Project 2, Program Design

Problem 1 - Dice (50 points)

In this program, you will build upon your program for project 1, problem #1 to allow multiple rounds of the game. In average, player, who won most of the rounds, is the winner of the game. In case if two players won the same number of rounds, the result of the game is draw.

```
Example input/output:
Input:
Enter the number of rounds: 3
Output:
Round 1: Player 1’s number is 6, Player 2’s number is 5.
Round 2: Player 1’s number is 4, Player 2’s number is 6.
Round 3: Player 1’s number is 4, Player 2’s number is 4.
It’s a draw!
```

1. Name your program `project2_dice.c`
2. The user enters the number of rounds for the game.
3. Use a for loop for counting rounds of the game.

Problem 2 – Characters (50 points)

In this program, you will calculate the result of an arithmetic operation of **two single digit positive integers**. The program allows add, subtract, and multiply two single digit positive integers by entering either +, -, * between the integers.

Example input/output #1:

```
    Enter input: 7 – 3
    Output:  4
```

Example input/output #2:

```
Enter input: 7    * 3

Output:  21
```

1) Name the program `project2_arithmetic.c`
2) Use **getchar()** function to read in the input. Do not use scanf.
3) There might be any number (1 or more) of white space(s) before the first integer, between the integer and the operator in the input, or after the second integer.
4) The user input ends with the user pressing the enter key (a new line character).
5) The input format is exactly in the format of: integer1 operator interger2
6) Use **switch** statement to process the operator and the math.
7) If the operator entered is not one of the acceptable operators, display an error message as the result.

Other requirements and submission:

1. Compile on ***student cluster (sc.rc.usf.edu)***:

   gcc –Wall project2_dice.c

   gcc –Wall project2_arithmetic.c

2. Test your program with the shell script on the student cluster:

   chmod +x try_project2_arithmetic

   ./try_project2_arithmetic

3. Download the programs from student cluster and submit it on Canvas>Assignments.

Grading:

Total points: 100

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 requirement 80%

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 either underscores or capitalization for compound names for variable: `tot_vol`, `total_volumn`, or `totalVolumn`.