Section F – Character Strings [Ans 2 Specified Qns from this Section]

1. **(insertChar)** Write the C function `insertChar()` that takes in a string *`str1`* as an argument, copies the contents of character string *`str1`* into character string *`str2`*. For every three characters copied from *`str1`* to *`str2`*, a character `'#'` is inserted into *`str2`*. The function returns the resultant string to the calling function via call by reference. For example, if the string *`str1`* is `"abcdefg"`, then the resultant string *`str2`* = `"abc#def#g"` will be returned to the calling function. The function prototype is:

```
void insertChar(char *str1, char *str2);
```

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter a string:
abc de
insertChar(): abc# de#
```

(2) Test Case 2:

```
Enter a string:
abc123456de
insertChar(): abc#123#456#de
```

A sample program to test the functions is given below:

```
#include <stdio.h>
void insertChar(char *str1, char *str2);
int main()
{
   char a[80],b[80];

   printf("Enter a string: \n");
   gets(a);
   insertChar(a,b);
   printf("insertChar(): ");
   puts(b);
   return 0;
}
void insertChar(char *str1, char *str2)
{
  /* Write your code here */
}
```

2. (**compareStr**) Write a C function `compareStr()` that takes in two parameters *`s`* and *`t`*, and compares the two character strings *`s`* and *`t`* according to alphabetical order. If *`s`* is greater than *`t`*, then it will return a positive value. Otherwise, it will return a negative value. For example, if *`s`* is "boy" and *`t`* is "girl", then the function will return -5 which is the difference between the ASCII values of 'b' and 'g'. If *`s`* is "car" and *`t`* is "apple", then it will return 2 which is the difference between the ASCII values of 'c' and 'a'. You should not use any string functions from the standard C library in this function. The function prototype is given as follows:

```
int compareStr(char *s, char *t);
```

Write a C program to test the function.

Some test input and output sessions are given below:

(1) Test Case 1:

```
Enter the first string:
boy
Enter the second string:
```

```
girl
compareStr(): -5
```

(2) Test Case 2:

```
Enter the first string:
car
Enter the second string:
apple
compareStr(): 2
```

(3) Test Case 3:

```
Enter the first string:
abcd
Enter the second string:
abcD
compareStr(): 32
```

A sample template for the program is given below:

```
#include <stdio.h>
int compareStr(char *s, char *t);
int main()
{
   char a[80],b[80];

   printf("Enter the first string: \n");
   gets(a);
   printf("Enter the second string: \n");
   gets(b);
   printf("compareStr(): %d\n", compareStr(a,b));
   return 0;
}
int compareStr(char *s, char *t)
{
   /* Write your code here */
}
```

3. (**countWords**) Write a function `countWords()` that accepts a string `s` as its parameter. The string contains a sequence of words separated by spaces. The function then displays the number of words in the string. The function prototype is given as follows:

```
int countWords(char *s);
```

Write a C program to test the function.

A sample input and output session is given below:

(1) Test Case 1:

```
Enter the string:
How are you?
countWords(): 3
```

(2) Test Case 2:

```
Enter the string:
There are 12 dollars.
countWords(): 4
```

(3) Test Case 3:

```
Enter the string:
Oneword?
countWords(): 1
```

A sample template for the program is given below:

```
#include <stdio.h>
int countWords(char *s);
int main()
{
   char str[50];

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

4. (**longWordLength**) Write a C function that accepts an English sentence as parameter, and returns the length of the longest word in the sentence. For example, if the sentence is `"I am happy."`, then the length of the longest word `"happy"` in the sentence 5 will be returned. Assume that each word is a sequence of English letters. The function prototype is given as follows:

   ```
   int longWordLength(char *s);
   ```

   Write a C program to test the function.

   Some test input and output sessions are given below:

   (1) Test Case 1:
   ```
   Enter a string:
   I am happy.
   longWordLength(): 5
   ```

   (2) Test Case 2:
   ```
   Enter a string:
   There are 40 students in the class.
   longWordLength(): 8
   ```

   A sample template for the program is given below:

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

      printf("Enter a string: \n");
      gets(str);
      printf("longWordLength(): %d\n", longWordLength(str));
      return 0;
   }
   int longWordLength(char *s)
   {
      /* Write your code here */
   }
   ```

5. (**longestStrInAr**) Write a C function `longestStrInAr()` that takes in an array of strings *`str`* and *`size`* (>0) as paramters, returns the longest string and also the length of the longest string via the pointer parameter *`length`*. If two or more strings have the same longest string length, then the first appeared string will be retruned to the calling function. For example, if *`size`* is 5 and the array of strings is

{"peter","john","mary","jane","kenny"}, then the longest string is "peter" and the string length is 5 will be returned to the calling function. The function prototype is:

```
char *longestStrInAr(char str[N][40], int size, int *length);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter array size:
4
Enter string 1:
Kenny
Enter string 2:
Mary
Enter string 3:
Peter
Enter string 4:
Sun
longest: Kenny
length: 5
```

(2) Test Case 2:

```
Enter array size:
2
Enter string 1:
Sun
Enter string 2:
Mary
longest: Mary
length: 4
```

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

```
#include <stdio.h>
#include <string.h>
#define N 20
char *longestStrInAr(char str[N][40], int size, int *length);
int main()
{
   int i, size, length;
   char str[N][40], first[40], last[40], *p;
   char dummychar;

   printf("Enter array size: \n");
   scanf("%d", &size);
   scanf("%c", &dummychar);
   for (i=0; i<size; i++) {
      printf("Enter string %d: \n", i+1);
      gets(str[i]);
   }
   p = longestStrInAr(str, size, &length);
   printf("longest: %s \nlength: %d\n", p, length);
   return 0;
}
char *longestStrInAr(char str[N][40], int size, int *length)
{
   /* Write your code here */
}
```

6. (**palindrome**) Write a function `palindrome()` that reads a character string and determines whether or not it is a `palindrome`. A palindrome is a sequence of characters that reads the same forwards and

backwards. For example, `"abba"` and `"abcba"` are `palindromes`, but `"abcd"` is not. The function returns 1 if it is `palindrome`, or 0 if otherwise. The function prototype is given as follows:

```
int palindrome(char *str);
```

Write a C program to test the function.

Some test input and output sessions are given below:

(1) Test Case 1:

```
Enter a string:
abcba
palindrome(): A palindrome
```

(2) Test Case 2:

```
Enter a string:
abba
palindrome(): A palindrome
```

(3) Test Case 3:

```
Enter a string:
abcde
palindrome(): Not a palindrome
```

A sample template for the program is given below:

```
#include <stdio.h>
#define INIT_VALUE -1
int palindrome(char *str);
int main()
{
   char str[80];
   int result = INIT_VALUE;

   printf("Enter a string: \n");
   gets(str);
   result = palindrome(str);
   if (result == 1)
      printf("palindrome(): A palindrome\n");
   else if (result == 0)
      printf("palindrome(): Not a palindrome\n");
   else
      printf("An error\n");
   return 0;
}
int palindrome(char *str)
{
   /* Write your code here */
}
```

7. **(strInterset)** Write the C function `strInterset()` that takes in three strings ***str1***, ***str2*** and ***str3*** as parameters, stores the same characters that appeared in both ***str1*** and ***str2*** into the string, and returns ***str3*** to the calling function via call by reference. For example, if ***str1*** is `"abcdefghijk"` and ***str2*** is `"123i4bc78h9"`, then ***str3*** is `"bchi"` will be returned to the calling function after executing the function. If there is no common characters in the two strings, ***str3*** will be a null string. You may assume that each string contains unique characters in the string, i.e. the characters contained in the same string will not be repeated. The function prototype is:

```
void strInterset(char *str1, char *str2, char *str3);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter str1:
abcde
Enter str2:
dec
strIntersect(): cde
```

(2) Test Case 2:

```
Enter str1:
abcdefghijk
Enter str2:
akdhf
strIntersect(): adfhk
```

(3) Test Case 3:

```
Enter str1:
abc
Enter str2:
def
strIntersect(): null string
```

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

```
#include <stdio.h>
void strInterset(char *str1, char *str2, char *str3);
int main()
{
   char str1[50],str2[50],str3[50];

   printf("Enter str1: \n");
   scanf("%s",str1);
   printf("Enter str2: \n");
   scanf("%s",str2);
   strInterset(str1, str2, str3);
   if (*str3 == '\0')
      printf("strIntersect(): null string\n");
   else
      printf("strIntersect(): %s\n", str3);
   return 0;
}
void strInterset(char *str1, char *str2, char *str3)
{
   /* Write your code here */
}
```

8. (**maxCharToFront**) Write a C function `maxCharToFront()` 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 `"aedcb"`, then the string will be `"eadcb"` after executing the function. The string will be passed to the caller via call by reference. You may assume that the string will contain unique lower case characters. The function prototype is given as follows:

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

Write a C program to test the function.

Some test input and output sessions are given below:

(1) Test Case 1:

```
Enter a string:
```

```
abcde
maxCharToFront(): eabcd
```

(2) Test Case 2:

```
Enter a string:
abfcde
maxCharToFront(): fabcde
```

A sample template for the program is given below:

```
#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 */
}
```

9. (**mergeStr**) Write a C function `mergeStr()` that merges two alphabetically ordered character strings *a* and *b* into character string *c* according to alphabetical order. For example, if *a* is `"agikmpq"` and *b* is `"bcdefhjlnr"`, then the string *c* will be `"abcdefghijklmnpqr"`. The string *c* will be passed to the caller via call by reference. The function prototype is given as follows:

```
void mergeStr(char *a, char *b, char *c);
```

Write a C program to test the function.

Some test input and output sessions are given below:

(1) Test Case 1:

```
Enter the first string:
ace
Enter the second string:
bdg
mergeStr(): abcdeg
```

(2) Test Case 2:

```
Enter the first string:
agikmpq
Enter the second string:
bcdefhjlnr
mergeStr(): abcdefghijklmnpqr
```

(3) Test Case 3:

```
Enter the first string:
afkm
Enter the second string:
bbbggg
mergeStr(): abbbfgggkm
```

A sample template for the program is given below:

```c
#include <stdio.h>
#include <string.h>
void mergeStr(char *a, char *b, char *c);
int main()
{
   char a[80],b[80];
   char c[80];

   printf("Enter the first string: \n");
   gets(a);
   printf("Enter the second string: \n");
   gets(b);
   mergeStr(a,b,c);
   printf("mergeStr(): ");
   puts(c);
   return 0;
}
void mergeStr(char *a, char *b, char *c)
{
   /* Write your code here */
}
```

10. (**findSubstring**) Write a C function `findSubstring()` that takes two character string arguments, ***str*** and ***substr*** as input and returns 1 if ***substr*** is a substring of ***str*** (i.e. if ***substr*** is contained in ***str***) and 0 if not. For example, the function will return 1 if ***substr*** is `"123"` and ***str*** is `"abc123xyz"`, but it will return 0 if otherwise. Note that for this question you are not allowed to use any string functions from the standard C library. The prototype of the function is given below:

```c
int findSubstring(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:
abcde
Enter the substring:
abc
findSubstring(): 1
```

(2) Test Case 2:

```
Enter the string:
abcde
Enter the substring:
cdef
findSubstring(): 0
```

A sample template for the program is given below:

```c
#include <stdio.h>
#define INIT_VALUE -1
int findSubstring(char *str, char *substr);
int main()
{
   char str[40], substr[40];
   int result = INIT_VALUE;

   printf("Enter the string: \n");
   gets(str);
   printf("Enter the substring: \n");
   gets(substr);
   result = findSubstring(str, substr);
```

```
   if (result == 1)
      printf("findSubstring(): Is a substring\n");
   else if ( result == 0)
      printf("findSubstring(): Not a substring\n");
   else
      printf("findSubstring(): An error\n");
   return 0;
}
int findSubstring(char *str, char *substr)
{
   /* Write your code here */
}
```