Assignment 4

```
The tao that can be tar(1)ed
is not the entire Tao.
The path that can be specified
is not the Full Path.
```

– `/usr/games/fortune`

Due by 11:59:59pm, Friday, November 9th.
This assignment may be done with a partner.

Program: `mytar`

This assignment is to build a file archiving tool, `mytar`, that is a version of the standard utility `tar(1)`. `Tar(1)`, standing for Tape ARchive, is one of the more ancient programs from the Unix world. It bundles files and directories together in a single file so that it can be easily transferred to some other location.

Your program must be able to build and restore archives in a way that is interoperable with GNU tar.

Running mytar

Usage:

```
mytar [ctxvS]f tarfile [ path [ ... ] ]
```

Mytar is a subset of tar and only supports five options. One of `'c'`, `'t'`, or `'x'` is required to be present. Traditionally, `'f'` is optional, but you are not required to support its being absent.

Options Supported:

c	Create an archive
t	Print the table of contents of an archive
x	Extract the contents of an archive
v	Increases verbosity
f	Specifies archive filename
S	Be strict about standards compliance

The 'f' option

The argument following the `'f'` option specifies the name of the archive file to use. Real tar uses stdin or stdout if the `'f'` is missing, but you may require `'f'` to be present for this assignment.

Archive Creation (c)

In create mode, `mytar` creates a new archive. If the archive file exists, it is truncated to zero length, then all the remaining arguments on the command line are taken as paths to be added to the archive.

Because `tar` was originally intended as a tape archiver, the order in which files are listed or extracted should be the order in which they are found in the archive. It is a much more expensive operation to back up a tape (or even a file) and re-read than to look through a list of strings to see if this is one of the files you're looking for.

`mytar` supports the following file types:

- Regular files (Regular and alternate markings)
- Directories
- Symbolic links

Any file of any other type should cause an error to be reported but no action taken.

If a given path is a directory, that directory and all the files and directories below it are added to the archive.

If the verbose (`'v'`) option is set, `mytar` lists files as they are added, one per line

Archive Listing (t)

In list (Table of contents) mode, `mytar` lists the contents of the given archive file, in order, one per line. If no names are given on the command line, `mytar`, lists all the files in the archive. If a name or names are given on the command line, `mytar` will list the given path and any and all descendants of it. That is, all files and directories beginning with the same series of directories.

If the verbose (`'v'`) option is set, mytar gives expanded information about each file as it lists them. For example:

```
% mytar tvf archive.tar
drwx------ ngonella/ngonella           0 2010-11-02 13:49 Testdir/
-rwx--x--x ngonella/ngonella          72 2010-11-02 13:49 Testdir/file1
-rw------- ngonella/ngonella         200 2010-11-02 13:49 Testdir/file2
```

The elements of each listing are the permissions, the owner/group, the size, the last modified date (mtime) and the filename. The symbolic names stored in the

header are preferable. If the symbolic names are absent, use the numeric ones. (See the description of the header below.)

The listing consists of a line of the following fields, each separated by a space:

Field	Width	Description
Permissions	10	(See below)
Owner/Group	17	Name of the file's owner
Size	8	Size of the file (in bytes)
Mtime	16	Last modification time (YYYY-MM-DD HH:MM)
Name	variable	Filename

The permissions string consists of 10 characters. The first gives the files type: `'d'` for a directory, `'l'` for a symbolic links, or `'-'` for any other type of file. The remaining nine characters indicate the presence or absence of read (`r`), write (`w`) or execute (`x`) permission for the file's owner, group, and other respectively. If a permission is not granted, write a dash (`-`).

Archive Extraction (x)

In extract mode, `mytar` extracts files from an new archive. If no names are given on the command line, `mytar`, extracts all the files in the archive. If a name or names are given on the command line, `mytar` will extract the given path and any and all descendants of it just like listing.

Extract restores the modification time of the extracted files.

Strict (S)

This option forces `mytar` to be strict in its interpretation of the standard. That is, it requires the magic number to be null-terminated and checks for the version number.

Without this option, `mytar` only checks for the five characters of "ustar" in the magic number and ignores the version field. This is required to interoperate with GNU's tar.

If "S" is not specified, must interact with GNU's tar.

Optional Extensions

Not worth anything extra, but if you get into it...

- Allow stdin/stdout (make "-" a valid argument to -f, and/or make `f` optional)

- Support for integers that do not fit in the allowed number of octal digits: When it discovers that an integer will not fit in octal in the allotted field it places it there anyway as a binary integer in network (big-endian) order. To signal that it's done this, it sets the first bit of the field to 1, then the rest is the integer. You are certainly not required to do this, but it is the most robust solution. Since it is beyond the scope of what I expected you to do, you may use the functions in the appendix to help with it.

  This is only in non-strict mode, of course.

  Why do this: Some of you have very large user IDs that will not fit in 7 octal digits. Doing this will allow you to test your mytar on your own files. If you have this problem and don't want to implement this extension, alternatives are to substitute a special uid (e.g. "7777777") or test on files owned by the system which will have lower user IDs.

Miscellaneous Restrictions

- If duplicate entries exist in an archive, use the latest one.
- Directory names stored in the archive must end in '/'

USTAR Archive Format

`mytar` implements the POSIX-specified USTAR archive format. The format of the archive is fully specified as part of POSIX, available on the web at `http://www.unix.org/single unix specification/1`. It is also described below.

File format documented at

`http://www.opengroup.org/onlinepubs/9699919799/utilities/pax.html#tag_20_92_13_06`
`/usr/include/tar.h` contains useful definitions and a description of the header fields.

Archive Format

A USTAR archive is a sequential list of records each of which consists of a header block followed by zero or more data blocks. After the last record of the file is an End of Archive marker which consists of two blocks of all zero bytes.

- All blocks are 512 bytes
- Any portion of a block not completely filled by data (e.g. the last block of a file) should be filled by zero ('\0') bytes

Header Format

Field Name	Offset	Length	Notes
name	0	100	NUL-terminated if NUL fits
mode	100	8	
uid	108	8	
gid	116	8	
size	124	12	
mtime	136	12	
chksum	148	8	
typeflag	156	1	
linkname	157	100	NUL-terminated if NUL fits
magic	257	6	must be “ustar”, NUL-terminated
version	263	2	must be “00” (zero-zero)
uname	265	32	NUL-terminated
gname	297	32	NUL-terminated
devmajor	329	8	
devminor	337	8	
prefix	345	155	NUL-terminated if NUL fits

The fields are:

name The name of the archived file is produced by concatenating the `prefix` (if of non-zero length), a slash, and the `name` field. If the prefix is of zero length, `name` is the complete name.

mode The numeric representation of the file protection modes stored as an octal number in an ASCII string terminated by one or more space or `'\0'` characters.

uid The numeric user id of the file owner, encoded as an octal number in an ASCII string terminated by one or more space or nul characters

gid The numeric group id of the file owner, encoded as an octal number in an ASCII string terminated by one or more space or nul characters.

size The size of the file encoded as an octal number in an ASCII string terminated by one or more space or nul characters. The size of symlinks and directories is zero.

mtime The last modified time of the file encoded as an octal number in an ASCII string terminated by one or more space or nul characters.

chksum The checksum is the result of adding up all the characters (treated as unsigned characters) in the header block, encoded as an octal number in an ASCII string terminated by one or more space or nul characters. For purposes of computing the checksum the checksum field itself is treated as if it were filled with spaces.

typeflag A character indicating the type of the archived file. Valid file types for mytar are:

'0'	Regular file
'\0'	Regular file (alternate)
'2'	Symbolic link
'5'	Directory

linkname If the file is of type symlink (or hard link should you implement that) this is the value of the link.

magic The magic number shall consist of the string "ustar", nul-terminated.

version The version shall be the two characters "00" (zero-zero)

uname The symbolic name of the file's owner, truncated to fit if necessary, nul-terminated.

gname The symbolic name of the file's group, truncated to fit if necessary, nul-terminated.

devmajor Major number of an archived device special file, encoded as an octal number in an ASCII string terminated by one or more space or nul characters.

devminor Minor number of an archived device special file, encoded as an octal number in an ASCII string terminated by one or more space or nul characters.

prefix See `name` field.

When creating archives, as much of the name as will fit should be placed in the `name` field, with the overflow going to `prefix`.

Other things waiting to be integrated into the narrative:

- Permissions for extracted files
  - By default, tar does not try to restore a file's archived permissions.
  - It offers rw permission to everyone, and the umask applies.
  - If any execute bits are set in the archived permissions, tar offers execute permission to all on the extracted file.
- Differences between `mytar` and `tar(1)`:
  - you don't have to handle stdin/stdout
  - Only handles regular files, directories, and symlinks
  - verbose option can be repeated
  - "S" strict option that makes it strict on conformance.
  - `mytar` will archive absolute paths. Most tar implementations refuse.
- If you want to be able to try diffing against my archive files, my version of `mytar` goes through its arguments in order and does a preorder DFS for each while inserting. It is not required to do it this way, but a successful diff is a comforting thing.

- Error handling:
  - When reading, stop at the first corrupt record. When writing, skip files you can't read, but go on. (report, of course)
  - If you find a bad header (invalid checksum, magic number, or version (as appropriate)), declare that you're lost and give up.
  - If other errors are encountered, report them with meaningful error messages and continue if possible.
  - Paths are at most 256 characters. If a path is longer than that, tar must print an error message
  - Names can only be broken on a '/'. If a name can not be partitioned, print an error and go on to the next file (if any).

Tricks and Tools

- `<stdint.h>` defines a set of fixed-width data types which are more portable than `int` when you really need to know how big something is. These exist in signed and unsigned versions named `intXX_t` and `uintXX_t`, where XX is the number of bits. For example, `uint32_t` size makes size a 32-bit unsigned integer.
- Think about your data structures
- Write a function that builds directory trees (Note: a file being in the archive does not require all its parent directories to exist.)
- Write functions that pack, unpack, and verify tar headers. (don't go on until you are sure you are reading headers accurately)
- Write a recursive traversal of the directory tree that only descends into real directories (avoid symlinks).
- Be careful not to recurse on "." or ".."
- Be careful to remember where you started. All of the arguments are relative to the original current working directory.
- Write debugging functions early so you can observe your progress.

What To Turn In

Submit via `handin` to the `asgn4` directory of the `ngonella` account:

- Your well-documented source files.
- A makefile (called Makefile) that will build your program when given the target "mytar" or no target.
- A README file that contains:
  - Your name(s). In addition to your names, please include your Cal Poly login names with it, in parentheses. E.g. (ngonella)
  - Any special instructions for running your program.
  - Any other thing you want me to know while I am grading it.

The README file should be **plain text**, i.e, not a **Word document**, and should be named "README", all capitals with no extension.

Useful Functions

`memset(3)`
`memmove(3)`
`memcpy(3)`
	Functions for manipulating blocks of memory

`strtol(3)`	For converting numbers in various bases.
`strcpy(3)` `strncpy(3)`	For copying stringsx
`lstat(2)`	to get information about a file including its type, owner, size and permissions.
`chdir(2)` `getcwd(3)`	for navigating within the filesystem.
`htonl(3)` `ntohl(3)`	transforming to and from network byte order

`opendir(3)`
`closedir(3)`
`readdir(3)`
`rewinddir(3)`
etc.
	for reading and manipulating directory entries.

`getpwuid(3)` `getgrgid(3)`	for help translating the user and group IDs returned by `lstat(2)` into names.

`readlink(2)`
`printf(3)`
`sprintf(3)`
	to read the value of a symbolic link
	for generating formatted output.

`time(2)`
`localtime(3)`
`strftime(3)`
	for handling times.

Sample runs

Below are some sample runs of `mytar`. I will also place executable versions in `~ngonella/public/csc-357/asgn4/mytar` so you can run it yourself.

```
% ls -lR Test
Test:
total 8
drwx------. 2 ngonella ngonella 4096 Nov  5 06:17 Subdir
-rw-------. 1 ngonella ngonella   96 Nov  5 06:16 file1

Test/Subdir:
total 8
-rw-------. 1 ngonella ngonella 156 Nov  5 06:17 file1
-rwx------. 1 ngonella ngonella 135 Nov  5 06:17 file2
% mytar cvf Test.tar Test
Test
Test/Subdir
Test/Subdir/file1
Test/Subdir/file2
Test/file1
% mytar tf Test.tar
Test/
Test/Subdir/
Test/Subdir/file1
Test/Subdir/file2
Test/file1
% mytar tvf Test.tar
drwx------ ngonella/ngonella              0 2010-11-05 06:17 Test/
drwx------ ngonella/ngonella              0 2010-11-05 06:17 Test/Subdir/
-rw------- ngonella/ngonella            156 2010-11-05 06:17 Test/Subdir/file1
-rwx------ ngonella/ngonella            135 2010-11-05 06:17 Test/Subdir/file2
-rw------- ngonella/ngonella             96 2010-11-05 06:16 Test/file1
% mkdir Output
% cd Output
% ls
% mytar tf ../Test.tar
Test/
Test/Subdir/
Test/Subdir/file1
Test/Subdir/file2
Test/file1
% mytar tf ../Test.tar
Test/
Test/Subdir/
Test/Subdir/file1
```

```
Test/Subdir/file2
Test/file1
% mytar xvf ../Te
Test.tar  Test/     Testdir/
% mytar xvf ../Test.tar Test/Subdir
Test/Subdir/
Test/Subdir/file1
Test/Subdir/file2
% ls -R Test/
Test/:
Subdir

Test/Subdir:
file1  file2
% mytar xvf ../Test.tar
Test/
Test/Subdir/
Test/Subdir/file1
Test/Subdir/file2
Test/file1
% ls -R Test/
Test/:
Subdir  file1

Test/Subdir:
file1  file2
%
```

Appendix

```c
#include <arpa/inet.h>
#include <string.h>

uint32_t extract_special_int(char *where, int len) {
    /* For interoperability with GNU tar. GNU seems to set the high-order bit
     * of the first byte, then treat the rest of the field as a binary integer
     * in network byte order.
     * I don't know for sure if it's a 32 or 64-bit int, but for this version,
     * we'll only support 32. (well, 31) returns the integer on success, -1 on
     * failure.  In spite of the name of htonl(), it converts int32_t
     */
    int32_t val = -1;
    if (len >= sizeof(val) && where[0] & 0x80) {
        /* the top bit is set and we have space
         * extract the last four bytes */
        val = *(int32 t *)(where + len - sizeof(val));
        val = ntohl(val);
    }

    return val;
}

int insert_special_int(char *where, size_t size, int32_t val) {
    /* For interoperability with GNU tar. GNU seems to set the high-order bit
     * of the first byte, then treat the rest of the field as a binary integer
     * in network byte order.
     * Insert the given integer into the given field using this technique.
     * Returns 0 on success, nonzero otherwise
     */
    int err = 0;
    if (val < 0 || (size < sizeof(val))) {
        /* if it's negative, bit 31 is set and we can't use the flag if len is
         * too small, we can't write it. Either way, we're done.
         */
        err++;
    } else {
        /* game on....*/
        memset(where, 0, size); /* Clear out the buffer */
        *(int32 t *)(where + size - sizeof(val)) = htonl(val); /* place the int */
        *where |= 0x80; /* set that high-order bit */
    }
    return err;
}
```