Department of Computer Science and Software Engineering

CITS2002 Systems Programming

Project 2 2020, see also: Project 2 clarifications

A medium-sized business has decided to migrate its files to cloud-based storage, requiring it to first identify all files to migrate. A critical disk failure at the worst possible time now requires all files to be recovered from recent backups. However, the business' IT wizard has recently left for a lucrative position at a cloud-based storage company.

Management has located the backups, but they have been poorly labeled, making it impossible to easily identify what is contained in each backup and when each was made. The decision has been made to migrate just the latest copy of each file to the cloud, which will require an 'intelligent merging' of the backups' contents.

The backups have been made using the widely available **tar command**, a well-defined file format whose name is a contraction of ***t****ape* ***ar****chive*, reflecting the backup media with which the command was first used. While the *tar* command supports many actions to create, list, extract, and append *tar archive files*, it offers no support to merge archives together.

The business has located many backups each holding thousands of files. The task to identify all duplicate files, and to find the most recent version of similar files, is too large to be performed manually, and your team has been contracted to develop a new command-line utility program to intelligently merge all of the backups' contents into a single (large) *tar* archive.

The **goal** of this project is to write a command-line utility program in C99, named *mergetars*, which merges the contents of multiple *tar* archive files into a single *tar* archive file.

Successful completion of the project will develop your understanding of advanced features of the C99 programming language, and many useful Linux operating system system-calls and POSIX function calls.

Program invocation

The purpose of your *mergetars* command-line utility is to merge the contents of multiple *tar* archive files into a single *tar* archive. The program receives the name of one or more input filenames, and a single output filename (if only a single input filename is provided, then *mergetars* will act like a simple file-copying program, although there is no requirement to check for this special case). A typical program invocation is:

```
prompt>  ./mergetars input_tarfile1 [input_tarfile2 ...] output_tarfile
```

Filenames will always end with the suffix *.tar* – indicating that the archive does not involve any compression – or with the suffix *.tar.gz* or *.tgz* – indicating that the archive is (or will be) compressed using the *GZIP* compression algorithm. The standard *tar* utility supports these cases using its *-z* command-line option. There is no requirement for *mergetars* to support any other compression schemes.

The merging criteria

The inputs are merged to form the output according to the following definitions and rules:

- Two or more files from different archives are considered the **same file** if they have the same relative pathname (including the name of the directory holding the file and the filename itself).

  If from different archives, the files *"project2/src/mergetars.c"* and *"project2/src/mergetars.c"* are considered the same file.
  In contrast, *"monday/project2/src/mergetars.c"* and *"wednesday/project2/src/mergetars.c"* are considered different files.

- Two or more files from different archives with the same relative pathname, are considered **different versions** of the same file. The output archive should contain just the *latest* of all such versions.
  Added: If two or more files have the same modification time, then the largest of these should be copied. If two or more files have the same modification time and size, the file from the latest tarfile (on the command-line) should be copied.

- All other files with different relative pathnames are considered **different files**. The output archive should contain one copy of each different file.

Suggested approach

The project can be completed by following these recommended (but not required) steps:

1. Learn how to use the standard *tar* utility from the command-line.
2. Create two or more 'similar' *tar* archives, and determine (by hand) which files should appear in a merged archive.
3. Create 'skeleton' C source files containing 'empty' functions for each distinct responsibility. Create and test a *Makefile* to compile and link the files.
4. Then, check the program's command-line arguments.
5. Use your system's standard *tar* utility (called from your C99 code) to expand each input tar archive into a new directory.
6. Identify and copy (to a new directory structure) all files that should be uniquely added to the output tar archive. Remember to set the modification time of each file, appropriately.
7. Use your system's standard *tar* utility to create the new output tar archive.
8. Cleanup up before exiting, removing any temporary files and directories that you have created.

It is anticipated (though not required) that a successful project will use (some of) the following system-calls, and standard C99 & POSIX functions:

- *perror()*, *exit()*,
- *mkdtemp()*, *mkdir()*, *opendir()*, *readdir()*, *stat()*, *closedir()*,
- *fork()*, *execl()*, *wait()*,
- *open()*, *read()*, *write()*, *close()*, *utimes()*,
- *strcpy()*, *strcmp()*, *strdup()*,
- *malloc()*, *calloc()*, *realloc()*, and *free()*.

Project requirements

1. Your project **must** be developed in multiple source files and **must** be compiled and linked using a *Makefile*, containing approriate variable definitions and automatic variables.

2. The default target of your *Makefile* **must** be named *mergetars*, and its execution must produce a program named *mergetars*.

3. Your program **must** 'clean up after itself'. If your program creates any temporary files or directories, then these must all be removed before your program exits. Your program *does not* have to free all of its dynamically allocated memory before it exits.

4. Your project **must** employ sound programming practices, including the use of meaningful comments, well chosen identifier names; appropriate choice of basic data-structures, data-types, and functions; and appropriate choice of control-flow constructs.

Assessment

The project is due **5PM Fri 16th October**, and is worth **30% of your final mark** for CITS2002.
It will be marked out of 40. The project may be completed **individually or in teams of two** (but not teams of three). The choice of project partners is up to you - you will not be automatically assigned a project partner.

You are **strongly** advised to work with another student who is around the same level of understanding and motivation as yourself. This will enable you to discuss your initial design together, and to assist each other to develop and debug your joint solution. Work together - do not attempt to split the project into two equal parts, and then plan to meet near the deadline to join your parts together.

20 of the possible 40 marks will come from the correctness of your solution. The remaining 20 marks will come from your programming style, including your use of meaningful comments; well chosen identifier names; appropriate choice of basic data-structures, data-types and functions; and appropriate choice of control-flow constructs.

During the marking, attention will obviously be given to the correctness of your solution. However, a correct and efficient solution should not be considered as the perfect, nor necessarily desirable, form of solution. Preference will be given to well presented, well documented solutions that use the appropriate features of the language to complete tasks in an easy to understand and easy to follow manner. That is, do not expect to receive full marks for your project simply because it works correctly. Remember, a computer program should not only convey a message to the computer, but also to other human programmers.

Submission requirements

1. The deadline for the project is **5PM Friday 16th October (end of week 11)**.

2. **You must submit your project electronically using *cssubmit*.**

   You should submit ALL C99 source-code (*.c) and header (*.h) files and a *Makefile* that specify the steps required to compile and link your application. You **do not need to submit** any tar files or testing scripts that you used while developing and testing your project. You can submit multiple files in one submission by first archiving them with *zip* or *tar*.

   The *cssubmit* program will display a receipt of your submission. You should print and retain this receipt in case of any dispute. Note also that *cssubmit* does not archive submissions and will simply overwrite any previous submission with your latest submission.

3. **At least one** of your submission's C99 source files **must** contain C99 block comment:

   ```
   //  CITS2002 Project 2 2020
   //  Name(s):             student-name1 (, student-name2)
   //  Student number(s):   student-number1 (, student-number2)
   ```

4. If working as a team, only one team member should make the team's submission.

5. Your submission will be examined, compiled, and run on a contemporary **Linux system** (such as Ubuntu). Your submission should work as expected on this platform. While you may develop your project on other computers, excuses such as *"it worked at home, just not when you tested it!"* will not be accepted.

6. This project is subject to UWA's Policy on Assessment - particularly §10.2 *Principles of submission and penalty for late submission*. In accordance with this policy, you may *discuss* with other students the general principles required to understand this project, but the work you submit must be the result of your own efforts. All projects will be compared using software that detects significant similarities between source code files. Students suspected of plagiarism will be interviewed and will be required to demonstrate their full understanding of their project submission.

Good luck!

Chris McDonald.

The University of Western Australia
University Homepage
Future Students
Current Students
Staff
Business and Industry
Alumni and Friends
Media

Department of Computer Science and Software Engineering
Courses
Research
Business and industry
Community
Alumni
Current Students
Staff
Contact us

University information
CRICOS Code: 00126G
Accessibility
Campus map
Contact the University
Indigenous Commitment
Terms of use

This Page
Last updated:
Tue Oct 13 2020 13:44:50
GMT+0800 (AWST)
Written by:
Chris.McDonald@uwa.edu.au