COMP 2510
Assignment 2

1 Introduction

The purpose of this assignment is to write a program to translate characters read from standard input, writing the result to standard output. The program is a much simplified version of the `tr` program available on Unix systems.

Note that as the program reads from standard input & writes to standard output (except for error messages), there is no need to explicitly open any file.

2 Translating

In this write-up, we'll call our program `tr`. It is based on the well-known loop:

```
int  c;

while ((c = getchar()) != EOF)
  putchar(c);
```

However, instead of simply echoing the character read, it may echo another character. For example, the following translates all occurrences of 'a' (lowercase) to 'A' (uppercase), leaving other characters unchanged:

```
int  c;

while ((c = getchar()) != EOF)
  if (c == 'a')
    putchar('A');
  else
    putchar(c);
```

The `tr` program can actually translate more than one specific character; it can translate one set of characters to another set. It is invoked as follows:

```
tr set1 set2
```

The 2 strings set1 & set2 specify 2 sequences of characters & `tr` basically translates each character in the sequence for set1 to the corresponding character in the sequence for set2. Characters not in the sequence for set1 are unchanged. For example, the following invocation translates all occurrences of 'a' to 'x', 'e' to 'y' & 'i' to 'z':

```
tr aei xyz
```

However, the sequence specified by set1 may have a different number of characters than the string set1 itself because we allow escape characters & ranges. The same applies to set2. As an example, consider the invocation:

```
tr 'a\nbc' d-g
```

The first thing to note is that the quotes are not part of the first set. We assume we are using a system where the backslash character has special meaning to the command shell (e.g., the bash shell in Linux). On such a system, it is necessary to enclose a set that contains backslashes in quotes. The shell automatically removes the quotes before passing the argument to our C program. What this means is that, in such a shell, the string passed to our C program actually contains the 5 characters 'a', '\','n', 'b' & 'c' & our C program needs to convert it to the sequence 'a', '\n' (newline), 'b' & 'c'.

Because of the escape character (which represents newline), the first set specifies a sequence consisting of only 4 characters. The second set specifies the range of characters between 'd' & 'g' inclusive, also a sequence of 4 characters (in the ASCII encoding). In both instances, the number of characters in the sequence specified by the set is different from the number of characters in the set itself. The invocation translates 'a' to 'd', the newline character to 'e', 'b' to 'f' & 'c' to 'g'. (See section 3 for details on escape characters & ranges.)

Note that only the backslash & the hyphen have special meanings. Without them, each character in a set represents the same character in the sequence.

In case there are more characters in the sequence for set2 than that for set1, the extra characters at the end of the sequence for set2 are ignored (i.e., the sequence for set2 is effectively truncated). Similarly, if the sequence for set1 has more characters than that for set2, then the sequence for set1 is truncated to the length of that for set2.

For example, the following pairs of commands are equivalent:

- `tr abc vwxyz` and `tr abc vwx` (set2 truncated)
- `tr abcdef vwx` and `tr abc vwx` (set1 truncated)

When a translation for a character is specified more than once, only the final specification is in effect. For example, the following pairs of commands are equivalent:

- `tr abaca vwxyz` and `tr bca wyz` (translation of `a` specified thrice; last `a` $\rightarrow$ `z` used)
- `tr abacad wxywxy` and `tr bcad xwxy`

Note that the translation is "performed once". This is best explained by an example. Consider the command

```
tr ab bc
```

This translates `a` to `b` and `b` to `c`. It does *not* translate `a` to `c` via `b`.

3 Escape Characters & Ranges

The program uses command-line arguments to specify sets of characters. Most characters in those arguments represent themselves. However, the following escape characters are also allowed:

\\ \a \b \f \n \r \t \v \' \"

Each represents one character & has the same meaning as in C, e.g. \\ is backslash , \r is the return character, \' is the single quote & \" is the double quote. Any other character immediately following a starting backslash is regarded as invalid.

Single & double quotes need to be escaped because with many shells (including the bash shell), we need to enclose a set that contains escape characters in single or double quotes. (We have already seen an example of this in the previous section.) This is because some characters have special meanings to the shell. Note that your C program needs to process the escape characters as each of them is generally passed as 2 characters (rather than 1 character) to your C program.

The only other character (besides backslash) that has special meaning when used to specify a set is the hyphen (`-`): it can be used to specify a range of characters. Typically, if the hyphen occurs between 2 characters (which may be escape characters), then it specifies the range of characters between & including the 2 characters. (But see exception below.) Note that this range depends on the character encoding used (which is usually ASCII). It is an error if the first character has a bigger code than the second. In other locations, the hyphen has no special meaning. The hyphen can also lose its special meaning even if it is between 2 characters. This happens if the character that precedes that hyphen is itself the end-point of a range specified by another hyphen before it. For example, the set specified by `a-d-g` is `a`, `b`, `c`, `d`, `-` and `g` — the second hyphen has no special meaning because `d` is already the end-point of a range.

Here are more examples (assuming ASCII encoding):

- `a-z` : the range of characters from `a` to `z` inclusive
- `a-z-` : `a` to `z`, then `-` (second hyphen has no special meaning)
- `-a-z` : `-`, then `a` to `z` (first hyphen has no special meaning)
- `a-d-g-j` : `a`, `b`, `c`, `d`, `-`, `g`, `h`, `i`, `j`
- `z-a` : this is an error since `z` has a bigger ASCII code than `a`
- `a-a` : the range from `a` to `a`; hence this is just an `a`
- `---` : the range from `-` to `-`; hence just a hyphen
- `---a` : `-` & `a`

Note: In your implementation, it may be necessary to write a function to expand the ranges specified by command-line arguments. (For example, the string `"a-dgk-n"` is expanded to `"abcdgklmn"`.) Since it is difficult to know in advance how many characters a string will expand to, use a buffer of `MAXSIZE` characters, where `MAXSIZE` is a macro with a value of 256, to store the expanded string. If this limit is exceeded, print an error message & exit the program.

4 Further Details

Note that, not counting I/O redirection, the only valid way of invoking `tr` is:

```
tr set1 set2
```

The program should print a "usage" message (to standard error) before exiting if it is not invoked correctly.

Note that `tr` is typically invoked using I/O redirection. For example,

```
    tr a-z A-Z < infile > outfile
```

In the above, the shell passes only the first 3 words to the C program; the rest are not regarded as command-line arguments.

You are not allowed to use external variables (variables defined outside of functions) in your program. You may lose substantial marks if you do.

Sample input & output files will be provided. Make sure that the output of your program matches exactly those in the provided output files.

5 Submission & Grading

This assignment is due at 11pm, Friday, March 4, 2016. Submit a zip file to "ShareIn" in the directory:

```
\COMP\2510\a2\set<X>\
```

where `<X>` is your set. Your zip file should be named `<name_id>.zip`, where `<name_id>` is your name & student ID separated by an underscore (for example, `SimpsonHomer_a12345678.zip`) Do not use spaces to separate your last & first names. Your zip file must unzip directly to your source file (without creating any directories). We'll basically compile your file using

```
gcc -ansi -W -Wall -pedantic *.c
```

after unzipping.

Do not submit rar files. They will not be accepted.

If you need to submit more than one version, name the zip file of each later version with a version number after your name, e.g., `SimpsonHomer_a12345678_v2.zip`. If more than one version is submitted, we'll only mark the version with the highest version number. (In this case, there must be exactly one zip file with the highest version number; if there are two or more of them, you may fail to get credit for the assignment.)

Your program must compile without warnings or errors under `gcc` with the following options:

```
    -ansi -W -Wall -pedantic
```

Furthermore, you must implement any function that you use & that is not in the standard ANSI C library. (In practice, this usually means that you should not use functions that we have not mentioned in class.) If your program does not meet these requirements, you may receive a score of 0 for the assignment. Otherwise the grade breakdown for this assignment is *approximately* as follows:

Design & code clarity	10%
Command-line validation	10%
Basic translation (includes truncation of sequences)	35%
Handling escape characters	20%
Handling ranges	25%