CSE 303, Spring 2005, Assignment 4
Due: Thursday 5 May, 9:00AM

Last updated: April 25

Summary: You will complete a C program that processes a file of "friends" and prints out each person's friends. Your implementation must use the code provided on the course website. There are also other requirements for your implementation. The sample solution is 99 lines.

Behavior Specification:

1. The program takes one argument, the name of a file with the input.
2. The program may assume the input file is correctly formatted, as described here.
   - The input file will have only non-blank text lines that end with a newline (`'\n'`) character, i.e., UNIX-style line-endings.
   - Each line will have a sequence of one or more full-names. A full-name is a string with exactly one space character (`' '`) in the middle. An example is "`Dan Grossman`" (without the quotes). Before the first full-name and after the last full-name there can be any number of space characters. Between full-names there can be any non-zero number of space characters.
3. The input file describes "friendships":
   - A line with one full-name means nothing.
   - A line with more than one full-name means the first person is friends with all the other people ***and*** all the other people are friends with the first person.
   - There can be redundant information (repeated friendships) on one line and/or different lines.
4. The output file describes "friendships" in a different format than the input:
   - For every name `x y` in the input file, there is one line in the output file, beginning with `x y:` and followed by a space-separated list of that person's friends.
   - The lists of friends on each line must be accurate and *not* have any names repeated.
   - Any order of names and their friends is acceptable.
5. Example: If the input is

```
Dan Grossman  Wayne Gretzky Mario Limieux
George Bush John Kerry
John Kerry Dan Grossman Dan Grossman
a b c d e f
```

   then one correct output is

```
e f: a b
c d: a b
a b: e f c d
John Kerry: Dan Grossman George Bush
George Bush: John Kerry
Mario Limieux: Dan Grossman
Wayne Gretzky: Dan Grossman
Dan Grossman: John Kerry Mario Limieux Wayne Gretzky
```

Implementation Specification:

1. Compile with `gcc -Wall -o hw4 hw4.c share_string.c` where the C files and `share_string.h` are provided to you. (The provided `hw4.c` will *not* compile until you edit it.)
2. Change *only* `hw4.c` and do *not* change any of the code already in this file. That is, only add code and do not change the function bodies already provided.
3. You must use the library provided by `share_string.c` for creating "names" that you will store in a data structure that keeps track of friends. You can compare strings returned from `find_word` with pointer-equality (`==`); do not compare such strings' contents.
4. Do not use unnecessary space. In particular, for each "person" you should only ever make one `struct` describing that person. (The next section explains how to do this.)

Algorithm Overview (i.e., "how to do it"):

1. Define three `structs`. Because you cannot change `print_friends`, you have little choice in the definitions. A "people list" is a linked-list of "persons". A "person" is a "name" and a "friend list". A "name" is just a string returned by `find_word`. A "friend list" is a linked list of "persons".
2. As you process the input, maintain a `people_lst_t` that holds all the people and friends you have seen so far, without having any repeats. These functions will help:
   - `add_person` takes a *pointer to* your "people list" and a name, and returns the "person" for that name. If the pointed to "people list" already has a person with the name, it returns that person, else it creates a longer "people list" (use `malloc`) with a new "person" and returns the new "person". (Sample solution: 11 lines) Note: The function is somewhat misnamed since it also "finds" a person already in the "people list".
   - `add_friend` takes a "people list", a "person", and a "name" and returns nothing. If the person is already friends with the name, it does nothing. If necessary, it adds a new person (see above) for the name. It then adds the person with the name to the argument person's friend list. (Sample solution: 10 lines.)
   - `get_name_begin` takes and returns a `char*`. If the argument string has only (zero or more) space characters before a `'\n'`, it returns `NULL`. Else it returns a pointer to the first character in the string that is not `' '`. That is, if the argument starts with n spaces, then the function returns its argument plus n. (Sample solution: 7 lines.)
   - `get_name_end` takes and returns a `char*`. It assumes a full-name is at the beginning of the string it is passed. It returns a pointer to the position in the string *just after* the next full-name. That is, there is memory between where the argument points and the result points (not including where the result points). This memory contains some number of non-space characters, then exactly one `' '` and then as many non-space (and non-newline) characters as possible.
3. Have `get_friends` create a word-cache (using the code in `share_strings`) and space for the "people list" result (initially `NULL`). Use `getline` (see homework 3) to read each line of the input file. For each line, get the first name (using `get_name_begin`, `get_name_end`, and `find_word`) and (potentially) update the "people list" with `add_person`. Then for each additional name on the line, get the name (similar to how you got the first name) and call `add_friend` *twice*. Note that to find the beginning of the next name, you should start where the previous name ended. (Sample solution: 26 lines, including code to free the word-cache and the buffer used to hold input lines. It's okay if you don't do this, but you will have a (small and short-lived) space leak.)

Advice: Understand the basic idea of the algorithm before you write too much code. Pictures may be a big help. For `add_person` and `add_friend`, remember that `find_word` allows you to compare names with pointer equality. This assignment is hard; start early and debug as you go.

Extra Credit: Write a version of the program `hw4_ec.c` that also computes and outputs the data's "degree of separation". The degree of separation is the least n such that you can get between any two people with at most $n-1$ intermediate friends (as in "a is friends with b and b is friends with c and ..."). If there is one person, the degree of separation is 0. If everyone is friends with everyone, the degree of separation is 1. If there are two (or more) people that cannot be connected via any number of friends, the degree is "infinity".

The key to computing the degree efficiently is boolean matrix multiplication! Assign each person a number and build a matrix A where $A[i, j]$ is 1 if $i == j$ or i and j are friends, and 0 otherwise. Multiplication is "boolean and" – x times y is 1 if x and y are 1 and 0 otherwise. Addition is "boolean or" – x plus y is 1 if x or y is 1 and 0 otherwise.

Build a second matrix B initialized to A. If after setting B equal to AB (i.e., matrix multiplication of A and B) n times B has no 1 entries, then the degree of separation is $n+1$ (assuming after $n-1$ iterations there were 1 entries). If at some point B and AB are the same and there are 1 entries, the degree of separation is infinite.

Print out one additional line of the form "Degree of separation: n" where n is the degree of separation.

Warning: The sample solution did not do the extra credit. The algorithm above may have unintentional errors.

Assessment: Your solutions should be:

- Correct C programs that compile without warnings using `gcc -Wall`.
- In good style, including indentation and line breaks
- Of reasonable size

Turn-in Instructions: Follow the link on the course website and follow the instructions there.