Project 5, Program Design

1. (50 points) Suppose you are given a string s1, such as "psg", representing the types of precious metal bars. Each character represents one type of metal bar, for example, 'p' for platinum bar, 's' for silver bar, and 'g' for gold bar.

   Another string s2 is given representing the bars you have, such as "abc". Each character in s2 is a type of metal bar you have, for example, 'a' for aluminum, 'b' for "brass", 'c' for "copper".

   Write a program to calculate how many of the metal bars you have in s2 are in s1. Letters are case sensitive, so 'a' is considered a different type of metal bar from 'A'.

   Example input/output:

```
Enter s1: psg
Enter s2: abc
Output: 0

Enter s1: psg
Enter s2: ppbgc
Output: 3
```

1) Name your program `metal.c`.

2) Your program should include the following function:

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

   The `count` function expects `s1` and s2 to be strings. The function returns the number of characters in s2 are in s1. The `count` function should use pointer arithmetic (instead of array subscripting). In other words, eliminate the loop index variables and all use of the [] operator in the function.

3) Assume s1 and s2 have no more than 1000 characters.

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

2. (50 points) Command-line arguments

Modify project 2, problem 2 (validate words) so the word is a command line argument. A word as a command line argument is valid if all characters of the word are alphabetic letters and one of following conditions holds:

1. All letters are capitals, like “USF”,
2. All letters are not capitals, like “program”.

Example input/output:

```
./a.out fall
Output: valid
./a.out 8littlepigs
Output: invalid
./a.out
Output: Incorrect number of arguments. Usage: ./a.out word
./a.out spring fall
Output: Incorrect number of arguments. Usage: ./a.out word
```

1) Name your program `command_word.c`.
2) The program should include the following function:

```
int validate(char *word);
```

The function expects `word` to point to a string containing the word to be validated. The function returns 1 if word is valid, and 0 otherwise. The function should use pointer arithmetic (instead of array subscripting). In other words, eliminate the loop index variables and all use of the [] operator in the function.

3) The program should also 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.

4) The main function displays the output.

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 metal.c

   gcc –Wall command_word.c

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

   chmod 600 metal.c

   chmod 600 command_word.c

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

   chmod +x try_metal

   ./try_metal

   chmod +x try_command_word

   ./try_command_word

4. Submit *metal.c* and *command_word.c* on Canvas.

Grading

Total points: 100 (50 points problem #1, 50 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.