Section B – Character Strings [Ans 1 Specified Qn from this Section]

1. (**sweepSpace**) Write a C function `sweepSpace()` that removes all the blank spaces in a string. The function takes in a character string *`str`* as argument, and returns the resultant string to the calling function. For example, if *`str`* is `"I am a boy"`, the resultant string `"Iamaboy"` will be returned to the calling function. The function prototype is given below:

```
char *sweepSpace(char *str);
```

Write a C program to test the functions.

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter the string:
I am a boy
sweepSpace(): Iamaboy
```

(2) Test Case 2:
```
Enter the string:
anybody
sweepSpace(): anybody
```

A sample template for the program is given below:

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

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

2. (**processString**) Write a C function `processString()` that accepts a string, *`str`*, and returns the total number of vowels and digits 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);
```

Write a C program to test the function.

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
```

A sample template for the program is given below:

```
#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 */
}
```

3. (**stringncpy**) Write a C function `stringncpy()` that copies not more than *n* characters (characters that follow a null character are not copied) from the array pointed to by *s2* to the array pointed to by *s1*. If the array pointed to by *s2* is a string shorter than n characters, null characters are appended to the copy in the array pointed to by *s1*, until *n* characters in all have been written. The `stringncpy()` returns the value of *s1*. The function prototype is:

```
char *stringncpy(char *s1, char *s2, int n);
```

In addition, write a C program to test the stringncpy function. Your program should read the string and the target *n* characters from the user and then call the function with the user input. In this program, you are not allowed to use any functions from the C standard String library.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter the string:
I am a boy.
Enter the number of characters:
7
stringncpy(): I am a
```

(2) Test Case 2:

```
Enter the string:
I am a boy.
Enter the number of characters:
21
stringncpy(): I am a boy.
```

A sample program to test the function is given below:

```
#include <stdio.h>
char *stringncpy(char *s1, char *s2, int n);
```

```
int main()
{
   char targetStr[40], sourceStr[40], *target;
   int length;

   printf("Enter the string: \n");
   gets(sourceStr);
   printf("Enter the number of characters: \n");
   scanf("%d", &length);
   target = stringncpy(targetStr, sourceStr, length);
   printf("stringncpy(): %s\n", target);
   return 0;
}
char *stringncpy(char *s1, char *s2, int n)
{
   /* Write your program code here */
}
```

4. (**findTarget**) Write a C program that reads and searches character strings. In the program, it contains the function `findTarget()` that searches whether a target name string has been stored in the array of strings. The function prototype is

```
int findTarget(char *target, char nameptr[SIZE][80], int size);
```

where ***`nameptr`*** is the array of strings entered by the user, ***`size`*** is the number of names stored in the array and ***`target`*** is the target string. If the target string is found, the function will return its index location, or -1 if otherwise.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter size:
4
Enter 4 names:
Peter Paul John Mary
Enter target name:
John
findTarget(): 2
```

(2) Test Case 2:

```
Enter size:
5
Enter 5 names:
Peter Paul John Mary Vincent
Enter target name:
Jane
findTarget(): -1
```

A sample template for the program is given below:

```
#include <stdio.h>
#include <string.h>
#define SIZE 10
#define INIT_VALUE 10
int findTarget(char *target, char nameptr[SIZE][80], int size);
int main()
{
   char nameptr[SIZE][80];
   char t[40];
   int i, result=INIT_VALUE, size;

   printf("Enter size: \n");
   scanf("%d", &size);
```

```
    printf("Enter %d names: \n", size);
    for (i=0; i<size; i++)
       scanf("%s", nameptr[i]);
    printf("Enter target name: \n");
    scanf("\n");
    gets(t);
    result = findTarget(t, nameptr, size);
    printf("findTarget(): %d\n",  result);
    return 0;
}
int findTarget(char *target, char nameptr[SIZE][80], int size)
{
    /* Write your program code here */
}
```

5. (**cipherText**) Cipher text is a popular encryption technique. What we do in cipher text is that we can encrypt each 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. Write the C functions `cipher()` and `decipher()` with the following function prototypes:

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

Write a C program to test the `cipher()` and `decipher()` functions.

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
To cipher: HELLO -> IFMMP
To decipher: IFMMP -> HELLO
```

A sample template for the program is given below:

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

    printf("Enter the string: \n");
    gets(str);
    printf("To cipher: %s -> ", str);
    q = cipher(str);
    printf("%s\n", str);
    printf("To decipher: %s -> ", str);
```

```
    q = decipher(str);
    printf("%s\n", str);
    return 0;
}
char *cipher(char *s)
{
    /* Write your program code here */
}
char *decipher(char *s)
{
    /* Write your program code here */
}
```

6. (**findMinMaxStr**) Write a C function that reads in names separated by space, finds the first and last names according to ascending alphabetical order, and returns them to the calling function through the string parameters *first* and *last*. The calling function will then print the first and last names on the screen. You may assume that the first character in each input name is in capital letter, and the subsequent characters in the input name are lower case letters, e.g. Peter, Mary, Ann, etc. The function prototype is given as follows:

```
void findMinMaxStr(char word[][40], char *first, char *last, int
size);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter size:
4
Enter 4 words:
Peter Paul John Mary
First word = John, Last word = Peter
```

(2) Test Case 2:

```
Enter size:
1
Enter 1 words:
Peter
First word = Peter, Last word = Peter
```

(3) Test Case 3:

```
Enter size:
2
Enter 2 words:
Peter Mary
First word = Mary, Last word = Peter
```

A sample template for the program is given below:

```
#include <stdio.h>
#include <string.h>
#define SIZE 10
void findMinMaxStr(char word[][40], char *first, char *last, int
size);
int main()
{
    char word[SIZE][40];
    char first[40], last[40];
    int i, size;

    printf("Enter size: \n");
    scanf("%d", &size);
```

```
    printf("Enter %d words: \n", size);
    for (i=0; i<size; i++)
       scanf("%s", word[i]);
    findMinMaxStr(word, first, last, size);
    printf("First word = %s, Last word = %s\n", first, last);
    return 0;
}
void findMinMaxStr(char word[][40], char *first, char *last, int
size)
{
    /* Write your program code here */
}
```

7. (**countSubstring**) Write a C function `countSubstring()` 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. The function prototype is given as follows:

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

Write a C program to test the function.

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:
abababcdef
Enter the substring:
ab
countSubstring(): 3
```

A sample template for the program is given below:

```
#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 */
}
```