WEEK 5 - CHARACTER STRINGS

You are required to do the following:

1. Lab Questions – please do the lab questions during the lab session. When doing your lab questions, please follow exactly the question requirements on program input/output as our automated assessment system is based on test cases using exact string matching on program input/output.
2. Lab Assignment Questions – please do the assignment questions and submit your code to the online Automated Programming Assessment System (APAS) for grading.

Lab Questions

1. **(sweepSpace)** Write two versions of a C function that remove all the blank spaces in a string. The first version `sweepSpace1()` will use array notation for processing the string, while the other version `sweepSpace2()` will use pointer notation. In this program, you are not allowed to use any functions from the C standard String library.The function prototypes are given below:

```
char *sweepSpace1(char *str);
char *sweepSpace2(char *str);
```

Write a C program to test the functions. In this program, you are not allowed to use any functions from the C standard String library.

A sample input and output session is given below:

(1) Test Case 1:
```
Enter the string:
i am a boy
sweepSpace1(): iamaboy
sweepSpace2(): iamaboy
```

(2) Test Case 2:
```
Enter the string:
anybody
sweepSpace1(): anybody
sweepSpace2(): anybody
```

A sample template for the program is given below:

```
#include <stdio.h>
char *sweepSpace1(char *str);
char *sweepSpace2(char *str);
int main()
{
   char str[80];

   printf("Enter the string: \n");
   gets(str);
   printf("sweepSpace1(): %s\n", sweepSpace1(str));
   printf("sweepSpace2(): %s\n", sweepSpace2(str));
   return 0;
}
char *sweepSpace1(char *str)
{
   int i, j, len;

   i=0; len=0;
   while (str[i]!='\0'){
      len++;
      i++;
   }
   j = 0;
   for (i=0; i < len; i++)
   {
         if (str[i] != ' ')
         {
             // update the string by removing any space detected
         }
   }
   str[j] = '\0'; // add a null character
   return str;
}
char *sweepSpace2(char *str)
{
   /* Write your program code here */
}
```

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:

```
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.
```

(3) Test Case 4:
```
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
```

A sample program to test the function is given below:

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

3. (**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[][80], int size);
```

where *`nameptr`* is the array of strings, *`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. In addition, it also contains the function readNames() that reads a number of names from the user. The function prototype is

```
void readNames(char nameptr[][80], int *size);
```

where *`nameptr`* is the array of strings to store the input names, and *`size`* is a pointer parameter which passes the number of names to the caller.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Select one of the following options:
1: readNames()()
2: findTarget()
3: printNames()
4: exit()
Enter your choice:
1
Enter size:
4
```

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

(2) Test Case 2:

```
Select one of the following options:
1: readNames()()
2: findTarget()
3: printNames()
4: exit()
Enter your choice:
1
Enter size:
5
Enter 5 names:
Peter Paul John Mary Vincent
Enter your choice:
2
Enter target name:
Jane
findTarget(): -1
Enter your choice:
4
```

(3) Test Case 3:

```
Select one of the following options:
1: readNames()()
2: findTarget()
3: printNames()
4: exit()
Enter your choice:
1
Enter size:
5
Enter 5 names:
Peter Paul John Mary Vincent
Enter your choice:
3
Peter Paul John Mary Vincent
```

(4) Test Case 4:

```
Select one of the following options:
1: readNames()()
2: findTarget()
3: printNames()
4: exit()
Enter your choice:
1
Enter size:
6
Enter 6 names:
Peter Paul John Mary Vincent Joe
```

```
Enter your choice:
2
Enter target name:
Joe
findTarget(): 5
Enter your choice:
4
```

A sample template for the program is given below:

```
#include <stdio.h>
#include <string.h>
#define SIZE 10
#define INIT_VALUE 999
void printNames(char nameptr[][80], int size);
void readNames(char nameptr[][80], int *size);
int findTarget(char *target, char nameptr[][80], int size);
int main()
{
   char nameptr[SIZE][80], t[40];
   int size, result = INIT_VALUE;
   int choice;

   printf("\nSelect one of the following options: \n");
   printf("1: readNames()\n");
   printf("2: findTarget()\n");
   printf("3: printNames()\n");
   printf("4: exit()\n");
   do {
      printf("Enter your choice: \n");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            readNames(nameptr, &size);
            break;
         case 2:
            printf("Enter target name: \n");
            scanf("\n");
            gets(t);
            result = findTarget(t, nameptr, size);
            printf("findTarget(): %d\n",  result);
            break;
         case 3:
            printNames(nameptr, size);
            break;
      }
   } while (choice < 4);
   return 0;
}
void printNames(char nameptr[][80], int size)
{
   /* Write your program code here */
}
void readNames(char nameptr[][80], int *size)
{
   /* Write your program code here */
}
int findTarget(char *target, char nameptr[][80], int size)
{
   /* Write your program code here */
}
```

4. (**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
```

(4) Test Case 4:
```
Enter a string:
abb a
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 */
```

```
        }
```

Section B – Character Strings

1. (**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 */
}
```

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

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. (**cipherText**) Cipher text is a popular encryption technique. What we do in cipher text is that we can encrypt each apha (English) 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. 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);
```

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

A sample template for the program is given below:

```
#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 */
}
```

5. (**findMinMaxStr**) Write a C function that reads in five 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);
```

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 */
}
```

6. (**longestStrInAr**) Write a C function `longestStrInAr()` 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);
```

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 */
}
```

7. **(strIntersect)** Write the C function `strIntersect()` 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 strIntersect(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 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 */
}
```

8. **(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:

```
#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 */
}
```

9. (**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:

```
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(): Is a substring
```

(2) Test Case 2:
```
Enter the string:
abcde
Enter the substring:
cdef
findSubstring(): Not a substring
```

A sample template for the program is given below:

```
#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 */
}
```

10. (**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. 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[]);
    ```

    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 */
    }
    ```