Project 3, Program Design

1. (50 points) Write a program to create a randomized integer array of randomized length. The program will start with creating a random integer in the range of 8 to 15 for the length of the integer array, then for each array element, create a random integer in the range of 33 to 126 (ASCII values of printable characters except space), then display the corresponding ASCII characters. For example, for the sample output below, the array of random integers generated is: 63 47 86 95 110 60 68 97 101 59 39 48

   Sample output:
   ?/V_nDae;'[

The program should include the following function where a is the array for the randomly generated values and n is the size of the array

```
void generate_values(int a[], int n);
```

1) Name your program ASCII_values.c
2) The main function creates a random number for the array length between 8 to 15, and declares the array, calls the `random_values` function, and then displays the array.
3) In the `generate_values()` function, use `rand()` function to generating random values in the range of 33 to 126.
4) To display an integer as a character, use "%c" format specifier in printf function.
5) To use the `rand()` and time function, you need to include <stdlib.h> and <time.h>.

How to use the rand() function to generate a random number:

1) With the help of `rand ()` function, a number in range of lower to upper can be generated as **num = (rand() % (upper – lower + 1)) + lower**

2) `rand()` function generates the same sequence again and again every time the program runs. Use `srand()` function with time to set seed for `rand()` function so it generates different sequences of random numbers. Include the following statement at the beginning of the main function: `srand(time(NULL));`

2. (50 points) In this program, you will evaluate a randomly generated sequence of characters in problem 1 to determine if it a strong password. In this problem, a strong password contains at least one upper case letter, one lower case letter, one digit, and one special character. A special character is **any printable character that is not a letter or digit**. The password cannot contain > or <, since these characters may cause problems on some systems.

   Example runs:

```
Example output #1:
?/V_nDae;'[

NOT a strong password


Example output #2:
%~I+BI92aBM

A strong password
```

The program should include the following function. The function returns 1 if it is a strong password, otherwise returns 0.

```
int is_strong_password(int a[], int n);
```

1) Name your program `password.c`
2) The main function and display the result.

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 ASCII_values.c
   gcc –Wall password.c
2. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

   chmod 600 ASCII_values.c
   chmod 600 password.c

3. Test your programs multiple times to check its correctness. There are no testing scripts for this project since the arrays are generated randomly.

4. Submit ASCII_values.*c* and password.c on Canvas.

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. 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.
4. Use consistent indentation to emphasize block structure.
5. Full line comments inside function bodies should conform to the indentation of the code where they appear.
6. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: `#define PI 3.141592`
7. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
8. Use underscores to make compound names easier to read: `tot_vol` or `total_volumn` is clearer than totalvolumn.