Week 4 Practice Questions – Character Strings

1. insertChar
2. delNum
3. convertCaseStr
4. locateFirstChar
5. locateLastChar
6. reverseStr
7. processString
8. compareStr
9. longWordLength
10. stringcmp
11. countWords
12. cipherText
13. findMinMaxStr
14. longestStrInAr
15. maxCharToFront
16. strIntersect
17. findSubstring
18. countSubstring

Questions

1. **(insertChar)** Write the C function that takes in a string `str1` as an argument, copies the contents of character string `str1` into character string `str2`. In addition, the function also has a character parameter `ch`. For every three characters copied from `str1` to `str2`, the character `ch` 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"`, and the inserted character `ch` is `'#'`, then the resultant string `str2` = `"abc#def#g"` will be returned to the calling function. The function prototype is given as follows:

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

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

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

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

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter a string:
```

```
abc de
Enter a character to be inserted:
#
insertChar(): abc# de#
```

(2) Test Case 2:

```
Enter a string:
abc
Enter a character to be inserted:
#
insertChar(): abc#
```

(3) Test Case 3:

```
Enter a string:
I am a boy.
Enter a character to be inserted:
$
insertChar(): I a$m a$ bo$y.
```

(4) Test Case 4:

```
Enter a string:
hi
Enter a character to be inserted:
$
insertChar(): hi
```

2. **(delNum)** Write the C function that takes in a string `str` as a parameter, removes any numerical characters in the string, and returns the resultant string to the calling function via call by reference. For example, if the string `str` is `"ab12def"`, then the resultant string `"abdef"` will be returned to the calling function. The function prototype is given as follows:

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

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

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

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

Some sample input and output sessions are given below:

(1) Test Case 1

```
Enter a string:
ab12def
delNum(): abdef
```

(2) Test Case 2

```
Enter a string:
I have 10 dollars
delNum(): I have  dollars
```

3. (**convertCaseStr**) Write a C function that takes a character string `str` as argument, and converts lower case characters into upper case characters, and upper case characters into lower case characters. The function prototype is given as follows:

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

A sample template for the program is given below:

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

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

Some sample input and output sessions are given below:

(1) Test Case 1

```
Enter the string:
i am a boy
convertCaseStr(): I AM A BOY
```

(2) Test Case 2

```
Enter the string:
I am a BOY
convertCaseStr(): i AM A boy
```

4. (**locateFirstChar**) Write a C function that locates the first occurrence of `ch` in the string `str`. The function returns the index, or -1 if `ch` does not occur in the string. The function prototype is given as follows:

```
int locateFirstChar(char *str, char ch);
```

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

```
#include <stdio.h>
int locateFirstChar(char *str, char ch);
int main()
{
   char str[40], ch;

   printf("Enter a string: \n");
   gets(str);
   printf("Enter the target character: \n");
   scanf("%c", &ch);
   printf("locateFirstChar(): %d\n", locateFirstChar(str, ch));
   return 0;
}
int locateFirstChar(char *str, char ch)
```

```
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1

```
Enter a string:
I am a boy
Enter the target character: a
locateFirstChar(): 2
```

(2)Test Case 2

```
Enter a string:
I am a boy
Enter the target character: z
locateFirstChar(): -1
```

5. (**locateLastChar**) Write a C function that locates the last occurrence of `ch` in the string `str`. The function returns the index, or -1 if `ch` does not occur in the string. The function prototype is given as follows:

```
int locateLastChar(char *str, char ch);
```

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

```
#include <stdio.h>
int locateLastChar(char *str, char ch);
int main()
{
   char str[40], ch;

   printf("Enter a string: \n");
   gets(str);
   printf("Enter the target character: \n");
   scanf("%c", &ch);
   printf("locateLastChar(): %d\n", locateLastChar(str, ch));
   return 0;
}
int locateLastChar(char *str, char ch)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1

```
Enter a string: I am a boy
Enter the target character: a
locateLastChar(): 5
```

(2) Test Case 2

```
Enter a string: I am a boy
Enter the target character: z
locateLastChar(): -1
```

6. (**reverseStr**) Write a C function that accepts a character string `str` as its parameter and reverses the contents of the string. The function returns the reversed string to the calling function through the parameter `str`. The function prototype is given as follows:

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

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

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

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

Some test input and output sessions are given below:

(1) Test Case 1

```
Enter a string:
I am a boy
reverseStr(): yob a ma I
```

(2) Test Case 2

```
Enter a string:
abcde
reverseStr(): edcba
```

7. (**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
```

8. (**compareStr**) Write a C function 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);
```

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

```
#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 */
}
```

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:
abc
Enter the second string:
abcD
compareStr(): -68
```

9. (**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);
```

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

```
#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 */
}
```

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 forty students in the class.
longWordLength(): 8
```

(3) Test Case 3:

```
Enter a string:
Good day!
longWordLength(): 4
```

(4) Test Case 4:

```
Enter a string:
Hello!
longWordLength(): 5
```

10. (**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:

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

   printf("Enter a source string: \n");
   gets(source);
   printf("Enter a target string: \n");
   gets(target);
   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
stringcmp(): greater than
```

(3) Test Case 3:

```
Enter a source string:
abc123
Enter a target string:
abcdef
stringcmp(): less than
```

(4) Test Case 4:

```
Enter a source string:
abcdef
Enter a target string:
abcdefg
stringcmp(): less than
```

11. (**countWords**) Write a function 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);
```

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

```
#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 */
}
```

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
```

12. (**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
```

13. (**findMinMaxStr**) Write a C function that reads in words separated by space, finds the first and last words 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 strings on the screen. The function prototype is given as follows:

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

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

```
#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 */
}
```

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
```

14. (**longestStrInAr**) Write a C function that takes in an array of strings `str` and `size` (>0) as paramters, and 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);
```

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

```
#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 */
}
```

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
```

15. (**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
```

16. **(strIntersect)** Write the C function 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, i.e. the characters contained in the same string will not be repeated. The function prototype is given as follows:

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

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

```
#include <stdio.h>
void strIntersect(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);
   strIntersect(str1, str2, str3);
   if (*str3 == '\0')
      printf("strIntersect(): null string\n");
   else
      printf("strIntersect(): %s\n", str3);
   return 0;
}
void strIntersect(char *str1, char *str2, char *str3)
{
   /* Write your code here */
}
```

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
```

17. (**findSubstring**) Write a C function 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:

```
int findSubstring(char *str, char *substr);
```

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

```
#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 */
}
```

Some test input and output sessions are given below:

(1) Test Case 1:

```
Enter the string:
abcde fgh
Enter the substring:
abc
findSubstring(): Is a substring
```

(2) Test Case 2:

```
Enter the string:
abcde f
```

```
Enter the substring:
cdef
findSubstring(): Not a substring
```

18. (**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:
ababab cdef
Enter the substring:
ab
countSubstring(): 3
```