Assignment 2: a file archiver

NOTE:

You may find the assignment ov [illegible] l place to start.

Contents

Aims

- building a concrete understanding of file system objects;
- practising C, including byte-level operations and robust error handling;
- understanding file operations, including input-output operations on binary data

The Task

A file archive is a single file which can contain the contents, names and other metadata of multiple files. These can make backup and transport of files more convenient, and can often make compression more efficient. We often refer to tools that can create or manipulate these as *file archivers*.

There are a vast number of archive formats: on *nix-like systems, *tar* is common; whereas on Windows, Zip is common. Wikipedia's list of archive formats is a marvellous rabbit-hole to explore.

In this assignment, you will be implementing **space**, a file archiver for the galaxy format.

The galaxy format is made up of one or more stars; where a star records one file system object; This format is described in more detail below.

A complete implementation of space can

- list the path names of each object in a galaxy (subset 0);
- list the permissions of each object in a galaxy (subset 0);
- list the size (number of bytes) of files in a galaxy (subset 0);
- check the star magic number (subset 0);
- extract files from a galaxy (subset 1);
- check a galaxy for integrity, by checking star hashes; (subset 1);
- set the file permissions of files extracted from a galaxy (subset 1);
- create a galaxy from a list of files (subset 2);

- list, extract, and create galaxies that include directories (subset 3); and
- extract, and create galaxies in 7-bit and 6-bit formats (subset 3).

Getting Started

Create a new directory for this assignment, change to this directory, and fetch the provided code by running

```
$ mkdir -m 700 space
$ cd space
$ 1521 fetch space
```

If you're not working at CSE, you can [illegible]es as a zip file or a tar file.

This will give you the following files:

`space.c` i[illegible] change: it contains partial definitions of four functions, *list_galaxy*, [illegible]xy, and *create_galaxy*, to which you need to add code to complete the a[illegible] add your own functions to this file.

`space_main.c` [illegible]s code to parse the command line arguments, and which then calls one of *list_galaxy*, *extract_galaxy*, *create_galaxy*, or *check_galaxy*, depending on the command line arguments given to space. *Do not change this file.*

`space.h` contains shared function declarations and some useful constant definitions. *Do not change this file.*

`space_hash.c` contains the *star_hash* function; you should call this function to calculate hashes for subset 1. *Do not change this file.*

`space_6_bit.c` contains the *star_to_6_bit* and *star_from_6_bit* functions. You should call these to implement the 6-bit format for subset 3. *Do not change this file.*

`space.mk` contains a Makefile fragment for space.

You can run *make* to compile the provided code; and you should be able to run the result.

```
$ make
dcc -c -o space.o space.c
dcc -c -o space_main.o space_main.c
dcc -c -o space_hash.o space_hash.c
dcc -c -o space_6_bit.o space_6_bit.c
dcc space.o space_main.o space_hash.o space_6_bit.o -o space
$ ./space -l a.galaxy
list_galaxy called to list galaxy: 'a.galaxy'
```

If you don't have *make* available you can compile like this:

```
$ dcc space.c space_main.c space_hash.c space_6_bit.c -o space
$ ./space -C b.galaxy
check_galaxy called to check galaxy: 'a.galaxy'
```

You may optionally create extra `.c` or `.h` files.

You should run *unzip* to get a directory called `examples/` full of *.galaxy* files to test your program against.

```
$ unzip examples.zip
```

Subset 0

To complete subset 0, you need to implement code that can

- print a list of the contents of a galaxy, and
- print a detailed list of the contents of a galaxy.

Subset 0: Print a list of the contents of a galaxy

Given the `-l` command-line argument, *space* should print the path names of the files/directories in a galaxy.

For example:

```
# List each item in the galaxy called text_file.galaxy, which is in the examples directory
$ ./space -l examples/text_file.galaxy
hello.txt
# List each item in the galaxy called 4_files.galaxy, which is in the examples directory
$ ./space -l examples/4_files.galaxy
256.bin
hello.txt
last_goodbye.txt
these_days.txt
# List each item in the galaxy called hello_world.galaxy, which is in the examples directory
$ ./space -l examples/hello_world.galaxy
hello.c
hello.cpp
hello.d
hello.go
hello.hs
hello.java
hello.js
hello.pl
hello.py
hello.rs
hello.s
hello.sh
hello.sql
```

Subset 0: Print a detailed list of the contents of a galaxy

Given the `-L` command-line argument, *space* should, for each file in the specified galaxy, print:

1. the file/directory permissions,
2. the star format which will be one of 6, 7 or 8 (the default),
3. the file/directory size in bytes, and
4. the file/directory path name.

```
$ ./space -L examples/text_file.galaxy
-rw-r--r--  8     56  hello.txt
# List the details of each item in the galaxy called 4_files.galaxy, which is in the examples directory
$ ./space -L examples/4_files.galaxy
-rw-r--r--  8    256  256.bin
-rw-r--r--  8     56  hello.txt
-r--r--r--  8    166  last_goodbye.txt
-r--rw-r--  8    148  these_days.txt
# List the details of each item in the galaxy called hello_world.galaxy, which is in the examples directory
$ ./space -L examples/hello_world.galaxy
-rw-r--r--  8     93  hello.c
-rw-r--r--  8     82  hello.cpp
-rw-r--r--  8     65  hello.d
-rw-r--r--  8     77  hello.go
-rw-r--r--  8     32  hello.hs
-rw-r--r--  8    117  hello.java
-rw-r--r--  8     30  hello.js
-rwxr-xr-x  8     47  hello.pl
-rwxr-xr-x  8    103  hello.py
-rw-r--r--  8     45  hello.rs
-rw-r--r--  8    123  hello.s
-rwxr-xr-x  8     41  hello.sh
-rw-r--r--  8     24  hello.sql
```

HINT:

`space_main.c` calls the function *list_galaxy* in `space.c` when either of the `-l` or `-L` options are specified on the command line.

Add code to *list_galaxy* in `space.c`.

Use *fopen* to open the galaxy file.

Use *fgetc* to read bytes.

Make sure you understand the star format specification below

Use C bitwise operations such as `<<` `&` and `|` to combine bytes into integers.

Think carefully about the functions you can construct to avoid repeated code.

Review `print_bytes.c` from our week 8 lab.

fseek can be used to skip over parts of the galaxy file, but you can also use a loop and fgetc

NOTE:

The order you list files is the order they appear in the galaxy.

galaxy files do not necessarily end with .galaxy. This has been done with the provided example files purely as a convenience.

Hint: use a format like `"%5lu"` to print the file size.

Subset 1

To complete subset 1, you need to im

- check the contents of a galaxy,
- extract files from a galaxy.

Subset 1: Check the contents of a galaxy

Given the `-C` command-line argument, *space* should check the hashes in the specified galaxy. For example:

```
# Check the galaxy called 4_files.galaxy, which is in the examples directory
$ ./space -C examples/4_files.galaxy
256.bin - correct hash
hello.txt - correct hash
last_goodbye.txt - correct hash
these_days.txt - correct hash
# Check the galaxy called examples/hello_world.bad_hash.galaxy, which is in the examples directory
$ ./space -C examples/hello_world.bad_hash.galaxy
hello.c - correct hash
hello.cpp - correct hash
hello.d - correct hash
hello.go - correct hash
hello.hs - correct hash
hello.java - correct hash
hello.js - correct hash
hello.pl - correct hash
hello.py - correct hash
hello.rs - correct hash
hello.s - correct hash
hello.sh - correct hash
hello.sql - incorrect hash 0x19 should be 0x43
```

It should also check the star magic number (first byte) of each star, and emit an error if it is incorrect.

```
# Check the galaxy called text_file.bad_magic.galaxy, which is in the examples directory
$ ./space -C examples/text_file.bad_magic.galaxy
error: incorrect first star byte: 0x39 should be 0x63
```

HINT:

`space_main.c` calls the function *check_galaxy* in `space.c` when the `-C` option is specified on the command line.

Add code to `check_galaxy` in `space.c`.

Call `star_hash` to calculate hash values.

Think carefully about the functions you can construct to avoid repeated code.

For example, for every byte you read with `fgetc` you need to call `star_hash` to calculate a new hash value, so write a function that does both. Hint: have the function take a pointer to a hash value which it can update.

Subset 1: Extract files from a galaxy

Given the `-x` command-line argument, *space* should extract the files in the specified galaxy.

It should set file permissions for extracted files to the permissions specified in the galaxy.

```
# space will extract files into the current working directory.
# So as not to clutter your assignment directory, you should create a
# temporary directory, 'tmp', and change to it. Once in that directory,
# both your space program and 'examples/' will be in its parent
# directory --- hence the use of '..' in these path names.

# Make a directory called tmp.
$ mkdir -p tmp/
# Change into the tmp directory.
$ cd tmp/
# Forcibly remove all files ins
$ rm -f * .*
# Use your program to extract t                         le.galaxy.
$ ../space -x ../examples/text_
Extracting: hello.txt
# Show the contents of hello.t
# You can manually open it in y                         if you like.
$ cat hello.txt
Hello COMP1521
I hope you are enjoying this assignment.
# Forcibly remove all files inside the tmp directory.
$ rm -f * .*
# Use your program to extract the contents of hello_world.galaxy
$ ../space -x ../examples/hello_world.galaxy
Extracting: hello.c
Extracting: hello.cpp
Extracting: hello.d
Extracting: hello.go
Extracting: hello.hs
Extracting: hello.java
Extracting: hello.js
Extracting: hello.pl
Extracting: hello.py
Extracting: hello.rs
Extracting: hello.s
Extracting: hello.sh
Extracting: hello.sql
# Show the first 25 lines from the extracted files to confirm the extraction was successful.
$ cat $(echo * | sort) | head -n 25
extern int puts(const char *s);

int main(void)
{
    puts("Hello, World!");
    return 0;
}
#include <iostream>

int main () {
  std::cout << "Hello, world!" << std::endl;
}
import std.stdio;

void main() {
    writeln("Hello, world!");
}
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
main = putStrLn "Hello, World!"
# Forcibly remove all files inside the tmp directory
$ rm -f * .*
# Use your program to extract the contents of meta.galaxy.
$ ../space -x ../examples/meta.galaxy
Extracting: 1_file.subdirectory.7-bit.galaxy
Extracting: 1_file.subdirectory.galaxy
Extracting: 2_files.7-bit.galaxy
```

```
Extracting: 2_files.galaxy
Extracting: 3_files.7-bit.galaxy
Extracting: 3_files.bad_hash.galaxy
Extracting: 3_files.bad_magic.galaxy
Extracting: 3_files.galaxy
Extracting: 3_files.subdirectory.7-bit.galaxy
Extracting: 3_files.subdirectory.bad_hash.galaxy
Extracting: 3_files.subdirectory.bad_magic.galaxy
Extracting: 3_files.subdirectory.galaxy
Extracting: 4_files.galaxy
Extracting: all_the_modes.subd
Extracting: all_the_modes.subd
Extracting: all_three_formats.
Extracting: binary_file.galaxy
Extracting: hello_world.7-bit.
Extracting: hello_world.bad_has
Extracting: hello_world.bad_mag
Extracting: hello_world.galaxy
Extracting: lecture_code.subdi
Extracting: lecture_code.subdirectory.galaxy
Extracting: small.6-bit.galaxy
Extracting: small.7-bit.galaxy
Extracting: small.galaxy
Extracting: text_file.7-bit.galaxy
Extracting: text_file.bad_hash.galaxy
Extracting: text_file.bad_magic.galaxy
Extracting: text_file.galaxy
Extracting: tiny.6-bit.galaxy
Extracting: tiny.7-bit.galaxy
Extracting: tiny.galaxy
# Show the first 10 items in this directory alphabetically to check extraction was successful.
$ ls -1 $(echo * | sort) | head
1_file.subdirectory.galaxy
1_file.subdirectory.compressed.galaxy
2_files.galaxy
2_files.compressed.galaxy
3_files.bad_hash.galaxy
3_files.bad_magic.galaxy
3_files.galaxy
3_files.compressed.galaxy
3_files.subdirectory.bad_hash.galaxy
3_files.subdirectory.bad_magic.galaxy
# Go back into the directory with your code.
$ cd ../
# Remove the tmp directory and everything inside it.
$ rm -rf tmp/
```

HINT:

`space_main.c` calls the function *extract_galaxy* in `space.c` when the `-x` option is specified on the command line.

Add code to *extract_galaxy* in `space.c` .

Use fopen to open each file you are extracting.

Use fputc to write bytes to each file.

In our lectures on files we covered copying bytes to a file in the cp_fgetc.c example and setting the permissions of a file in the chmod.c example.

NOTE:

space should overwrite an files that already exist.

space can leave already extracted/partially extracted files in the event of an error.

Subset 2

To complete subset 2, you need to implement code that can

- create a galaxy from a list of files.

Subset 2: Create a galaxy from a list of files

Given the `-c` command-line argument, *space* should create a galaxy containing the specified files.

```
# These "echo" lines show you how to create these test files and what their contents are.

# Create a file called hello.txt with the contents "hello".
$ echo hello >hello.txt
# Create a file called hola.txt with the contents "hola".
$ echo hola >hola.txt
# Create a file called hi.txt [illegible]
$ echo hi >hi.txt
# Set the permissions of these [illegible]ermission string (equivalent to rw-r--r--)).
# When you list the contents o[illegible]issions should match this.
$ chmod 644 hello.txt hola.txt [illegible]
# Create a galaxy called selama[illegible]es hello.txt, hola.txt, and hi.txt.
$ ./space -c selamat.galaxy hel[illegible]
Adding: hello.txt
Adding: hola.txt
Adding: hi.txt
# List the contents of selamat.galaxy.
$ ./space -L selamat.galaxy
-rw-r--r-- 8 6 hello.txt
-rw-r--r-- 8 5 hola.txt
-rw-r--r-- 8 3 hi.txt
# Make a directory called tmp.
$ mkdir -p tmp/
# Change into the tmp directory.
$ cd tmp/
# Forcibly remove all files inside the tmp directory.
$ rm -f * .*
# Use your program to extract the contents of selamat.galaxy.
$ ../space -x ../selamat.galaxy
Extracting: hello.txt
Extracting: hola.txt
Extracting: hi.txt
# Check that the extracted file hello.txt is the same as the source file ../hello.txt.
$ diff -s ../hello.txt hello.txt
Files ../hello.txt and hello.txt are identical
# Check that the extracted file hola.txt is the same as the source file ../hola.txt.
$ diff -s ../hola.txt hola.txt
Files ../hola.txt and hola.txt are identical
# Check that the extracted file hi.txt is the same as the source file ../hi.txt.
$ diff -s ../hi.txt hi.txt
Files ../hi.txt and hi.txt are identical
# Go back into the directory with your code.
$ cd ../
# Remove the tmp directory and everything inside it.
$ rm -rf tmp/
```

It is also possible to append stars to an existing galaxy file using the `-a` command-line option. For example:

```
$ ./space -a bonjour.galaxy hello.txt
Adding: hello.txt
$ ./space -L bonjour.galaxy
-rw-r--r--  8      6  hello.txt
$ ./space -a bonjour.galaxy hola.txt hi.txt
Adding: hola.txt
Adding: hi.txt
$ ./space -L bonjour.galaxy
-rw-r--r--  8      6  hello.txt
-rw-r--r--  8      5  hola.txt
-rw-r--r--  8      3  hi.txt
```

HINT:

`space_main.c` calls the function *create_galaxy* in `space.c` when either of the `-c` or `-a` options are specified on the command line.

Add code to *create_galaxy* in `space.c`.

Use *fopen* and *fputc* to create the new galaxy.

In our lectures on files we covered obtaining file metadata including its size and mode (permissions) in the stat.c example.

NOTE:

You must add/store files in the order they are given.

Subset 3

To complete subset 3, you need to in

- create a galaxy from a list of file
- extract directories from a galax
- manipulate 6-bit and 7-bit stora

Subset 3: Create a galaxy and directories

Given the `-c` command-line argument, *space* should be able to add files in sub-directories. For example:

```
# Create a galaxy called a.galaxy with the file "hello.txt" that is contained within 2 levels of directories.
$ ./space -c a.galaxy examples/2_files.d/hello.txt
Adding: examples
Adding: examples/2_files.d
Adding: examples/2_files.d/hello.txt
```

If a directory is specified when creating a galaxy, space should add the entire directory tree to the galaxy.

```
# Create a galaxy called a.galaxy with *all* the contents within the directory "3_files.subdirectory.d"
# which is in the "examples" directory.
$ ./space -c a.galaxy examples/3_files.subdirectory.d
Adding: examples
Adding: examples/3_files.subdirectory.d
Adding: examples/3_files.subdirectory.d/goodbye
Adding: examples/3_files.subdirectory.d/goodbye/last_goodbye.txt
Adding: examples/3_files.subdirectory.d/hello
Adding: examples/3_files.subdirectory.d/hello/hello.txt
Adding: examples/3_files.subdirectory.d/these_days.txt
```

Given the `-L` command-line argument and a galaxy containing directories, *space* should be able to list files and directories. For example:

```
$ ./space -L examples/1_file.subdirectory.galaxy
drwxr-xr-x  8      0  hello
-rw-r--r--  8     56  hello/hello.txt
```

HINT:

In our lectures on files we covered listing a directory's contents in the list_directory.c example.

Traversing a directory tree is challenging and can be done in several ways.

NOTE:

The space reference implementation will add subdirectories in alphabetical order. You do not need to match this behaviour: your implementation can add subdirectories in any order.

If a file in a different directory is added to a galaxy, then the directories in the path need to be added to the galaxy.

Subset 3: Extract directories from a galaxy

Given the `-x` command-line argument, and a galaxy containing directories, *space* should be able to extract files and directories. For example:

```
$ ./space -x examples/3_files.subdirectory.galaxy
Creating directory: goodbye
Extracting: goodbye/last_goodbye.txt
Creating directory: hello
Extracting: hello/hello.txt
Extracting: these_days.txt
```

HINT:

In our lectures on files we covered creating a directory in the mkdir.c example

NOTE:

When extracting a galaxy with directories, the directory needs to be created if it does not already exist, and its permissions need to be set to those specified in the galaxy.

Subset 3: Manipulate 6-b e formats

The `-7` and `-6` options allow stars 6-bit format. For example:

```
$ ./space -7 -c seven.galaxy h
Adding: hello.txt
$ ./space -L seven.galaxy
-rw-r--r-- 7 6 hello.txt
$ ./space -6 -c six.galaxy hola.txt hi.txt
Adding: hola.txt
Adding: hi.txt
$ ./space -L six.galaxy
-rw-r--r--  6      5  hola.txt
-rw-r--r--  6      3  hi.txt
```

It is possible for galaxies to contain stars in multiple formats. For example:

```
$ ./space -a mixed.galaxy hello.txt
Adding: hello.txt
$ ./space -L mixed.galaxy
-rw-r--r--  8      6  hello.txt
$ ./space -7 -a mixed.galaxy hi.txt
Adding: hi.txt
$ ./space -L mixed.galaxy
-rw-r--r--  8      6  hello.txt
-rw-r--r--  7      3  hi.txt
$ ./space -6 -a mixed.galaxy hola.txt
Adding: hola.txt
$ ./space -L mixed.galaxy
-rw-r--r--  8      6  hello.txt
-rw-r--r--  7      3  hi.txt
-rw-r--r--  6      5  hola.txt
```

Your code should handle creating, listing, checking, and extracting galaxies in 7-bit and 6-bit format.

Your code should produce an error if asked to create a star containing bytes which can be encoded in the specified format. For example:

```
$ echo Hello >Hello.txt
$ ./space -6 -c broken.galaxy Hello.txt
error: byte 0x48 can not be represented in 6-bit format
```

HINT:

The functions *star_to_6_bit* and *star_from_6_bit* in `space_6_bit.c` convert 8-bit values to and from 6-bit format.

Handling Errors

Error checking is an important part of this assignment. Automarking will test error handling.

Error messages should be one line (only) and be written to `stderr` (not `stdout`).

space should `exit` with status 1 after an error.

space should check all file operations for errors.

As much as possible match the reference implementation error messages exactly.

The reference implementation uses *perror* to report errors from file operations and other system calls.

It is not necessary to remove files and directories already created or partially created when an error occurs.

You may extract a file or directory from star before determining if the star hash is correct.

You can extract previous file or directory from a star.

Where multiple errors messages could be produced, for example, if two non-existent files are specified to be added to a galaxy, *space* may produce any one of the error messages.

Reference implementation

A reference implementation is a common, efficient, and effective method to provide or define an operational specification; and it's something you will likely work with after you leave UNSW.

We've provided a reference implementation, `1521 space`, which you can use to find the correct outputs and behaviours for any input:

```
$ 1521 space -L examples/tiny.6-bit.galaxy
-rw-r--r--  6      0  a
```

Every concrete example shown below [illegible] reference implementation; run `1521 space` instead of `./space`.

Where any aspect of this assignment [illegible] fication, you should match the behaviour exhibited by the reference implementation. Discovering and ma[illegible] mentation's behaviour is deliberately a part of this assignment.

If you discover what you believe to be [illegible] implementation, please report it in the class forum. If it is a bug, we may fix the bug; or otherwise indicate that you d[illegible] reference implementation's behaviour in that specific case.

The galaxy and star format

galaxies must follow exactly the format produced by the reference implementation.

A galaxy consists of a sequence of one or more stars. Each star contains the information about one file or directory.

The first byte of a galaxy file is the first byte of the first star. That star is immediately followed by either another star, or by the end of the galaxy file.

name	length	type	description
magic number	1 B (byte)	unsigned, 8-bit, little-endian	byte 0 in every star must be 0x63 (ASCII `'c'`)
star format	1 B (byte)	unsigned, 8-bit, little-endian	byte 1 in every star must be one of 0x36, 0x37, 0x38 (ASCII `'6'`, `'7'`, `'8'`)
permissions	10 B (byte)	characters	bytes 2—11 are the type and permissions as a *ls*-like character array; e.g., `"-rwxr-xr-x"`
pathname length	2 B (byte)	unsigned, 16-bit, little-endian	bytes 12—13 are an unsigned 2-byte (16-bit) little-endian integer, giving the length of
pathname	*pathname-length*	characters	the filename of the object in this star.
content length	6 B (byte)	unsigned, 48-bit, little-endian	the next bytes are an unsigned 6-byte (48-bit) little-endian integer giving the length of the file that was encoded to give
content	*content-length* for 8-bit format, see below for other formats	bytes	the data of the object in this star.
hash	1 B (byte)	unsigned, 8-bit, little-endian	the last byte of a star is a star-hash of all bytes of this star except this byte.

star content encodings (Subset 3 only)

- **8-bit format (star format == 0x38** *contents* is an array of bytes, which are exactly equivalent to the bytes in the original file.
- **7-bit format (star format == 0x37)** *contents* is an array of bytes representing packed seven-bit values, with the trailing bits set to zero. Every byte of the original file is taken as a seven-bit value, and packed as described below. This format can store any seven bit value — so, for example, any byte containing valid ASCII can be stored.

  This format needs $\lceil (7.0/8) * \text{content-length} \rceil$ bytes. 7-bit format is used only in subset 3.
- **6-bit format (star format == 0x36)** *contents* is an array of bytes of packed six-bit values where the trailing bits in the last byte are zero, and which are translated using the functions *star_to_6_bit* and *star_from_6_bit* in *space_6_bit.c*.

  This format cannot store all ASCII values, for example upper case letters can't be stored in 6-bit format.

  This format needs $\lceil (6.0/8) * \text{content-length} \rceil$ bytes.

7-bit and 6-bit format is used only in subset 3.

Packed *n*-bit encoding (Subset 3 only)

We often store smaller values inside larger types. For example, the integer 42 only needs six bits; but we often will store it in a full thirty-two-bit integer, wasting many bits of zeroes. Assuming we know how many bits the value needs, we could only store the relevant bits.

For example, let's say we have three seven-bit values *a*, *b*, *c*, made up of arbitrary bit-strings, and stored in eight-bit variables

a: `0b0AAA_AAAA` ,
b: `0b0BBB_BBBB` ,
c: `0b0CCC_CCCC` ,

then a packed seven-bit encoding of these values in order would be:

```
0bAAAA_AAAB_BBBB_BBCC_CCCC_C???
```

However, we have a problem: what happens to the trailing bits, which don't have a value? Note that we've defined all trailing bits to be zero above, which would here give:

```
0bAAAA_AAAB_BBBB_BBCC_CCCC_C000
```

Inspecting galaxies ar[illegible]

The *hexdump* utility can show the inc[illegible] can use this to inspect galaxies and stars.

For example, here is a galaxy, made u[illegible]

```
$ hexdump -vC examples/2_files.[illegible]
00000000 63 38 2d 72 77 2d 72 2d 2d 72 2d 2d 09 00 68 65 |c8-rw-r--r--..he|
00000010 6c 6c 6f 2e 74 78 74 38 00 00 00 00 00 48 65 6c |llo.txt8.....Hel|
00000020 6c 6f 20 43 4f 4d 50 31 35 32 31 0a 49 20 68 6f |lo COMP1521.I ho|
00000030 70 65 20 79 6f 75 20 61 72 65 20 65 6e 6a 6f 79 |pe you are enjoy|
00000040 69 6e 67 20 74 68 69 73 20 61 73 73 69 67 6e 6d |ing this assignm|
00000050 65 6e 74 2e 0a 2d 63 38 2d 72 77 2d 72 2d 2d 72 |ent..-c8-rw-r--r|
00000060 2d 2d 10 00 6c 61 73 74 5f 67 6f 6f 64 62 79 65 |--..last_goodbye|
00000070 2e 74 78 74 a6 00 00 00 00 00 54 68 69 73 20 69 |.txt......This i|
00000080 73 20 6f 75 72 20 6c 61 73 74 20 67 6f 6f 64 62 |s our last goodb|
00000090 79 65 0a 49 20 68 61 74 65 20 74 6f 20 66 65 65 |ye.I hate to fee|
000000a0 6c 20 74 68 65 20 6c 6f 76 65 20 62 65 74 77 65 |l the love betwe|
000000b0 65 6e 20 75 73 20 64 69 65 0a 42 75 74 20 69 74 |en us die.But it|
000000c0 27 73 20 6f 76 65 72 0a 4a 75 73 74 20 68 65 61 |'s over.Just hea|
000000d0 72 20 74 68 69 73 20 61 6e 64 20 74 68 65 6e 20 |r this and then |
000000e0 49 27 6c 6c 20 67 6f 0a 59 6f 75 20 67 61 76 65 |I'll go.You gave|
000000f0 20 6d 65 20 6d 6f 72 65 20 74 6f 20 6c 69 76 65 | me more to live|
00000100 20 66 6f 72 0a 4d 6f 72 65 20 74 68 61 6e 20 79 | for.More than y|
00000110 6f 75 27 6c 6c 20 65 76 65 72 20 6b 6e 6f 77 0a |ou'll ever know.|
00000120 60 |`|
00000121
```

Each line of *hexdump* output is in three groups:

- the **address column**: this starts at `0x00000000` , and increases by 0x10 (or 16 in base 10) each line;
- the **data columns**: after the address, we get (up to) 16 two-digit hexadecimal values, grouped into two blocks of eight values each, which represents the actual data of the file, and
- the **human readable stripe**: at the very end of each line, between the vertical bars (`|`) is the human readable version of the bytes preceding, or a `'.'` if the byte wouldn't ordinarily be visible.

You could also use the *hd*, *od*, or *xxd* utilities instead of *hexdump* . Also provided for the assignment is **1521 show-galaxy** which prints the contents of a galaxy in a mroe structured way, e.g.:

```
$ 1521 show-galaxy examples/2_files.galaxy
Field name    Offset        Bytes                            ASCII/Numeric
--------------------------------------------------------------------------
============================== Star   0 ================================
magic         0x00000000    63                               chr c
format        0x00000001    38                               chr 8
mode          0x00000002    2d 72 77 2d 72 2d 2d 72 2d 2d    chr -rw-r--r--
path len      0x0000000c    09 00                            dec 9
pathname      0x0000000e    68 65 6c 6c 6f 2e 74 78 74       chr hello.txt
content len   0x00000017    38 00 00 00 00 00                dec 56
content       0x0000001d    48 [illegible] 50                chr Hello COMP
              0x00000027    31 [illegible] 70                chr 1521.I hop
              0x00000031    65 [illegible] 20                chr e you are
              0x0000003b    65 [illegible] 74                chr enjoying t
              0x00000045    68 [illegible] 6e                chr his assign
              0x0000004f    6d [illegible]                   chr ment..
hash          0x00000055    2d                               dec 45
============================== [illegible] =====================
magic         0x00000056    63                               chr c
format        0x00000057    38                               chr 8
mode          0x00000058    2d 72 77 2d 72 2d 2d 72 2d 2d    chr -rw-r--r--
path len      0x00000062    10 00                            dec 16
pathname      0x00000064    6c 61 73 74 5f 67 6f 6f 64 62    chr last_goodb
              0x0000006e    79 65 2e 74 78 74                chr ye.txt
content len   0x00000074    a6 00 00 00 00 00                dec 166
content       0x0000007a    54 68 69 73 20 69 73 20 6f 75    chr This is ou
              0x00000084    72 20 6c 61 73 74 20 67 6f 6f    chr r last goo
              0x0000008e    64 62 79 65 0a 49 20 68 61 74    chr dbye.I hat
              0x00000098    65 20 74 6f 20 66 65 65 6c 20    chr e to feel
              0x000000a2    74 68 65 20 6c 6f 76 65 20 62    chr the love b
              0x000000ac    65 74 77 65 65 6e 20 75 73 20    chr etween us
              0x000000b6    64 69 65 0a 42 75 74 20 69 74    chr die.But it
              0x000000c0    27 73 20 6f 76 65 72 0a 4a 75    chr 's over.Ju
              0x000000ca    73 74 20 68 65 61 72 20 74 68    chr st hear th
              0x000000d4    69 73 20 61 6e 64 20 74 68 65    chr is and the
              0x000000de    6e 20 49 27 6c 6c 20 67 6f 0a    chr n I'll go.
              0x000000e8    59 6f 75 20 67 61 76 65 20 6d    chr You gave m
              0x000000f2    65 20 6d 6f 72 65 20 74 6f 20    chr e more to
              0x000000fc    6c 69 76 65 20 66 6f 72 0a 4d    chr live for.M
              0x00000106    6f 72 65 20 74 68 61 6e 20 79    chr ore than y
              0x00000110    6f 75 27 6c 6c 20 65 76 65 72    chr ou'll ever
              0x0000011a    20 6b 6e 6f 77 0a                chr  know.
hash          0x00000120    60                               dec 96
```

6-bit format (Subset 3 only)

star 6-bit format defines a subset of 64 8-bit values (bytes) to have a six-bit encoding; those six bits are then stored packed.

The remaining 192 8-bit values can not be encoded in 6-bit format.

The functions *star_to_6_bit* and *star_from_6_bit* in *space_6_bit.c* to convert 8-bit values to and from 6-bit format.

You can find the mapping by reading the code in *space_6_bit.c*.

The star hash (Subsets 1, 2, 3)

Each star ends with a *hash* (sometimes referred to as a *digest*) which calculated from the other values of the star. This allows us to detect if any bytes of the galaxy have changed, for example by disk or network errors.

The `star_hash()` function makes one step of computation of the hash of a sequence of bytes:

```c
uint8_t star_hash(uint8_t current_hash_value, uint8_t byte_value) {
    return ((current_hash_value * 33) & 0xff) ^ byte_value;
}
```

Given the hash value of the sequence up to this byte, and the value of this byte it calculates the new hash value.

If we create a galaxy of a single one-byte file, like this:

```
$ echo >a
$ 1521 space -c a.galaxy a
```

We can then inspect the galaxy, and see its hash is `0x15` .

```
$ hexdump -Cv a.galaxy
00000000 63 38 2d 72 77 2d 72 2d 2d 72 2d 2d 01 00 61 01 |c8-rw-r--r--..a.|
00000010 00 00 00 00 00 0a 15 |.......|
00000017
```

Here's the sequence of calls that calculated that value:

```
star_hash(0x00, 0x63) = 0x63
star_hash(0x63, 0x38) = 0xfb
star_hash(0xfb, 0x2d) = 0x76
star_hash(0x76, 0x72) = 0x44
star_hash(0x44, 0x77) = 0xb3
star_hash(0xb3, 0x2d) = 0x3e
star_hash(0x3e, 0x72) = 0x8c
star_hash(0x8c, 0x2d) = 0x21
star_hash(0x21, 0x2d) = 0x6c
star_hash(0x6c, 0x72) = 0x9e
star_hash(0x9e, 0x2d) = 0x73
star_hash(0x73, 0x2d) = 0xfe
star_hash(0xfe, 0x01) = 0xbf
star_hash(0xbf, 0x00) = 0x9f
star_hash(0x9f, 0x61) = 0x1e
star_hash(0x1e, 0x01) = 0xdf
star_hash(0xdf, 0x00) = 0xbf
star_hash(0xbf, 0x00) = 0x9f
star_hash(0x9f, 0x00) = 0x7f
star_hash(0x7f, 0x00) = 0x5f
star_hash(0x5f, 0x00) = 0x3f
star_hash(0x3f, 0x0a) = 0x15
```

Assumptions and Clarifications

Like all good programmers, you should make as few assumptions as possible. If in doubt, match the output of the reference implementation.

- Your submitted code must be a single C program only. You may not submit code in other languages.
- You can call functions from the C standard library available by default on CSE Linux systems: including, e.g., `stdio.h`, `stdlib.h`, `string.h`, `math.h`, `assert.h`.
- We will compile your code with `dcc` when marking. Run-time errors from illegal or invalid C will cause your code to fail automarking (and will likely result in you losing marks).
- Your program must not require extra compile options. It must compile successfully with:

  ```
  $ dcc *.c -o space
  ```

- You may not use functions from other libraries. In other words, you cannot use the dcc `-l` flag.
- If your program prints debugging output, it will fail automarking tests. Make sure you disable any debugging output before submission.
- You may not create or use temporary files.
- You may not create subprocesses: you may not use posix_spawn, posix_spawnp, system, popen, fork, vfork, clone, or any of the `exec*` family of functions, like execve.
- You may assume that the length of a galaxy is less than the maximum value supported by a `long`.
- *space* only has to handle ordinary files and directories.

  space does not have to handle symbolic links, devices or other special files.

  space will not be given directories containing symbolic links, devices or other special files.

  space does not have to handle hard links.
- If completing a `space` command would produce multiple errors, you may produce any of the errors and stop.

  In this case you do not have to produce the particular error that the reference implementation does.
- If a star path name contains a directory then a star for the directory will appear in the galaxy beforehand.

  For example, if there is a star for the path name `a/b/file.txt` then there will be preceding stars for the directories `a` and `a/b`,

  You may also assume the star for the directory specifies the directory is writable.
- When adding an entire directory (subset 3) to a galaxy you may add the directory contents in any order to the galaxy, after the directory star.

  You do not have to match the order the reference implementation uses.
- When a `space` command specifies adding files with a common sub-directory. You may add a star for the sub-directory multiple times. For example, given this command:

```
$ ./space -c a.galaxy b/file1 b/file2
```

You may add two (duplicate) stars for `b` .

- You can assume the path name of a galaxy being created with `-c` , will not also be added to the galaxy, and will not be in a directory being added to the galaxy.
- It is not necessary to check the hashes or magic numbers of stars in subset 0. Subset 0 tests will only use valid stars.
- The reference implementation checks the magic number (first byte), format and hash when listing (`-l` and `-L`) and extracting (`-x`) galaxies and stops with an error message if they are are invalid, for example:

```
$ ./space -l examples/text[illegible]
error: incorrect star hash[illegible]
$ ./space -L examples/text[illegible]
error: incorrect first sta[illegible] 0x63
```

  This is very desirable behaviou[illegible]his in your code. However it will not be tested with `-l` , `-L` and `-x` command line options to avoid [illegible]

  Your code will only be tested w[illegible]xys with invalid hashes, magic numbers and formats

- It is not necessary to check the hashes or magic numbers in an existing galaxy when appending to it (*-a*).

If you need clarification on what you can and cannot use or do for this assignment, ask in the class forum.

You are required to submit intermediate versions of your assignment. See below for details.

Assessment

Testing

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest space [optionally: any extra .c or .h files]
```

You can also run autotests for a specific subset. For example, to run all tests from subset 0:

```
$ 1521 autotest space subset0 [optionally: any extra .c or .h files]
```

Some tests are more complex than others. If you are failing more than one test, you are encouraged to focus on solving the first of those failing tests. To do so, you can run a specific test by giving its name to the `autotest` command:

```
$ 1521 autotest space test1_subset0 [optionally: any extra .c or .h files]
```

`1521 autotest` will not test everything.
Always do your own testing.

Automarking will be run by the lecturer after the submission deadline, using a superset of tests to those `autotest` runs for you.

> WARNING:
>
> Whilst we can detect errors have occurred, it is often substantially harder to automatically explain what that error was. As you continue into later subsets. the errors from `1521 autotest` will become less and less clear or useful. You will need to do your own debugging and analysis.

Submission

When you are finished working on the assignment, you must submit your work by running `give` :

```
$ give cs1521 ass2_space space.c [optionally: any extra .c or .h files]
```

You must run `give` before **Week 10 Wednesday 18:00:00** to obtain the marks for this assignment. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

You can run `give` multiple times.
Only your last submission will be marked.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

You *cannot* obtain marks by emailing your code to tutors or lecturers.

You can check your latest submission on CSE servers with:

```
$ 1521 classrun check ass2_space
```

You can check the files you have submitted here.

Manual marking will be done by your tutor, who will mark for style and readability, as described in the **Assessment** section below. After your tutor has assessed your work, you can view your results here; The resulting mark will also be available via give's web interface.

Due Date

This assignment is due **Week 10 Wednesday 18:00:00** (2024-04-17 18:00:00).

The UNSW standard late penalty for assessment is 5% per day for 5 days - this is implemented hourly for this assignment.

Your assignment mark will be reduce[illegible]r part thereof) late past the submission deadline.

For example, if an assignment worth [illegible]n hour late, it would be awarded 59.8%, whereas if it was submitted past 10 hours late, it would be awarded 57.8[illegible]

Beware - submissions 5 or more day[illegible]ks. This again is the UNSW standard assessment policy.

Assessment Scheme

This assignment will contribute **15** marks to your final COMP1521 mark.

80% of the marks for assignment 2 will come from the performance of your code on a large series of tests.

20% of the marks for assignment 2 will come from hand marking. These marks will be awarded on the basis of clarity, commenting, elegance and style. In other words, you will be assessed on how easy it is for a human to read and understand your program.

An indicative assessment scheme for performance follows. The lecturer may vary the assessment scheme after inspecting the assignment submissions, but it is likely to be broadly similar to the following:

100% for performance	implements all behaviour perfectly, following the spec exactly.
90% for performance	completely working subsets[0-3].
80% for performance	completely working subsets[0-2].
65% for performance	completely working subsets[0-1].
50% for performance	completely working subset0.
30-40% for performance	good progress, but not passing subset0 autotests.
0%	knowingly providing your work to anyone and it is subsequently submitted (by anyone).
0 FL for COMP1521	submitting any other person's work; this includes joint work.
academic misconduct	submitting another person's work without their consent; paying another person to do work for you.

An indicative assessment scheme for style follows. The lecturer may vary the assessment scheme after inspecting the assignment submissions, but it is likely to be broadly similar to the following:

100% for style	perfect style
90% for style	great style, almost all style characteristics perfect.
80% for style	good style, one or two style characteristics not well done.
70% for style	good style, a few style characteristics not well done.
60% for style	ok style, an attempt at most style characteristics.
≤ 50% for style	an attempt at style.

An indicative style rubric follows:

- Formatting **(6/20)**:
  - Whitespace (e.g. `1 + 2` instead of `1+2`)
  - Indentation (consistent, tabs or spaces are okay)
  - Line length (below 80 characters unless very exceptional)
  - Line breaks (using vertical whitespace to improve readability)
- Documentation **(8/20)**:
  - Header comment (with name and zID)
  - Function comments (above each function with a description)
  - Descriptive variable names (e.g. `char *home_directory` instead of `char *h`)

- Descriptive function names (e.g. `get_home_directory` instead of `get_hd`)
- Sensible commenting throughout the code (don't comment every single line; leave comments when necessary)

- Elegance **(5/20)**:
  - Does this code avoid redundancy? (e.g. Don't repeat yourself!)
  - Are helper functions used to reduce complexity? (functions should be small and simple where possible)
  - Are constants appropriately created and used? (magic numbers should be avoided)
- Portability **(1/20)**:
  - Would this code be able to compile and behave as expected on other POSIX-compliant machines? (using standard libraries without platform-specific code)
  - Does this code make any dianness of the machine it is running on?

Note that the following penalties app agiarism:

0 for asst2	work to anyone bmitted (by anyone).
0 FL for COMP1521	son's work; this includes joint work.
academic misconduct	submitting another person's work without their consent; paying another person to do work for you.

Intermediate Versions of Work

You are required to submit intermediate versions of your assignment.

Every time you work on the assignment and make some progress you should copy your work to your CSE account and submit it using the `give` command below. It is fine if intermediate versions do not compile or otherwise fail submission tests. Only the final submitted version of your assignment will be marked.

Assignment Conditions

- **Joint work** is **not permitted** on this assignment.

  This is an individual assignment. The work you submit must be entirely your own work: submission of work even partly written by any other person is not permitted.

  Do not request help from anyone other than the teaching staff of COMP1521 — for example, in the course forum, or in help sessions.

  Do not post your assignment code to the course forum. The teaching staff can view code you have recently submitted with give, or recently autotested.

  Assignment submissions are routinely examined both automatically and manually for work written by others.

  Rationale: this assignment is designed to develop the individual skills needed to produce an entire working program. Using code written by, or taken from, other people will stop you learning these skills. Other CSE courses focus on skills needed for working in a team.

- The use of generative tools such as Github Copilot, ChatGPT, Google Bard is **not permitted** on this assignment.

  Rationale: this assignment is designed to develop your understanding of basic concepts. Using synthesis tools will stop you learning these fundamental concepts, which will significantly impact your ability to complete future courses.

- **Sharing, publishing, or distributing** your assignment work is **not permitted**.

  Do not provide or show your assignment work to any other person, other than the teaching staff of COMP1521. For example, do not message your work to friends.

  Do not publish your assignment code via the Internet. For example, do not place your assignment in a public GitHub repository.

  Rationale: by publishing or sharing your work, you are facilitating other students using your work. If other students find your assignment work and submit part or all of it as their own work, you may become involved in an academic integrity investigation.

- **Sharing, publishing, or distributing** your assignment work after the completion of COMP1521 is **not permitted**.

  For example, do not place your assignment in a public GitHub repository after this offering of COMP1521 is over.

  Rationale: COMP1521 may reuse assignment themes covering similar concepts and content. If students in future terms find your assignment work and submit part or all of it as their own work, you may become involved in an academic integrity investigation.

Violation of any of the above conditions may result in an academic integrity investigation, with possible penalties up to and including a mark of 0 in COMP1521, and exclusion from future studies at UNSW. For more information, read the UNSW Student Code, or contact the course account.

COMP1521 24T1: Computer Systems Fundamentals is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries please email the class account at cs1521@cse.unsw.edu.au
CRICOS Provider 00098G