Project 2, Program Design

1. (60 points) Write a program that simulates a game that allows players to win prizes. Assume that there are 12 symbols with internal representation of prizes as shown the table below.

Your program should use random number generators to randomize all outcomes to ensure that every play generates a result that is independent of the previous play.

Symbol	**Internal Storage**
Pizza	0
Shirt	1
Sweater	2
Raspberry	3
Brown Sugar	4
Star	**5**
Butter	6
Apple	7
Blackberry Jelly	8
Strawberry Cake	9
Bacon	10
Play Again	**11**

Requirements:

1) Name your program `prizes.c`.
2) The program allows the player to play by entering 1 and to stop playing by entering 0, then randomly generating a number in the range (0-11) and print out the prize.

   i. If the number is any number except 5 (star) or 11 (play again), the program prints out the prize and terminates.
   ii. If the number is 11 (play again), the program allows the user the play again by entering 1.
   iii. If the number is 5 (star), the player wins the jackpot and the player receives 4000 coins, and the player is allowed to play again by entering 1.

3) Use rand() function to generate a random number. With the help of rand () function, a number in range of lower to upper can be generated as **num = (rand() % (upper – lower + 1)) + lower**
4) 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));

5) To use the rand() and time function, you need to include <stdlib.h> and <time.h>.

Example run #1:

Would you like to play (press 1 to play or 0 to quit)? 1

Output:

Bacon

Example run #2:

Would you like to play (press 1 to play or 0 to quit)? 1

Output:

Play Again

Would you like to play again (press 1 to play or 0 to quit)? 1

Output:

Apple

Example run #3:

Would you like to play (press 1 to play or 0 to quit)? 1

Output:

Play Again

Would you like to play (press 1 to play or 0 to quit)? 0

Output:

Thank you for playing!

Example run #4:

Would you like to play (press 1 to play or 0 to quit)? 1

Output:

Star

Congratulations! You have won 4000 coins

Would you like to play again (press 1 to play or 0 to quit)? 1

Output:

Butter

2. (40 points) Write a program that reads a line of input (When the newline character is entered, the input ends) and prints out the consonants in the input.

1) Name your program consonants.c
2) **getchar()** function is required for reading a character from the keyboard.

Example input/output:

Input: W8 4 ME 2 finish 2!

Output:

Consonants: WMfnsh

Before you submit:

1. Compile with –Wall. –Wall shows the warnings by the compiler. Be sure it compiles on ***student cluster*** with no errors and no warnings.

   gcc –Wall prizes.c

   gcc –Wall consonants.c

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

   chmod 600 prizes.c

   chmod 600 consonants.c

3. The first program uses random numbers, you will test the program manually. Test your program consonants.c with the shell script on Unix:

   chmod +x try_consonants

   ./try_consonants

4. Download *prizes.c* and *cosonants.c* from the student cluster and submit on Canvas.

Grading:

Total points: 100 (60 points problem 1 and 40 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 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`**.