1 Counting and Upcasing/Downcasing Vowels (15 Points)

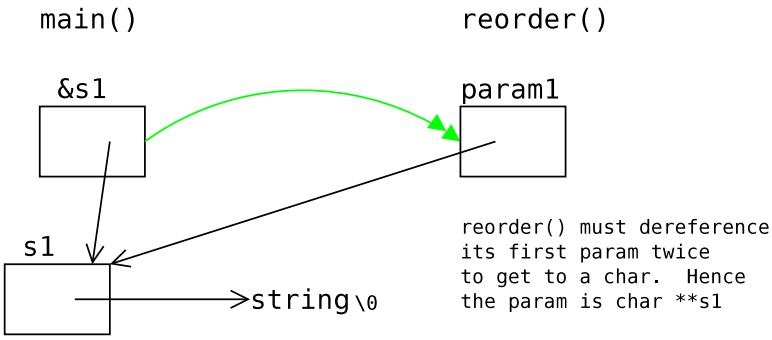
Write a C function

```
int countVowels(char *str)
```

You may assume that str points to a legal C string. For this problem the only vowels are 'a', 'A', 'e', 'E', 'i', 'I', 'o', 'O', 'u', and 'U'. The int returned by `countVowels()` is the total number of vowels found in the string (pointed to by) `str`. In addition, `countVowels()` modifies `str`, specifically it converts every uppercase vowel to lower case and vice versa.

Example: if originally `str = "Eabc@Uu"`, `countVowels(str)` returns 4 and, after the call, `str="eAbc@uU"`.

2 Ordering Strings by Length (20 Points)



Consider a string `char *str`. If you wanted a function `f()` to change a character in `str` to another character and return an `int` you would declare

```
int f(char *str)
```

and the caller would write `f(str)`

However, in this problem you are to write a function `f()` that is to change not individual characters, but strings themselves. Hence, the caller must write `f(&str)` and the declaration of the called program becomes

```
int f(char **str)
```

Write a C function

```
int reorder(char **s1, char **s2, char **s3)
```

If all the strings are of equal length, `reorder()` leaves all three strings unchanged, returns the common length, and prints "`All 3 are equal length.`"

If exactly two of the strings are equal in length, `reorder()` leaves all three strings unchanged, returns 2, and prints "`Two are equal length.`"

If all three strings are different in length, `reorder()` returns 1, sets `s1` to the longest string, sets `s2` to the middle length string, sets `s3` to the shortest string, and prints "`all three are unequal length`."

Feel free to use the `strlen()` and other library functions.

3 Upper and Lower Case (20 Points)

Write a C `main()` program

`int main(int argc, char *argv[])` This program has two optional command line argument, "-d" (indicating duplicate) and "-u" (indicating uppercase). As we did in class these arguments may be combined in several ways (search for "pink and yellow" in the lecture notes). Examples include "`-du`", "`-d`", "`-ud`", "`-d -u`", etc.

Your program reads characters with `getchar()` and, for each character read (call it `C`), the program prints corresponding character(s) using `putchar()`.

i. If there are no optional arguments, C is printed unchanged.

ii. If the duplicate option is present, C is printed twice.

iii. If C is alphabetic (use "isalpha(C)" to check) and the uppercase option is present C is printed in upper case.

iv. If both options are present, C is printed twice and, if C is alphabetic, both copies are printed in uppercase.

Feel free to use the `toupper()`, `isalpha()`, and other library functions.

4 Divisibility by 3 and 5 (25 Points Total)

4.1 `isdiv35()` (10 Points)

Write a C function

```
int isdiv35(int n)
```

`isdiv35(n)` returns 1 if `n` is divisible by 3 *and not* divisible by 5.

`isdiv35(n)` returns 2 if `n` is divisible by 5 *and not* divisible by 3.

`isdiv35(n)` returns 3 if `n` is divisible by *both* 3 and 5.

`isdiv35(n)` returns 4 if `n` is *not* divisible by 3 *and not* divisible by 5.

You may assume `n` is an integer greater than 1.

4.2 A C main Program to Test isdiv35() (15 Points)

```
int main(int argc, char *argv[])
```

This program first checks that `argc==3`. If it is not, the program prints an error message and terminates. You may assume that `argv[1]` and `argv[2]` are strings containing only digits. Have `atoi()` convert `argv[1]` and `argv[2]` to integers called `arg1` and `arg2`. Confirm that `arg1` <= `arg2` (print an error message if this is not so). Finally, for all `n`, such that `arg1` <= n <= `arg2`, call `isdiv35(n)` and print whether n is divisible by 3 and whether n is divisible by 5.

5 Three Command-line Arguments (20 Points)

Write a C main program that accepts three (non-optional) command-line arguments (not counting the command name itself). Call the first argument `str`, call the second `except` and call the third `dup`. Each argument is a string, i.e., a pointer to a `char`. Your program does all the following.

1. Checks that every character C in `str` is a lower-case letter. If a character is not a lower-case letter, then C is "illegal" and you print an error message and terminate the run immediately.
2. Assuming C is legal, check if it also appears in the `except` string.
3. If C does appear in `except`, skip it (i.e., print nothing for this character) and proceed to the next character in `str`.
4. If C does **not** appear in `except`, check if it appears in `dup`.
5. If C does **not** appear in either `except` or `dup`, print C using `putchar()`.
6. If C appears in `dup` but not in `except`, print C twice using `putchar()`.