Program Design Project 5

Problem 1 (50 points)

Library function strcmp function compare two strings lexicographically but uppercase letter and lowercase letter are not considered equivalent. In this program, you will write a program that implements a string comparison function that compares two strings of the same length and the letters' case does not matter. For example, string "abcd" is considered equivalent to string "aBcD".

If the first string is less than the second one, print "-1". If the second string is less than the first one, print "1". If the strings are equal, print "0". Note that the letters' case is not taken into consideration when the strings are compared.

Example input/output #1:

```
Enter first string: AbC
Enter second string: abc
Output: 0
```

Example input/output #2:

```
Enter first string: dog
Enter second string: Log
Output: -1
```

Example input/output #3:

```
Enter s1: pile
Enter s2: pale
Output: 1
```

1) Name your program `project5_compare.c`.
2) Assume the input have the same length. And the input length for the two strings is no more than 1000 characters.
3) Your program should include the following function:

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

   The function expects s1 and s2 to be strings. It returns the result of the comparison, -1, 0 or, 1.

   This function should use pointer arithmetic– not subscripting – to visit array elements. In other words, eliminate the loop index variables and all use of the [] operator in the function.

4) String library functions are NOT allowed for this program. If you use a string library function, you will NOT receive the credit for the count function part of the program.

5) Character handling functions such as tolower are allowed.

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

Problem 2 (50 points)

Modify project 2, problem 2 (validate words) so the arithmetic expression is a command line argument.

That is, the arithmetic operation and two single digit integers are entered as command line arguments. The program allows add, subtract, and multiply two single digit integers by entering either +, -, * between the integers:

Example input/output:

```
./a.out 7 - 3

Output:  The subtraction is 4
```

1) Name the program `project5_arithmetic.c`
2) The input format is exactly in the format of: integer1 operator interger2
3) There might be any number (1 or more) of white space(s) before the first integer, between the integers and the operator in the input, or after the second integer.
4) If the operator entered is not one of the acceptable operators, display an error message as the result.
5) The program should check if the correct number of arguments are entered on the command line. If an incorrect number of arguments are entered, the program should display a message.

Other requirements and submission:

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 project5_compare.c

   gcc –Wall project5_arithmetic.c

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

   chmod +x try_project5_compare

   ./try_project5_arithmetic

```
chmod +x try_project5_compare

./try_project5_arithmetic
```

3. Download the programs (.c files) from student cluster and submit it on Canvas>Assignments.

Grading

Total points: 100 (50 points each problem)

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% (**Including 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.