Project 6, Program Design

Write a program to check if an email address read from a file is a USF email, an email that contains `usf.edu`. The output file should contain all the USF email addresses read from the input file. You can assume email addresses in the file are valid email addresses.

Example input/output:

Enter input file: `emails1.txt`

Output: `output file name is USF_emails1.txt`

Technical requirements:

1. Name your program *USF_emails.c*
2. Prompt the user to enter a file name for the input file.
3. The output file name should be the same name but with an added prefix of `USF_`. For example, if the input file name is `emails.txt`, the output file name should be `USF_emails.txt`.
4. Assume the input file name is no more than 100 characters.
5. Read the emails from the file. Email addresses are stored in separate lines.
6. Assume length of email addresses is no more than 2000 characters.
7. The program should include the following function:

   ```
   int is_USF_email(char *email);
   ```

   The function expects `email` to point to a string containing an email address. The function returns 1 if email address contains `usf.edu`, and returns 0 otherwise. String library functions are allowed.

8. In the main function, call `is_USF_email` function and write the email address to the output file if it is an USF email.

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 USF_emails.c

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

   chmod 600 USF_emails.c

3. Test your fraction program with the shell scripts on Unix:

   chmod +x try_USF_emails

   ./try_USF_emails

4. Submit USF_emails.c, emails1.txt, and emails2.txt (for grading purpose) on Canvas.

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 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. Information to include in the comment for a function: name of the function, purpose of the function, meaning of each parameter, description of return value (if any), description of side effects (if any, such as modifying external variables)
4. 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.
5. Use consistent indentation to emphasize block structure.

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