Project #2

Due Wednesday by 11:59pm **Points** 100 **Available** until Sep 14 at 2am

Task #1 (50 points)

Task

A number is considered special if it satisfies both criteria below. Write a C program to determine if a number entered by the user is special

- The number representation contains at least one digit 3.
- The number is a multiple of 3.

Requirements

1. Follow the format of the examples below.

Examples (your program must follow this format precisely)

Example #1

```
Enter input: 72
no
```

Example #2

```
Enter input: 639
yes
```

Example #3

```
Enter input: 236
no
```

Submission instructions

1. Develop and test your program on the student cluster
   A. To compile your program, run: **$ gcc -std=c99 -Wall your_program.c**
   B. To execute your program, run: **$ ./a.out**
2. Name your program **project2_number.c**
   A. To rename your program, run: **$ mv your_program.c project2_number.c**
3. Test your program with the shell script on Unix: try_project2_number
   A. Upload the zip file to the same directory as your program.
   B. run: **$ unzip try_project2_number.zip**
   C. move your program to the same folder as try_project2_number file
   D. Run: **$ chmod +x try_project2_number**
   E. Run: **$ ./try_project2_number**
4. Download the program from student cluster and submit it on **Canvas->Gradescope**. Make sure you submit a file with the correct name!
5. You can submit your program as many times as needed before the due date. Gradescope will indicate the test cases with incorrect output, if any exists.

Task #2 (50 points)

Task

Write a program that determines if the input characters are in order. For example, if input is in, it's in order because 'i' is less than 'n'. and for input dog, it's not in order because 'o' is not less than or equal to 'g'. If the input contains non-alphabetical letters, the program should print "invalid input." It's considered in order if two letters are same. Hint: use two variables to keep track of two neighboring characters.

Requirements

1. Follow the format of the examples below.
2. Use **getchar()** function to read in the input. Do not use scanf.
3. **Arrays are not allowed to solve this problem.**
4. The user input ends with the user pressing the enter key (a new line character).
5. The input might contain uppercase or lowercase alphabetic letters. Convert uppercase to lowercase before comparing.
6. Assume the input is one word (no white spaces).
7. Character handling library functions in ctype.h are allowed.

Examples (your program must follow this format precisely)

Example #1

```
Enter input: o7k
invalid input
```

Example #2

```
Enter input: dPS
in order
```

Example #3

```
Enter input: Akd
not in order
```

Submission instructions

1. Develop and test your program on the student cluster
   A. To compile your program, run: **$ gcc -std=c99 -Wall your_program.c**
   B. To execute your program, run: **$ ./a.out**
2. Name your program **project2_inOrder.c**
   A. To rename your program, run: **$ mv your_program.c project2_inOrder.c**
3. Test your program with the shell script on Unix: try_project2_inOrder
   A. Upload the zip file to the same directory as your program.
   B. run: **$ unzip try_project2_inOrder.zip**
   C. move your program to the same folder as try_project2_inOrder file
   D. Run: **$ chmod +x try_project2_inOrder**
   E. Run: **$ ./try_project2_inOrder**
4. Download the program from student cluster and submit it on **Canvas->Gradescope**. Make sure you submit a file with the correct name!
5. You can submit your program as many times as needed before the due date. Gradescope will indicate the test cases with incorrect output, if any exists.

Grading

Task #1

Total points: 50

- A program that does not compile will result in a zero.
- Runtime error and compilation warning 3 points

  1 points off, if a warning is present.

  3 points off, if multiple warnings are present.

- Commenting and style 8 points

  1 point off for not putting name and description at the beginning

  2 to 4 points off if the code didn't have clarifying comments.

  1 to 3 points off (depending on how much indentation is off) if the program is not indented properly.

- Functionality

  8 points off for each incorrect test case

Task #2

Total points: 50

- A program that does not compile will result in a zero.
- Runtime error and compilation warning 3 points

  1 points off, if a warning is present.

  3 points off, if multiple warnings are present.

- Commenting and style 8 points

  1 point off for not putting name and description at the beginning

  2 to 4 points off if the code didn't have clarifying comments.

  1 to 3 points off (depending on how much indentation is off) if the program is not indented properly.

- Functionality

  8 points off for each incorrect test case

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.

- Your program should begin with a comment that briefly summarizes what it does. This comment should also include your name.
- 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.
- 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.
- Use consistent indentation to emphasize block structure.
- Full line comments inside function bodies should conform to the indentation of the code where they appear.
- Macro definitions (**#define**) should be used for defining symbolic names for numeric constants. For example: **#define PI 3.141592**
- Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
- Use underscores to make compound names easier to read: **tot_vol** and **total_volumn** are clearer than **totalvolumn**.