

## WEEK 5 TUTORIAL - 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>
#include <string.h>
void processString(char *str, int *totVowels, int *totDigits);
int main()
{
    char str[50], *p;
    int totVowels, totDigits;

    printf("Enter the string: \n");
    fgets(str, 80, stdin);
    if (p=strchr(str, '\n')) *p = '\0';
    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. (**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 given below:

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

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

```
#include <stdio.h>
#include <string.h>
char *stringncpy(char *s1, char *s2, int n);
int main()
{
    char targetStr[40], sourceStr[40], *target, *p;
    int length;

    printf("Enter the string: \n");
    fgets(sourceStr, 40, stdin);
    if (p=strchr(sourceStr, '\n')) *p = '\0';
    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 */
}
```

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.
- (3) Test Case 3:  
Enter the string:  
somebody  
Enter the number of characters:  
7  
stringncpy(): somebod
- (4) Test Case 4:  
Enter the string:

somebody

Enter the number of characters:

21

stringncpy(): somebody

3. (**stringcmp**) Write a C function that compares the string pointed to by *s1* to the string pointed to by *s2*. If the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*, then it returns 1, 0 or -1 respectively. Write the code for the function without using any of the standard C string library functions. The function prototype is given as follows:

```
int stringcmp(char *s1, char *s2);
```

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

```
#include <stdio.h>
#include <string.h>
#define INIT_VALUE 999
int stringcmp(char *s1, char *s2);
int main()
{
    char source[80], target[80], *p;
    int result = INIT_VALUE;

    printf("Enter a source string: \n");
    fgets(source, 80, stdin);
    if (p=strchr(source, '\n')) *p = '\0';
    printf("Enter a target string: \n");
    fgets(target, 80, stdin);
    if (p=strchr(target, '\n')) *p = '\0';
    result = stringcmp(source, target);
    if (result == 1)
        printf("stringcmp(): greater than");
    else if (result == 0)
        printf("stringcmp(): equal");
    else if (result == -1)
        printf("stringcmp(): less than");
    else
        printf("stringcmp(): error");
    return 0;
}
int stringcmp(char *s1, char *s2)
{
    /* Write your code here */
}
```

Some test input and output sessions are given below:

- (1) Test Case 1:  
 Enter a source string:  
abc  
 Enter a target string:  
abc  
 stringcmp(): equal
- (2) Test Case 2:  
 Enter a source string:  
abcdefg  
 Enter a target string:

abcde123  
 strcmp(): greater than

- (3) Test Case 3:  
 Enter a source string:  
abc123  
 Enter a target string:  
abcdef  
 strcmp(): less than

- (4) Test Case 4:  
 Enter a source string:  
abcdef  
 Enter a target string:  
abcdefg  
 strcmp(): less than

4. What does the following program print?

```
#include <stdio.h>
#define M1 "How are ya, sweetie?"
char M2[40] = "Beat the clock.";
char *M3 = "chat";
int main()
{
    char words[80];
    printf(M1);
    puts(M2);
    puts(M2+1);
    gets(words);           /* user inputs : win a toy. */
    puts(words);
    scanf("%s", words+6);  /* user inputs : snoopy. */
    puts(words);
    words[3] = '\0';
    puts(words);
    while (*M3) puts(M3++);
    puts(--M3);
    puts(--M3);
    M3 = M1;
    puts(M3);
    return 0;
}
```