Project 5, Program Design

1. (60 points) Write a program that reads a line of input and display the characters between the first two `'*'` characters. If no two `'*'` occur, the program should display a message about not finding two * characters.

   For example, if the user enters: `1abc*D2Efg_#!*345Higkl*mn+op*qr` the program should display the following:
   `D2Efg_#!`

1) Name your program `stars.c`.

2) Assume input is no more than 1000 characters.

3) String library functions are NOT allowed in this program.

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

5) Include and call the search function. The `search` function expects `s1` to point to a string containing the input as a string and stores the characters between the first two `'*'` to the string pointed by `s2`. If the input does not contain two `'*'`, s2 should contain an empty string. An empty string is a valid string with no characters except the null character. The function returns 1 if the input contains two `'*'`, and returns 0 otherwise.

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

6) The `search` 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.

7) The main function should display the output.

2. (40 points) Modify problem 2 of project 2 so that input comes in as command line arguments.

Sample run:

```
./a.out W8 4 ME 2 finish 2!

Output:

Consonants: WMfnsh
```

1) Name your program `command_consonants.c.`

2) If no command line arguments was provided, your program should print a usage message "Usage: ./a.out input"

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 stars.c

   gcc –Wall command_consonants.c

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

   chmod 600 stars.c

   chmod 600 command_onsonants.c

3. Test your fraction program with the shell scripts on Unix:

   chmod +x try_stars

   ./try_stars

   chmod +x try_command_consonants

   ./try_command_consonants

4. Submit stars.*c* and *command_consonants.c* on Canvas.

Grading

Total points: 100 (60 points for problem 1 and 40 points for 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% (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.