ASSIGNMENT – CHARACTER STRINGS

1. (**processString**) Write a C function that accepts a string `str` and returns the total number of vowels `totVowels` and digits `totDigits` in that string to the caller via call by reference. The function prototype is given as follows:

```
void processString(char *str, int *totVowels, int *totDigits);
```

A sample program template is given below to test the function:

```
#include <stdio.h>
void processString(char *str, int *totVowels, int *totDigits);
int main()
{
   char str[50];
   int totVowels, totDigits;

   printf("Enter the string: \n");
   gets(str);
   processString(str, &totVowels, &totDigits);
   printf("Total vowels = %d\n", totVowels);
   printf("Total digits = %d\n", totDigits);
   return 0;
}
void processString(char *str, int *totVowels, int *totDigits)
{
   /* Write your program code here */
}
```

Some test input and output sessions are given below:

(1) Test Case 1:

```
Enter the string:
I am one of the 400 students in this class.
Total vowels = 11
Total digits = 3
```

(2) Test Case 2:

```
Enter the string:
I am a boy.
Total vowels = 4
Total digits = 0
```

(3) Test Case 3:

```
Enter the string:
1 2 3 4 5 6 7 8 9
Total vowels = 0
Total digits = 9
```

(4) Test Case 4:

```
Enter the string:
ABCDE
Total vowels = 2
Total digits = 0
```

2. (**cipherText**) Cipher text is a popular encryption technique. What we do in cipher text is that we can encrypt each apha `('a' .. 'z', 'A' .. 'Z')` character with +1. For example, `"Hello"` can

be encrypted with +1 cipher to `"Ifmmp"`. If a character is `'z'` or `'Z'`, the corresponding encrypted character will be `'a'` or `'A'` respectively. For other characters, no encryption is performed. We use call by reference in the implementation. Write the C functions `cipher()` and `decipher()` with the following function prototypes:

```
void cipher(char *s);
void decipher(char *s);
```

A sample program template is given below to test the functions:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void cipher(char *s);
void decipher(char *s);
int main()
{
   char str[80];

   printf("Enter the string: \n");
   gets(str);
   printf("To cipher: %s -> ", str);
   cipher(str);
   printf("%s\n", str);
   printf("To decipher: %s -> ", str);
   decipher(str);
   printf("%s\n", str);
   return 0;
}
void cipher(char *s)
{
   /* Write your program code here */
}
void decipher(char *s)
{
   /* Write your program code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter the string:
123a
To cipher: 123a -> 123b
To decipher: 123b -> 123a
```

(2) Test Case 2:

```
Enter the string:
abcxyz
To cipher: abcxyz -> bcdyza
To decipher: bcdyza -> abcxyz
```

(3) Test Case 3:

```
Enter the string:
HELLO Hello
To cipher: HELLO Hello -> IFMMP Ifmmp
To decipher: IFMMP Ifmmp -> HELLO Hello
```

(4) Test Case 4:

```
Enter the string:
x y z
To cipher: x y z -> y z a
To decipher: y z a -> x y z
```

3. (**maxCharToFront**) Write a C function that accepts a character string `str` as parameter, finds the largest character from the string, and moves it to the beginning of the string. E.g., if the string is `"adecb"`, then the string will be `"eadcb"` after executing the function. The string will be passed to the caller via call by reference. If more than one largest character is in the string, then the **first appearance** of the largest character will be moved to the beginning of the string. For example, if the string is `"adecbe"`, then the resultant string will be `"eadcbe"`. The function prototype is given as follows:

```
void maxCharToFront(char *str);
```

A sample program template is given below to test the function:

```
#include <stdio.h>
void maxCharToFront(char *str);
int main()
{
   char str[80];

   printf("Enter a string: \n");
   gets(str);
   printf("maxCharToFront(): ");
   maxCharToFront(str);
   puts(str);
   return 0;
}
void maxCharToFront(char *str)
{
   /* Write your code here */
}
```

Some test input and output sessions are given below:

(1) Test Case 1:

```
Enter a string:
adebc
maxCharToFront(): eadbc
```

(2) Test Case 2:

```
Enter a string:
agfcdeg
maxCharToFront(): gafcdeg
```

(3) Test Case 3:

```
Enter a string:
cba
maxCharToFront(): cba
```

(4) Test Case 4:

```
Enter a string:
ab
maxCharToFront(): ba
```

4. (**countSubstring**) Write a C function that takes in two parameters `str` and `substr`, and counts the number of substring `substr` occurred in the character string `str`. If the `substr` is not contained in `str`, then it will return 0. Please note that you do not need to consider test cases such as `str = "aooob"` and `substr = "oo"`. The function prototype is given as follows:

```
int countSubstring(char str[], char substr[]);
```

A sample program template is given below to test the function:

```
#include <stdio.h>
int countSubstring(char str[], char substr[]);
int main()
{
   char str[80],substr[80];

   printf("Enter the string: \n");
   gets(str);
   printf("Enter the substring: \n");
   gets(substr);
   printf("countSubstring(): %d\n", countSubstring(str, substr));
   return 0;
}
int countSubstring(char str[], char substr[])
{
   /* Write your program code here */
}
```

Some test input and output sessions are given below:

(1) Test Case 1:

```
Enter the string:
abcdef
Enter the substring:
dd
countSubstring(): 0
```

(2) Test Case 2:

```
Enter the string:
abcabcabc cbaf
Enter the substring:
abc
countSubstring(): 3
```

(3) Test Case 3:

```
Enter the string:
bababababaabf
Enter the substring:
ab
countSubstring(): 4
```