CS537 Spring 2020, Project 1a

Updates

- For wis-untar if you are provided more than one argument you should print “wis-untar: tar-file” (followed by a newline) and exit with status 1.

Administrivia

- **Due Date** by Jan 29, 2020 at 10:00 PM
- Questions: We will be using Piazza for all questions.
- Collaboration: The assignment has to be done by yourself. Copying code (from others) is considered cheating. Read this (http://pages.cs.wisc.edu/~remzi/Classes/537/Spring2018/dontcheat.html) for more info on what is OK and what is not. Please help us all have a good semester by not doing this.
- This project is to be done on the lab machines (https://csl.cs.wisc.edu/services/instructional-facilities), so you can learn more about programming in C on a typical UNIX-based platform (Linux).
- Some tests will be provided at *~cs537-1/tests/p1a*. Read more about the tests, including how to run them, by executing the command `cat ~cs537-1/tests/p1a/README` on any lab machine. Note these test cases are not complete, and you are encouraged to create more on your own.
- Handing it in: Copy your files to *~cs537-1/handin/login/p1a* where login is your CS login.

Unix utilities

In this assignment, you will build a set of linux utilities but much simpler versions of common used commands like **ls**, **cat**, **grep** etc. We say simpler because to be mildly put the original are quite complicated! We will call each of these utilities slightly different to avoid confusion - **wis-grep**, **wis-tar**, **wis-untar**.

Learning Objectives:

- Re-familiarize yourself with the C programming language
- Familiarize yourself with a shell / terminal / command-line of UNIX
- Learn about how UNIX utilities are implemented

Summary of what gets turned in:

- Bunch of *.c* files, one each for a utility : **wis-grep.c**, **wis-tar.c**, **wis-untar.c**
- Each should compile successfully when compiled with the -Wall and -Werror flags.
- Each should (hopefully) pass tests we supply.
- Include a single README.md for all the files describing the implementation.
- Assume that input files for each utility can be larger than RAM available on the system.

Before beginning: Read this lab tutorial (http://pages.cs.wisc.edu/~remzi/OSTEP/lab-tutorial.pdf); it has some useful tips for programming in the C environment. Also read the hints (p1a-hints.html) document to help you get started.

wis-grep

The first utility you will build is called **wis-grep**, a variant of the UNIX tool grep. This tool looks through a file, line by line, trying to find a user-specified search term in the line. If a line has the word within it, the line is printed out, otherwise it is not.

Here is how a user would look for the term foo in the file bar.txt:

```
        prompt> ./wis-grep foo bar.txt
        this line has foo in it
        so does this foolish line; do you see where?
        even this line, which has barfood in it, will be printed.
```

For hints on how to get started you can read more about how to open a file and read it in the hints document (p1a-hints.html).

Details

- Your program wis-grep is always passed a search term and zero or more files to grep through (thus, more than one is possible). It should go through each line and see if the search term is in it; if so, the line should be printed, and if not, the line should be skipped.
- The matching is case sensitive. Thus, if searching for **foo**, lines with **Foo** will not match.
- Lines can be arbitrarily long (that is, you may see many many characters before you encounter a newline character, \n). wis-grep should work as expected even with very long lines. For this, you might want to look into the `getline()` library call.
- If wis-grep is passed no command-line arguments, it should print "wis-grep: searchterm [file …]" (followed by a newline) and exit with status 1.
- If wis-grep encounters a file that it cannot open, it should print "wis-grep: cannot open file" (followed by a newline) and exit with status 1.
- In all other cases, wis-grep should exit with return code 0.
- If a search term, but no file, is specified, wis-grep should work, but instead of reading from a file, wis-grep should read from standard input. Doing so is easy, because the file stream stdin is already open; you can use fgets() (or similar routines) to read from it.
- For simplicity, if passed the empty string as a search string, wis-grep can either match NO lines or match ALL lines, both are acceptable.

wis-tar and wis-untar

The next two utilities you will build are simpler versions of tar and untar, which are commonly used UNIX utilities to combine (or expand) a collection of files into one file (one file into a collection of files). This functionality is useful in a number of scenarios e.g. offering a single file to download for software. (If you've heard the phrase tarball, that comes from using tar!)

The input to your **wis-tar** program will be the name of the tar file followed by a list of files that need to be archived (Fun Fact: The name tar comes from tape archives!). Example:

```
prompt> echo abcd > a.txt # creates the file a.txt
prompt> echo efgh > b.txt # creates the file b.txt
prompt> ./wis-tar test.tar a.txt b.txt
```

wis-tar format

For the purpose of this assignment we will use a simple file format for our tar file. Our format will be

```
file1 name [100 bytes in ASCII]
file1 size [8 bytes as binary]
contents of file1 [in ASCII]
file2 name [100 bytes]
file2 size [8 bytes]
contents of file2 [in ASCII]
...
```

Here are a few points that will help you with your implementation

1. You can assume that the files provided as an input exist in the directory where the program is run from (no need to handle ways to store pesky path names).
2. You can also assume the files provided as inputs only contain ASCII characters.
3. You can read more about how to find the size of a file in hints document (p1a-hints.html).

Details

- If fewer than two arguments are supplied to your program then you should print “wis-tar: tar-file file […]” (followed by a newline) and exit with status 1.
- If any of the input files that should be a part of the tar file are not found you should print “wis-tar: cannot open file” (followed by a newline) and with exit status 1.
- If a tar-file of the same name already exists you can overwrite it with the new contents specified.

- If any of the input file names are longer than 100 characters, you can only use the first 100 characters as the filename to store in the tar format.
- If any of the input file names are shorter than 100 characters, you can pad the filename such that it uses 100 bytes. For example if the file name was “a.txt”, that only has 5 characters. In this case you can append 95 NULL i.e., `\0` characters to make the name use 100 bytes in the tar file. (Remember `'\0'` is not the same as `'0'`. The first one represents NULL while the second one represents the character 0!)

EXAMPLE: Lets look at a complete example to make sure we understand the format and how to understand the contents of a valid tar file. In the following example we first create a text file which contains the string `hey`. So its size is 3 (Remember this!).

Next we run `wis-tar` to create `a.tar` as shown below. Finally we print the contents of `a.tar` using `hexdump`, a utility to print the contents of a binary file. The comments to the right explain the output of hexdump. Remember that the bytes are represented in hexadecimal format, so handy table like this (https://www.ascii-code.com/) will help you lookup the ASCII values for strings. Try to see if you can decode the contents based on the comments on the right!

```
➜  p1a cat a.txt
hey%
➜  p1a ./wis-tar a.tar a.txt
➜  p1a hexdump -v a.tar
0000000 2e61 7874 0074 0000 0000 0000 0000 0000  --> The first five bytes here contain the fi
le name a.txt
0000010 0000 0000 0000 0000 0000 0000 0000 0000
0000020 0000 0000 0000 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0000 0000 0000 0000 0000
0000040 0000 0000 0000 0000 0000 0000 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000  ---> We have padded using \0 till we hit 100
bytes
0000060 0000 0000 0003 0000 0000 0000 6568 0079  ---> The byte containing 03 indicates the fi
le size is 3. Note that this is not in ASCII!
----------------------------------------------------> The last three bytes contain the string
hey, the contents of the file
```

As you can see from the above description the fields in our tar file are packed together and there are no new lines or other separators between them.

wis-untar

As the name suggests, the wis-untar program will do the reverse of the wis-tar program. Here you will take in the name of an archive created using **wis-tar** and the program will create files corresponding to those in the archive in the same directory where the program is run from. For example

```
  prompt> ./wis-untar test.tar
  prompt> ls # should contain a.txt and b.txt
```

Details

- If no arguments are provided then you should print “wis-untar: tar-file” (followed by a newline) and exit with status 1.
- If the tar filename provided on the command line doesn’t exist you should print “wis-untar: cannot open file” (followed by a newline) and exit with status 1.
- To simplify this assignment, you can assume that if a tar file is provided and if the file exists, it matches the wis-tar format described above.
- While extracting files, if there already exist files with the same name as those in the archive, you can overwrite them.

Acknowledgments

The assignment borrows content from assignment 1 of Prof. Remzi Arpaci-Dusseau’s course in Spring 2018.