Project 2, Program Design

1. (50 points) Maggie is planning for a trip for her and her friends. The group decided to purchase snacks together and split the cost. There are 10 people in total. The snacks are:

   1. Energy bars - $3.99 /box,
   2. Candy bars - $2.39 /box,
   3. Chips - $4.79 /bag,
   4. Pretzels - $2.99/bag,
   5. Popcorns - $3.50 /bag
   6. Energy drinks - $4.99/half dozen

   Write a program to calculate the cost per person. The program allows the user to select an item, input the number of items, and calculate the cost per person.

   An example input/output:

```
Please select from the list:
1. Energy bars - $3.99 /box,
2. Candy bars - $2.39 /box,
3. Chips -  $4.79 /bag,
4. Pretzels - $2.99/bag,
5. Popcorns - $3.50 /bag
6. Energy drinks - $4.99/half dozen


Enter selection: 1
Enter number of boxes: 3
Enter selection: 3
Enter number of bags: 2
Enter selection: 7
Invalid selection, select from 1 to 6, enter 0 to stop selection
Enter selection: 6
Enter number of drinks as multiples of six (6, 12, 18…): 24
Enter selection: 5
Enter number of bags: 1
Enter selection: 0
Cost per person ($): 4.50
```

1) Name your program *snacks.c*
2) If the user selects a number out of the range (1 through 6), display a message and allows the user to select again. When user enters 0, the program should exit from the loop for selection.
3) Use a **switch** statement to compute the amount due according to the selection(s).
4) Format the output to two decimal digits.

2. In this program, we define a word to be 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”.

    Write a program that prompts the user to enter a word. The program determines if the input is a valid word or invalid.

    1) Name your program word.c.
    2) The user input ends with the user pressing the enter key (a new line character).
    3) Use **getchar()** to read in the input. Character handling functions are allowed.

    Example 1:
    Input: “fall”
    Output: valid

    Example 2:
    Input: “8littlepigs”
    Output: Invalid

    Example 3:
    Input: “CpE”
    Output: Invalid

    Example 4:
    Input: “CSE2021”
    Output: Invalid

    Example 5:
    Input: “CSE”
    Output: Valid

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 snacks.c

   gcc –Wall word.c

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

   chmod 600 snacks.c

   chmod 600 word.c

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

   chmod +x try_snacks

   ./try_snacks

   chmod +x try_word

   ./try_word

4. Download *snacks.c* and *word.c* from the student cluster to your computer and submit on Canvas.

Grading:

Total points: 100 (50 points problem 1 and 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 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`.