CSC209 Programming Assignment #2

Due Date: Thursday, June 28, 2018 at 11:30PM

cycle.c

nftw - file tree walk

```
#include <ftw.h>

int nftw(const char *dirpath,
         int (*fn) (const char *fpath, const struct stat *sb,
                    int typeflag, struct FTW *ftwbuf),
         int nopenfd, int flags);
```

`nftw()` walks through the directory tree that is located under the directory `dirpath`, and calls `fn()` once for each entry in the tree. By default, directories are handled before the files and subdirectories they contain (preorder traversal).

READ the man pages for `nftw()` in teach.cs for more information.

Design a C program `cycle.c` which takes one command line argument, `directory name`, and determines if there is a cycle in the file system hierarchy that has that directory name as the root. You are to implement this using nftw(). If the command-line argument is missing, the default is the current working directory. Ensure your program do error-checking for common errors, e.g. non-existent directory.

```
cycle <dirname>
```

myfind.c

Read the man pages for the find command. Try it out yourself, as in:

```
$ find $HOME -name cycle.c -print
```

The real version of find has a more complicated syntax. For the purposes of this assignment, we use the term "object" to denote a file or a directory, as appropriate. You will be writing a C program `myfind.c` to implement only the find-an-object-with-a-given-name functionality of find, with just two arguments, as in

```
$ ./myfind $HOME mymv.c
```

This will search $HOME (your home directory) and all subdirectories of $HOME and print the full pathname of all objects named "`mymv.c`".

You must include a function

```
<type> searchdir (char *dirname, char *findme)
```

that opens a directory, reads it entry by entry, and prints out (the full pathname of) objects that have the individual name "findme"; the function's return type is your choice. Handle cases where an entry that you read is a directory name. Ensure your program do error-checking for common errors, e.g. non-existent directory. Assume filenames will not be any more than 31 characters.

Do not use `nftw()` to do this program. Familiarize yourself with `readdir()` and `opendir()`. Read the man pages for readdir in teach.cs.

```
#include <dirent.h>
struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *dirp, struct dirent *entry, struct dirent
**result);

#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
```

compress_file.c

Of the many techniques for compressing the contents of a file, one of the simplest and fastest is known as run-length encoding. This technique compresses a file by replacing sequences of identical bytes by a pair of bytes: a repetition count followed by a byte to be repeated. For example, suppose that the file to be compressed begins with the following sequence of bytes (shown in hexadecimal):

```
46 6F 6F 20 62 61 72 21 21 21 20 20 20 20 20
```

The compressed file will contain the following bytes:

```
01 46 02 6F 01 20 01 62 01 61 01 72 03 21 05 20
```

Run-length encoding works well if the original file contains many long sequences of identical bytes. In the worst case (a file with no repeated bytes), run-length encoding can actually double the length of the file.

Write a program named `compress_file` that uses run-length encoding to compress a file. To run `compress_file`, we'd use a command of the form

```
compress_file original-file
```

`compress_file` will write the compressed version of *`original-file`* to *`original-file.rle`*.

For example, the command

```
$ ./compress_file foo.txt
```

will cause `compress_file` to write a compressed version of `foo.txt` to a file named `foo.txt.rle`.

Ensure your program do error-checking for common errors, e.g. non-existent file. Remember how you read and write binary files in Lab 4.

uncompress_file.c

Write a program name `uncompressed_file` that reverses the compression performed by the `compress_file` program. The `uncompress_file` command will have the form

```
uncompress_file compressed_file
```

`compressed-file` should have the extension `.rle`. For example, the command

```
% ./uncompress_file foo.txt.rle
```

will cause `uncompress_file` to open the file `foo.txt.rle` and write an uncompressed version of its contents to `foo.txt`. `uncompress_file` should display an error message if its command-line argument doesn't end with the `.rle` extension.

What to submit

Remember to clone the MarkUs repository, though there are NO starter code for this assignment.

- `cycle.c`
- `myfind.c`
- `compress_file.c`
- `uncompress_file.c`
- `Makefile`

`Makefile` should allow for the following:

- `$ make cycle` - compile and create the executable for the cycle program
- `$ make myfind` - compile and create the executable for the myfind program
- `$ make compress_file` - compile and create the executbale for the compress-file program
- `$ make uncompress_file` - compile and create the executable for the uncompress_file program
- `$ make all` - compile and create the executables for all 5 programs.

Sample output

```
$ ./cycle dir1
No cycle found.

$ ./cycle ../f5
Cycle found at /u/joe/csc209/f5/f6/f7

$ ./myfind ../../ mymv.c
/u/joe/csc209/f5/mymv.c
/u/joe/csc209/f5/f6/mymv.c

$ ./compress_file foo.txt

$ ./uncompress_file foo.txt.rle
```