Assignment #1

Due: January 23, 2017 11:59pm

Overview

This assignment serves two purposes. Its first purpose is as an introduction to programming in C (a warm-up of sorts). Its second purpose is as an introduction to working in the Unix environment.

You will implement a (feature-reduced) version of the Unix `tr` program. This program "translates" characters based on a user-specified mapping. This can be used, for instance, to translate every '-' into a '_' (the full-featured program allows for more advanced translations).

mytr

Write a program called `mytr` that translates or deletes characters from its input. Your program should work as a filter; it should take its input from the standard input (stdin) and write its output to the standard output (stdout). All error messages should be written to standard error (stderr).

This program will support a subset of the features supported by `tr(1)`, so you can test your program by comparing its output to that of `tr`. The following example demonstrates how one might compare the output of these two programs. `diff(1)` is a program that reports differences between two files; it gives no output if the file contents are identical.

```
% gcc -o mytr -Wall -ansi -pedantic mytr.c
% mytr "abc" "def" < test.1 > mytr.1.out
% tr "abc" "def" < test.1 > tr.1.out
% diff mytr.1.out tr.1.out
%
```

The example uses I/O redirection. A `<` causes a program to read from the specified file as its standard input. A `>` causes a program to write to the specified file as its standard output. (Systems-level details will be discussed later.)

The compile-line above uses many switches for `gcc`. The first, `-o mytr`, tells the compiler to name the output `mytr` rather than `a.out`. The second switch, `-Wall`, turns on verbose warnings. This catches many common programming errors. All of the code that you write in this class should compile cleanly with `-Wall` turned on. The final two switches can be used for additional checks. You must, of course, provide a `Makefile` for your submission.

Features

The `mytr` program takes a number of command-line arguments. At a minimum the program must be given a "set" of characters. This set is simply a quoted string of characters specified on the command-line. The number of sets required depends on the operation desired.

Translation

If given two sets (`SET1` and `SET2`), then `mytr` copies its standard input to standard output while replacing each occurrence of a character in `SET1` with the corresponding (by position) character in `SET2`. For example, `mytr "abc" "def"` with input `cxbyaz` gives output `fxeydz` (replacing each 'a' with 'd', each 'b' with 'e', and each 'c' with 'f').

If too few or too many sets are given, then the execution is in error. If `SET1` is longer than `SET2`, then `SET2` is extended by repeating its last character as necessary (i.e., if `SET1` is "abcdef" and `SET2` is "xyz", then 'c', 'd', 'e', and 'f' are all replaced with 'z'). Excess characters in `SET2` are ignored.

If the first set contains repeats of the same character, then that character is translated based on its last occurrence in the set (i.e., with sets "abcad" and "12345", 'a' characters will translate to "4").

Deletion

If `mytr` is given the `-d` switch and a single set, then it copies its standard input to standard output while deleting those characters that appear in the set. Only a single set may be specified with the `-d` switch, otherwise an error is reported.

Sets

In addition to the typical single character entries, your program must also support the following escaped characters in the argument sets.

`\\` (backslash)

`\n` (new line)

`\t` (horizontal tab)

A `\` followed by any other character maps to that character (this is particularly useful to escape a `-` at the beginning of the first set to prevent that set being treated as a command-line switch).

Each of these escaped characters is written as a two-character sequence, but represents only a single character in the set.

If you have any doubt about what your program is doing relative to what it should do, check its output against the output of the `tr` program. They should do the same thing for those features that they share in common.

Tip

There is a classic tradeoff in computing between space and time. More specifically, one can often use additional space (memory) to decrease execution time, or vice versa. This program provides an example of such a tradeoff.

In particular, you might find your solution to be simpler if you treat characters as their integer values (in the range 0-255, for the typical 8-bit character) and use those values to index into an array that represents the mapping as defined by the argument sets.

Testing

Be sure to test each feature with various inputs. Test with large files, small files, and, importantly, an empty file. Test with a file that does not end with a newline. Though these different sorts of files should not make any difference for this program, such testing might expose issues with more complicated formats in future programs.

Test with too few, too many, and invalid command-line arguments. Test with invalid switches and with multiple switches.

Strive for a robust program that fails gracefully (reports issues) rather than one that crashes on bad input (because every user provides bad input to a program now and then). In short, we are now beyond the point of expecting only well-formed input.

Submit

- All source code.
- A `Makefile` that can be used to build your program (with the resulting executable named `mytr`).

Grading

Feature	Percentage
Translation	40
Deletion	30
Escaped Characters	20
Error Handling	10

High-Level Decomposition Example

- Parse command-line arguments
  Test: unit tests with specified arrays of strings
  or, system tests by echoing user input
- Verify command-line arguments
  Test: unit tests with specified arrays of strings
  or, system tests by echoing user input
- Simplify sets by reducing escape sequences to single characters.
  Test: unit tests on various string arguments
- Read/write input character-by-character (`getchar`/`putchar`)
  - Translate according to mapping defined by sets, if translate mode.
  - Remove characters listed in set, if delete mode.

  Test: system tests with correct and erroneous input