input file (i.e., archive or any file of the provided filenames) could not be opened.

C. Structural Requirements. Your submission must *at least* contain the following files:

1. A C source file with just the `main` function.
2. A C source file containing only the following function(s):
   - Function `void archive(char** filenames, int numFiles, char* archivename)` to create an archive file from a set of individual files.
   - Function `void unarchive(char* archivefile)` to unpack an archive file and create the original individual files.
3. A header file containing only the prototypes for the functions in the second item.
4. A C source file containing only the following function(s):
   - A function to handle the `-l` flag.
   - A function to handle the `-v` flag.
5. A header file containing only the prototypes for the functions in the fourth item.
6. A C source file containing the function to compute the size of a file.
7. A header file containing only symbolic constants.

C. Submission Instructions. Make sure you are logged in to ITSUnix, and your present working directory contains only the files you wish to submit. Use the `turnin-csi402` command to submit all of the .c and .h files, and any file named 'makefile'. Make sure there are no extra .c or .h files in the current directory. If your makefile is called 'Makefile' (with a capital M), then make sure you use a capital M in the turnin command. Instructions for using the `turnin-csi402` command have been provided in Programming Assignment 0, which you should have already submitted.