Project 5, Program Design

1. (60 points) A range is a sequence of characters in the form [character-character], such as [k-n] or [d-g]. Write a program to check if a string contains a range and if so, replace them with the characters contained in the range. For example, the program should replace [k-n] with **klmn**.

Assume the input contains either one range or none. Also assume that the range will be in the valid form [character-character] if the input contains one.

Example input/output #1:

```
Enter the input: [d-f]ine
Output: define
```

Example input/output #2:

```
Enter the input: red alert
Output: There is no range in the input
```

Your program should include the following function:

```
int replace(char *s1, char *s2);
```

The function expects `s1` to point to a string containing the input as a string and `s2` to point to a string containing the output as a string. The function returns 1 if there is a range in the input and 0 otherwise.

1) Name your program `range.c`.

2) Assume the alphabetic letters in the input string are lowercase.

3) The `replace` function should use pointer arithmetic (instead of array subscripting) to visit string s1 and s2.

4) In the main function, declare the string variables for input and output. Read a line of text, and then call the `replace` function. The main function should display the output.

5) To read a line of text, use the `read_line` function (the pointer version) in the lecture notes.

2. (40 points) Write a program that accepts as command line arguments the sign of a math operation (+, -, x, or /) and two integer numbers and displays the result of the operation (all integer operations). For example, if the arguments are `+`, `5`, `-3`, the program should display `2`. You may find strcmp function useful.

Note: x for the multiplication is letter x, not *. (* has special meaning to Unix shell.)

Sample run:

```
./a.out - 5 2
```

```
output: 3
```

1) Name your program `command_math.c`.
2) Use `atoi` function in `<stdlib.h>` to convert a string to integer form.

Before you submit

1. Compile both programs with –Wall. –Wall shows the warnings by the compiler. Be sure it compiles on ***student cluster*** with no errors and no warnings.

   gcc –Wall range.c
   gcc –Wall command_math.c

2. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

   chmod 600 range.c
   chmod 600 command_math.c

3. Test your programs with the shell scripts on Unix:

   chmod +x try_range
   ./try_range

   chmod +x try_command_math
   ./try_command_math

4. Submit *range.c* and *command_math.c* on Canvas.

Grading
Total points: 100 (60 points problem #1, 40 points problem #2)

1. A program that does not compile will result in a zero.
2. Runtime error and compilation warning 5%
3. Commenting and style 15%
4. Functionality 80%
   -Functions implemented as required

Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your **name**.
2. In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written only *needed* in order for a reader to understand what is happening.

3. Information to include in the comment for a function: name of the function, purpose of the function, meaning of each parameter, description of return value (if any), description of side effects (if any, such as modifying external variables)
4. Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
5. Use consistent indentation to emphasize block structure.
6. Full line comments inside function bodies should conform to the indentation of the code where they appear.
7. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: **`#define PI 3.141592`**
8. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
9. Use underscores to make compound names easier to read: **`tot_vol`** or **`total_volumn`** is clearer than totalvolumn.