Programming Lab Test (Toal time = 1 hr 30 minutes)

Instructions

1. This is a closed book test.
2. Lab test will be held on week 8 (6-10 March) of Semester 2 in your lab tutorial session.
3. The question paper contains 5 sections, A, B, C, D and E. Each section contains 6 questions.
4. You are required to answer a total of 5 questions with one question from each section. The questions to be answered in the lab test will be randomly chosen during the lab test session.
5. Each question carries 20 marks. Total marks = 100.
6. The submitted question code will be marked according to its correctness. If the question code is unable to compile, you will get 0 mark for that question.
7. If you need a piece of rough paper to draft your code, please ask the lab technician for it.
8. **Program templates for the questions will be given in the test. DO NOT CHANGE the code in the main() function inside the program template.**

Notes

1. Section A – contains questions from lab and tutorial on Functions and Pointers.
2. Section B – contains questions from lab and tutorial on Arrays.
3. Section C – contains questions from lab and tutorial on Character Strings.
4. Section D – contains questions from lab and tutorial on Structures.
5. Section E – contains questions from lab and tutorial on Recursive Functions.

Section A – Functions and Pointers [Ans 1 Specified Qn from this Section]

1. (**numDigits**) Write a function that counts the number of digits for a non-negative integer. For example, 1234 has 4 digits. The function **numDigits1()** returns the result. The function prototype is given below:

```
int numDigits1(int num);
```

Write another function **numDigits2()** that passes the result through the pointer parameter, *result*. The function prototype is given below:

```
void numDigits2(int num, int *result);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)

```
Enter the number:
5
numDigits1(): 1
numDigits2(): 1
```

(2)

```
Enter the number:
13579
numDigits1(): 5
numDigits2(): 5
```

A sample program to test the function is given below:

```c
#include <stdio.h>
int numDigits1(int num);
void numDigits2(int num, int *result);
int main()
{
   int number, result=-1;

   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("numDigits1(): %d\n", numDigits1(number));
   numDigits2(number, &result);
   printf("numDigits2(): %d\n", result);
   return 0;
}
int numDigits1(int num)
{
    /* Write your code here */
}
void numDigits2(int num, int *result)
{
    /* Write your code here */
}
```

2. (**digitPos**) Write the function **digitPos1()** that returns the position of the first appearance of a specified digit in a positive number. The position of the digit is counted from the right and starts from 1. If the required digit is not in the number, the function should return 0. For example, digitPos1(12315, 1) returns 2 and digitPos1(12, 3) returns 0. The function prototype is given below:

```c
int digitPos1(int num, int digit);
```

Write another function **digitPos2()** that passes the result through the pointer parameter, *result*. For example, if num = 12315 and digit = 1, then *result = 2 and if num=12 and digit = 3, then *result = 0. The function prototype is given below:

```c
void digitPos2(int num, int digit, int *result);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)

```
Enter the number:
1534567
Enter the digit:
6
digitPos1(): 2
digitPos2(): 2
```

(2)

```
Enter the number:
1234567
Enter the digit:
8
digitPos1(): 0
digitPos2(): 0
```

A sample program to test the function is given below:

```c
#include <stdio.h>
int digitPos1(int num, int digit);
void digitPos2(int num, int digit, int *result);
int main()
{
   int number, digit, result=-1;

   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("Enter the digit: \n");
   scanf("%d", &digit);
   printf("digitPos1(): %d\n", digitPos1(number, digit));
```

```
    digitPos2(number, digit, &result);
    printf("digitPos2(): %d\n", result);
    return 0;
}
int digitPos1(int num, int digit)
{
    /* Write your code here */
}
void digitPos2(int num, int digit, int *result)
{
    /* Write your code here */
}
```

3. (**power**) Write a function that computes the power of a number. The power may be any integer value. Write two iterative versions of the function. The function **power1()** returns the computed result, while **power2()** passes the result through the pointer parameter *result*. The function prototypes are given below:

```
float power1(float num, int p);
void power2(float num, int p, float *result);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)
```
Enter the number and power:
2 3
power1(): 8.00
power2(): 8.00
```
(2)
```
Enter the number and power:
2 -4
power1(): 0.06
power2(): 0.06
```
(3)
```
Enter the number and power:
2 0
power1(): 1.00
power2(): 1.00
```

A sample program to test the function is given below:

```
#include <stdio.h>
float power1(float num, int p);
void power2(float num, int p, float *result);
int main()
{
    int power;
    float number, result;

    printf("Enter the number and power: \n");
    scanf("%f %d", &number, &power);
    printf("power1(): %.2f\n", power1(number, power));
    power2(number,power,&result);
    printf("power2(): %.2f\n", result);
    return 0;
}
float power1(float num, int p)
{
    /* Write your code here */
}
void power2(float num, int p, float *result)
{
    /* Write your code here */
}
```

4. (**gcd**) Write a C function gcd() that computes the greatest common divisor. For example, if num1 is 4 and num2 is 7, then the function will return 1; if num1 is 4 and num2 is 32, then the function will return 4; and

if num1 is 4 and num2 is 38, then the function will return 2. Write two iterative versions of the function. The function **gcd11()** returns the computed result, while **gcd2()** passes the result through the pointer parameter *result*. The function prototypes are given as follows:

```
int gcd1(int num1, int num2);
void gcd2(int num1, int num2, int *result);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)
```
Enter 2 numbers:
4 7
gcd1(): 1
gcd2(): 1
```
(2)
```
Enter 2 numbers:
4 32
gcd1(): 4
gcd2(): 4
```
(3)
```
Enter 2 numbers:
4 38
gcd1(): 2
gcd2(): 2
```

A sample program to test the function is given below:

```
#include <stdio.h>
int gcd1(int num1, int num2);
void gcd2(int num1, int num2, int *result);
int main()
{
   int x,y,result;
   printf("Enter 2 numbers: \n");
   scanf("%d %d", &x, &y);
   printf("gcd1(): %d\n", gcd1(x, y));
   gcd2(x,y,&result);
   printf("gcd2(): %d\n", result);
   return 0;
}
int gcd1(int num1, int num2)
{
   /* Write your code here */
}
void gcd2(int num1, int num2, int *result)
{
   /* Write your code here */
}
```

5. (**countOddDigits**) Write a C function to count the number of odd digits, i.e. digits with values 1,3,5,7,9 in a non-negative number. For example, if num is 1234567, then 4 will be returned. Write the function in two versions. The function countOddDigits1() returns the result to the caller, while countOddDigits2() passes the result through the pointer parameter *result*. The function prototypes are given below:

```
int countOddDigits1(int num);
void countOddDigits2(int num, int *result);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)
```
Enter the number:
124567
countOddDigits1(): 3
countOddDigits2(): 3
```

(2)

```
Enter the number:
2468
countOddDigits1(): 0
countOddDigits2(): 0
```

A sample program to test the function is given below:

```
#include <stdio.h>
int countOddDigits1(int num);
void countOddDigits2(int num, int *result);
int main()
{
   int number, result=-1;

   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("countOddDigits1(): %d\n", countOddDigits1(number));
   countOddDigits2(number, &result);
   printf("countOddDigits2(): %d\n", result);
   return 0;
}
int countOddDigits1(int num)
{
   /* Write your code here */
}
void countOddDigits2(int num, int *result)
{
   /* Write your code here */
}
```

6. (**extOddDigits**) Write a function that extracts the odd digits from a positive number, and combines the odd digits sequentially into a new number. If the input number does not contain any odd digits, then the function returns 0. For example, if the input number is 1234567, then 1357 will be returned; and if the input number is 28, then 0 will be returned. Write the function in two versions. The function extOddDigits1() returns the result to the caller, while countOddDigits2() passes the result through the pointer parameter *result*. The function prototypes are given as follows:

```
int extOddDigits1(int num);
void extOddDigits2(int num, int *result);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)

```
Enter the number:
1234567
extOddDigits1(): 1357
extOddDigits2(): 1357
```

(2)

```
Enter the number:
246
extOddDigits1(): 0
extOddDigits2(): 0
```

A sample program to test the function is given below:

```
#include <stdio.h>
int extOddDigits1(int num);
void extOddDigits2(int num, int *result);
int main()
{
   int number, result=-1;

   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("extOddDigits1(): %d\n",extOddDigits1(number));
   extOddDigits2(number, &result);
```

```
    printf("extOddDigits2(): %d\n", result);
    return 0;
}
int extOddDigits1(int num)
{
    /* Write your code here */
}
void extOddDigits2(int num, int *result)
{
    /* Write your code here */
}
```

Section B - Arrays [Answer 1 Specified Qn from this Section]

1. (**absoluteSum**) Write a C function **`absoluteSum()`** that returns the sum of the absolute values of the elements of a vector with the following prototype:

```
float absoluteSum(int size, float vector[]);
```

where **`size`** is the number of elements in the vector.

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)

```
Enter vector size:
5
Enter 5 data:
1.1 3 5 7 9
absoluteSum(): 25.10
```

(2)

```
Enter vector size:
5
Enter 5 data:
1 -3 5 -7 9
absoluteSum(): 25.00
```

A sample program to test the functions is given below.

```
#include <stdio.h>
#include <math.h>
float absoluteSum(int size, float vector[]);
int main()
{
    float vector[10];
    int i, size;

    printf("Enter vector size: \n");
    scanf("%d", &size);
    printf("Enter %d data: \n", size);
    for (i=0; i<size; i++)
        scanf("%f", &vector[i]);
    printf("absoluteSum(): %.2f\n", absoluteSum(size, vector));
    return 0;
}
float absoluteSum(int size, float vector[])
{
    /* write your code here */
}
```

2. (**findMinMax**) Write a C function findMinMax() that takes an one-dimensional array of integer numbers as a parameter. The function finds the minimum and maximum numbers of the array. The function returns the minimum and maximum numbers through the pointer parameters min and max. The function prototype is given as follows:

```
void findMinMax(int ar[], int size, int *min, int *max);
```

Write a C program to test the function.

Some sample input and output sessions are given below:
(1)

```
Enter array size:
5
Enter 5 data:
1 2 3 5 6
min = 1; max = 6
```

(2)

```
Enter array size:
1
Enter 1 data:
1
min = 1; max = 1
```

A sample program to test the functions is given below.

```
#include <stdio.h>
void findMinMax(int ar[], int size, int *min, int *max);
int main()
{
   int ar[40];
   int i, size;
   int min, max;

   printf("Enter array size: \n");
   scanf("%d", &size);
   printf("Enter %d data: \n", size);
   for (i=0; i<size; i++)
      scanf("%d", &ar[i]);
   findMinMax(ar, size, &min, &max);
   printf("min = %d; max = %d\n", min, max);
   return 0;
}
void findMinMax(int ar[], int size, int *min, int *max)
{
   /* Write your code here */
}
```

3. (**swap2RowsCols**) Write the code for the following functions:

```
void swap2Rows(int ar[SIZE][SIZE], int r1, int r2);
```
/* the function swaps the row r1 with the row r2 */

```
void swap2Cols(int ar[SIZE][SIZE], int c1, int c2);
```
/* the function swaps the column c1 with the column c2 */

Write a C program to test the above functions. In addition, your program should print the resultant matrix after each operation. You may assume that the input matrix is a 3x3 matrix when testing the functions.

Some sample input and output sessions are given below:
(1)

```
Enter the matrix (3x3) row by row:
5 10 15
15 20 25
25 30 35

Enter two rows for swapping:
1 2
The new array is:
5 10 15
```

```
25 30 35
15 20 25

Enter two columns for swapping:
1 2
The new array is:
5 15 10
25 35 30
15 25 20
```

(2)

```
Enter the matrix (3x3) row by row:
1 2 3
4 5 6
7 8 9

Enter two rows for swapping:
0 2
The new array is:
7 8 9
4 5 6
1 2 3

Enter two columns for swapping:
0 2
The new array is:
9 8 7
6 5 4
3 2 1
```

A sample program to test the functions is given below:

```c
#include <stdio.h>
#define SIZE 3
   void swap2Rows(int ar[SIZE][SIZE], int r1, int r2);
   void swap2Cols(int ar[SIZE][SIZE], int c1, int c2);
   void display(int ar[SIZE][SIZE]);

   int main()
   {
      int array[SIZE][SIZE];
      int row1, row2, col1, col2;
      int i,j;

      printf("Enter the matrix (3x3) row by row: \n");
      for (i=0; i<SIZE; i++)
         for (j=0; j<SIZE; j++)
            scanf("%d", &array[i][j]);

      printf("Enter two rows for swapping: \n");
      scanf("%d %d", &row1, &row2);
      swap2Rows(array, row1, row2);
      printf("The new array is: \n");
      display(array);

      printf("Enter two columns for swapping: \n");
      scanf("%d %d", &col1, &col2);
      swap2Cols(array, col1, col2);
      printf("The new array is: \n");
      display(array);
      return 0;
   }
   void display(int M[SIZE][SIZE])
   {
      int l,m;
      for (l = 0; l < 3; l++) {
         for (m = 0; m < 3; m++)
            printf("%d ", M[l][m]);
         printf("\n");
      }
      printf("\n");
   }
```

```
void swap2Rows(int M[SIZE][SIZE], int r1, int r2)
/* swaps row r1 with row r2 */
{
   /* Write your code here */
}
void swap2Cols(int M[SIZE][SIZE], int c1, int c2)
/* swaps column c1 with column c2 */
{
   /* Write your code here */
}
```

4. (**transpose**) Write a function **`transpose()`** that transposes a square matrix **`M`**. The function prototype is given below:

```
void transpose(int M[SIZE][SIZE]);
```

Write a C program to test the function. In addition, your program should print the resultant matrix after each operation. You may assume that the input matrix is a 3x3 matrix when testing the function.

Some sample input and output sessions are given below:

(1)

```
Enter the matrix (3x3) row by row:
5 10 15
15 20 25
25 30 35
transpose():
5 15 25
10 20 30
15 25 35
```

(2)

```
Enter the matrix (3x3) row by row:
-5 -6 -7
3 4 5
-1 -2 -3
transpose():
-5 3 -1
-6 4 -2
-7 5 -3
```

A sample program to test the function is given below.

```
#include <stdio.h>
#define SIZE 3
void transpose(int M[SIZE][SIZE]);
void display(int M[SIZE][SIZE]);
int main()
{
   int ar[SIZE][SIZE];
   int i,j;

   printf("Enter the matrix (3x3) row by row: \n");
   for (i=0; i<SIZE; i++)
      for (j=0; j<SIZE; j++)
         scanf("%d", &ar[i][j]);
   printf("transpose():\n");
   transpose(ar);
   display(ar);
   return 0;
}
void display(int M[SIZE][SIZE])
{
   int l,m;
   for (l = 0; l < 3; l++) {
      for (m = 0; m < 3; m++)
         printf("%d ", M[l][m]);
      printf("\n");
   }
   printf("\n");
```

```
}
void transpose(int M[SIZE][SIZE])
{
   /* Write your code here */
}
```

5. (**minOfMax**) Write a C function **minOfMax()** that takes a 4x4 two-dimensional array matrix of integer numbers as a parameter. The function returns the minimum of the maximum numbers of each row of a 2-dimensional array `a`. For example, if `ar` is `{{1,3,5,2}, {2,4,6,8}, {8,6,4,9}, {7,4,3,2}}`, then the maximum numbers will be 5, 8, 9 and 7 for rows 0, 1, 2 and 3 respectively, and the minimum of the maximum numbers will be 5. The prototype of the function is given as follows:

```
int minOfMax(int ar[4][4]);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)

```
Enter the matrix (4x4) row by row:
1 2 3 4
2 3 4 5
5 6 7 8
8 10 2 4
minOfMax(): 4
```

(2)

```
Enter the matrix (4x4) row by row:
1 -3 3 6
-3 2 4 6
3 6 -8 9
-3 -2 -3 -6
minOfMax(): -2
```

A sample program to test the function is given below.

```
#include <stdio.h>
int minOfMax(int ar[4][4]);
int main() {
   int ar[4][4],row,col,min;

   printf("Enter the matrix (4x4) row by row: \n");
   for (row=0; row<4; row++)
      for (col=0; col<4; col++)
         scanf("%d", &ar[row][col]);
   min=minOfMax(ar);
   printf("minOfMax(): %d\n", min);
   return 0;
}
int minOfMax(int ar[4][4])
{
   /* write your code here */
}
```

6. (**reduceMatrix**) A square matrix (2-dimensional array of equal dimensions) can be reduced to upper-triangular form by setting each diagonal element to the sum of the original elements in that column and setting to 0s all the elements below the diagonal. For example, the 4-by-4 matrix:

```
4 3 8 6
9 0 6 5
5 1 2 4
9 8 3 7
```

would thus be reduced to

```
27 3 8 6
 0 9 6 5
 0 0 5 4
 0 0 0 7
```

Write a function **reduceMatrix()** to reduce a 4-by-4 matrix. The prototype of the function is:

```
void reduceMatrix (int matrix[4][4]);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)

```
Enter the matrix (4x4) row by row:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
reduceMatrix():
28 2 3 4
0 30 7 8
0 0 26 12
0 0 0 16
```

(2)

```
Enter the matrix (4x4) row by row:
1 0 0 0
0 2 0 0
0 0 3 0
0 0 0 4
reduceMatrix():
1 0 0 0
0 2 0 0
0 0 3 0
0 0 0 4
```

A sample program to test the functions is given below.

```
#include <stdio.h>
void readMatrix(int M[4][4]);
void reduceMatrix (int matrix[4][4]);
void display(int M[4][4]);
int main()
{
   int A[4][4];
   readMatrix(A);
   reduceMatrix(A);
   printf("reduceMatrix(): \n");
   display(A);
   return 0;
}
void display(int M[4][4])
{
   int l,m;
   for (l = 0; l < 4; l++) {
      for (m = 0; m < 4; m++)
         printf("%d ", M[l][m]);
      printf("\n");
   }
   printf("\n");
}
void readMatrix(int M[4][4])
{
   int i, j;
   printf("Enter the matrix (4x4) row by row: \n");
   for (i=0; i<4; i++)
      for (j=0; j<4; j++)
         scanf("%d", &M[i][j]);
}
void reduceMatrix(int matrix[4][4])
{
   /* write your code here */
}
```

Section C – Character Strings [Answer 1 Specified Qn from this Section]

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)
```
Enter the string:
I am one of the 400 students in this class.
Total vowels = 11
Total digits = 3
```
(2)
```
Enter the string:
I am a boy.
Total vowels = 4
Total digits = 0
```
(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 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. However, if the two strings are the same, it will return the value 0. 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)
```
Enter the first string:
```

```
boy
Enter the second string:
girl
compareStr(): -5
```

(2)

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

(3)

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

(4)

```
Enter the first string:
abcd
Enter the second string:
abcd
compareStr(): 0
```

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. (**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)

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

(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 sourceStr[40] = "source";
   char 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 code here */
}
```

4. (**findTarget**) Write a C program that reads and searches character strings. In the program, it contains a 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.

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)

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

(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
int findTarget(char *target, char nameptr[SIZE][80], int size);
int main()
{
   char nameptr[SIZE][80];
   char t[40];
   int i, result, 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 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)
```
Enter size:
4
Enter 4 words:
Peter Paul John Mary
First word = John, Last word = Peter
```
(2)
```
Enter size:
1
Enter 1 words:
Peter
First word = Peter, Last word = Peter
```
(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 code here */
}
```

6. (**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)
```
Enter the string:
abcdef
Enter the substring:
dd
countSubstring(): 0
```

(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 code here */
}
```

Section D - Structures [Answer 1 Specified Qn from this Section]

1. **(circle)** A structure called circle is defined below. The structure consists of the radius of the circle and the (x,y) coordinates of its centre.

```
struct circle {
    double radius;
    double x;
    double y;
};
```

(a) Implement the function **intersect()** that returns 1 if two circles intersect, and 0 otherwise. Two circles intersect when the distance between their centres is less than or equal to the sum of their radii. The function prototype is given below:

```
int intersect(struct circle c1, struct circle c2);
```

(b) The function prototype of **contain()** is given below:

```
int contain(struct circle *c1, struct circle *c2);
```

The function contain() returns 1 if c1 contains c2, i.e. circle c2 is found inside circle c1. Otherwise, the function returns 0. Circle c1 contains circle c2 when the radius of c1 is larger than or equal to the sum of the radius of c2 and the distance between the centres of c1 and c2. Implement the function contain().

Write a C program to test the functions.

Some sample input and output sessions are given below:

(1)
```
Enter circle 1 (radius x y):
10 5 5
Enter circle 2 (radius x y):
5 1 1
intersect(): 1
contain(): 0
```

(2)
```
Enter circle 1 (radius x y):
10 5 5
Enter circle 2 (radius x y):
1 1 1
intersect(): 1
contain(): 1
```

(3)
```
Enter circle 1 (radius x y):
1 5 5
Enter circle 2 (radius x y):
1 10 10
intersect(): 0
contain(): 0
```

A sample template for the program is given below:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct circle {
   double radius;
   double x;
   double y;
};
int intersect(struct circle c1, struct circle c2);
int contain(struct circle *c1, struct circle *c2);
int main()
{
   struct circle c1, c2;

   printf("Enter circle 1 (radius x y): \n");
   scanf("%lf %lf %lf", &c1.radius, &c1.x, &c1.y);
   printf("Enter circle 2 (radius x y): \n");
   scanf("%lf %lf %lf", &c2.radius, &c2.x, &c2.y);
   printf("intersect(): %d\n", intersect(c1, c2));
   printf("contain(): %d\n", contain(&c1, &c2));
   return 0;
}
int intersect(struct circle c1, struct circle c2)
{
   /* Write your code here */
}
int contain(struct circle *c1, struct circle *c2)
{
   /* Write your code here */
}
```

2. **(compute)** A structure is defined to represent an arithmetic expression:

```
typedef struct {
    float operand1, operand2;
```

```
    char op;       /* operator '+','-','*' or '/'  */
} bexpression;
```

(a) Write a C function that computes the value of an expression and returns the result. For example, the function will return the value of 4/2 if in the structure passed to it, operand1 is 4, operator is '/' and operand2 is 2. The function prototype is given as:

```
float compute1(bexpression expr);
```

(b) Write another C function that performs the same computation with the following function prototype:

```
float compute2(bexpression *expr);
```

Write a C program to test the functions.

Some sample input and output sessions are given below:

(1)
```
Enter expression (op1 op2 op):
5 8 +
compute1(): 13.00
compute2(): 13.00
```

(2)
```
Enter expression (op1 op2 op):
8 5 /
compute1(): 1.60
compute2(): 1.60
```

(3)
```
Enter expression (op1 op2 op):
5 8 *
compute1(): 40.00
compute2(): 40.00
```

A sample template for the program is given below:

```
#include <stdio.h>
typedef struct {
   float operand1, operand2;
   char op;
} bexpression;
float compute1(bexpression expr);
float compute2(bexpression *expr);
int main()
{
   bexpression e;

   printf("Enter expression (op1 op2 op): \n");
   scanf("%f %f %c", &e.operand1, &e.operand2, &e.op);
   printf("compute1(): %.2f\n", compute1(e));
   printf("compute2(): %.2f\n", compute2(&e));
   return 0;
}
float compute1(bexpression expr)
{
   /* Write your code here */
}
float compute2(bexpression *expr)
{
   /* Write your code here */
}
```

3. (**encodeChar**) Write a function **`encodeChar()`** that accepts two character strings **`s`** and **`t`**, and an array of structures as parameters, encodes the characters in **`s`** to **`t`**, and passes the encoded string **`t`** to the caller via call by reference. During the encoding process, each `source` character is converted into the corresponding `code` character based on the following rules: `'a'→'d'; 'b'→'z'; 'z'→'a';` and `'d'→'b'`. For other source characters, the `code` will be the same as the `source`. For example, if the

character string **s** is "abort", then the encoded string **t** will be "dzort". The structure Rule is defined below:

```
typedef struct {
   char source;
   char code;
} Rule;
```

The function prototype is given below:

```
void encodeChar(Rule table[5], char *s, char *t);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)

```
Source string:
abort
Encoded string: dzort
```

(2)

```
Source string:
fgh
Encoded string: fgh
```

A sample template for the program is given below:

```
#include <stdio.h>
typedef struct {
   char source;
   char code;
} Rule;
void encodeChar(Rule table[5], char *s, char *t);
int main()
{
   char s[80],t[80];
   Rule table[5] = {'a','d', 'b','z', 'z','a', 'd','b', '\0','\0' };

   printf("Source string: \n");
   scanf("%s", s);
   encodeChar(table,s,t);
   printf("Encoded string: %s\n", t);
   return 0;
}
void encodeChar(Rule table[5], char *s, char *t)
{
   /* Write your code here */
}
```

4. (**customer**) Write a C program that repeatedly reads in customer data from the user and prints the customer data on the screen until the customer name ***"End Customer"*** (i.e., first_name last_name) is read. Your program should include the following two functions: the function **nextCustomer()** reads and returns a record for a single customer to the caller via a pointer parameter **acct**, and the function **printCustomer()** takes a parameter **acct** and then prints the customer information. The prototypes of the two functions are given below:

```
void  nextCustomer(struct account *acct);
void  printCustomer(struct account acct);
```

The structure definition for **struct account** is given below:

```
struct account {
   struct
   {
      char lastName[10];
      char firstName[10];
   } names;
```

```
   int accountNum;
   double balance;
};
```

You are required to implement the functions **`printCustomer()`** and **`nextCustomer()`**. Write a C program to test the function. In your code, you should follow the exact format of the required input and output given in the following test sample session.

Some sample input and output sessions are given below:

(1)

```
Enter names (firstName lastName):
SC Hui
Enter account number:
123
Enter balance:
6789.89
Customer record: SC Hui 123 6789.89
Enter names (firstName lastName):
End Customer
```

(2)

```
Enter names (firstName lastName):
SC Hui
Enter account number:
123
Enter balance:
6789.89
Customer record: SC Hui 123 6789.89
Enter names (firstName lastName):
FY Tan
Enter account number:
13
Enter balance:
69.89
Customer record: FY Tan 13 69.89
Enter names (firstName lastName):
End Customer
```

(3)

```
Enter names (firstName lastName):
End Customer
```

A sample program to test the functions is given below:

```
#include <stdio.h>
#include <string.h>
struct account {
   struct
   {
      char lastName[10];
      char firstName[10];
   } names;
   int accountNum;
   double balance;
};
void  nextCustomer(struct account *acct);
void  printCustomer(struct account acct);
int main()
{
   struct account record;
   int flag = 0;
   do {
      nextCustomer(&record);
      if ((strcmp(record.names.firstName, "End") == 0) &&
          (strcmp(record.names.lastName, "Customer") == 0))
         flag = 1;
      if (flag != 1)
         printCustomer(record);
   } while (flag != 1);
}
void nextCustomer(struct account *acct)
{
```

```
    /* write your code here */
}
void  printCustomer(struct account acct)
{
    /* write your code here */
}
```

5. (**phoneBook**) Write a C program that implements the following two functions. The function **`readin()`** reads a number of persons' names and their corresponding telephone numbers, passes the data to the caller via the parameter p, and returns the number of names that have entered. The character **`'#'`** is used to indicate the end of user input. The function **`search()`** finds the telephone number of an input name `target`, and then prints the name and telephone number on the screen. If the input name cannot be found, then it will print an appropriate error message. The prototypes of the two functions are given below:

```
int readin(PhoneBk *p);
void search(PhoneBk *p, int size, char *target);
```

The structure definition for `PhoneBk` is given below:

```
typedef struct {
   char name[20];
   char telno[20];
} PhoneBk;
```

You are required to implement the two functions. Write a C program to test the functions.

Some test input and output sessions are given below:

(1)

```
Enter name:
Hui Siu Cheung
Enter tel:
1234567
Enter name:
Philip Fu
Enter tel:
2345678
Enter name:
Chen Jing
Enter tel:
3456789
Enter name:
#
Enter search name:
Philip Fu
Name = Philip Fu, Tel = 2345678
```

(2)

```
Enter name:
Hui Siu Cheung
Enter tel:
1234567
Enter name:
Chen Jing
Enter tel:
3456789
Enter name:
#
Enter search name:
Philip Fu
Name not found!
```

(3)

```
Enter name:
#
Enter search name:
Philip Fu
Name not found!
```

A sample program to test the functions is given below:

```c
#include <stdio.h>
#include <string.h>
#define MAX 100
typedef struct {
   char name[20];
   char telno[20];
} PhoneBk;
int readin(PhoneBk *p);
void search(PhoneBk *p, int size, char *target);
int main()
{
   PhoneBk s[MAX];
   char t[20];
   int size;
   size = readin(s);
   printf("Enter search name: \n");
   gets(t);
   search(s,size,t);
   return 0;
}
int readin(PhoneBk *p)
{
   /* write your code here */
}
void search(PhoneBk *p, int size, char *target)
{
    /* write your code here */
}
```

6. (**mayTakeLeave**) Given the following information, write the code for the functions **`getInput()`**, **`mayTakeLeave()`** and **`printList()`**.

```
typedef struct {
      int id;                  /* staff identifier */
      int totalLeave;          /* the total number of days of leave allowed */
      int leaveTaken;          /* the number of days of leave taken so far */
} leaveRecord;
```

(a) void getInput(leaveRecord list[ ], int *n);

Each line of the input has three integers representing one staff identifier, his/her total number of days of leave allowed and his/her number of days of leave taken so far respectively. The function will read the data into the array **`list`** until end of input and returns the number of records read through **`n`** The function prototype is given as follows:

(b) int mayTakeLeave(leaveRecord list[ ], int id, int leave, int n);

It returns 1 if a leave application for **`leave`** days is approved. Staff member with identifier **`id`** is applying for **`leave`** days of leave. **`n`** is the number of staff in **`list`**. Approval will be given if the leave taken so far plus the number of days applied for is less than or equal to his total number of **`leave`** days allowed. If approval is not given, it returns 0. It will return -1 if no one in **`list`** has identifier **`id`**.

(c) void printList(leaveRecord list[ ], int n);

It prints the **`list`** of leave records of each staff. **`n`** is the number of staff in **`list`**.

Write a C program to test the functions. You do not need to check any errors in the input.

Some sample input and output sessions are given below:
(1)

```
Enter the number of staff records:
2
Enter id, totalleave, leavetaken:
```

```
11 28 25
Enter id, totalleave, leavetaken:
12 28 6
The staff list:
id = 11, totalleave = 28, leave taken = 25
id = 12, totalleave = 28, leave taken = 6
Please input id, leave to be taken:
11 6
The staff 11 cannot take leave
```

(2)

```
Enter the number of staff records:
2
Enter id, totalleave, leavetaken:
11 28 25
Enter id, totalleave, leavetaken:
12 28 6
The staff list:
id = 11, totalleave = 28, leave taken = 25
id = 12, totalleave = 28, leave taken = 6
Please input id, leave to be taken:
12 6
The staff 12 can take leave
```

(3)

```
Enter the number of staff records:
2
Enter id, totalleave, leavetaken:
11 28 25
Enter id, totalleave, leavetaken:
12 28 6
The staff list:
id = 11, totalleave = 28, leave taken = 25
id = 12, totalleave = 28, leave taken = 6
Please input id, leave to be taken:
13 6
The staff 13 is not in the list
```

A sample program to test the functions is given below:

```c
#include <stdio.h>
typedef struct {
   int id;              /* staff identifier */
   int totalLeave;      /* the total number of days of leave allowed */
   int leaveTaken;      /* the number of days of leave taken so far */
} leaveRecord;
int mayTakeLeave(leaveRecord list[], int id, int leave, int n);
void getInput(leaveRecord list[], int *n);
void printList(leaveRecord list[], int n);
int main()
{
   leaveRecord listRec[10];
   int len;
   int id, leave, canTake=-1;

   getInput(listRec, &len);
   printList(listRec, len);
   printf("Please input id, leave to be taken: \n");
   scanf("%d %d", &id, &leave);
   canTake = mayTakeLeave(listRec, id, leave, len);
   if (canTake == 1)
      printf("The staff %d can take leave\n", id);
   else if (canTake == 0)
      printf("The staff %d cannot take leave\n", id);
   else
      printf("The staff %d is not in the list\n", id);
   return 0;
}
void getInput(leaveRecord list[], int *n)
{
    /* write your code here */
}
int mayTakeLeave(leaveRecord list[], int id, int leave, int n)
```

```
{
    /* write your code here */
}
void printList(leaveRecord list[], int n)
{
   int p;

   printf("The staff list:\n");
   for (p = 0; p < n; p++)
      printf ("id = %d, totalleave = %d, leave taken = %d\n",
         list[p].id, list[p].totalLeave, list[p].leaveTaken);
}
```

Section E – Recursive Funcitions [Ans 1 Specified Qn from this Section]

1. **(rDigitValue1)** Write a recursive function that returns the value of the k^{th} digit (k>0) from the right of a non-negative integer *num*. For example, if num=1234567 and k=3, the function will return 5 and if num=1234 and k=8, the function will return 0. The recrusive function **rdigitValue1()** returns the result. The prototype of the function is given below:

   ```
   int rDigitValue1(int num, int k);
   ```

   Write a C program to test the functions.

   Some sample input and output sessions are given below:
   (1)
   ```
   Enter the number:
   1234567
   Enter k position:
   3
   rDigitValue1(): 5
   ```
   (2)
   ```
   Enter the number:
   123
   Enter k position:
   8
   rDigitValue1(): 0
   ```

   A sample program to test the functions is given below:

   ```
   #include <stdio.h>
   int rDigitValue1(int num, int k);
   int main()
   {
      int k;
      int number;

      printf("Enter the number: \n");
      scanf("%d", &number);
      printf("Enter the position: \n");
      scanf("%d", &k);
      printf("rDigitValue1(): %d\n", rDigitValue1(number, k));
      return 0;
   }
   int rDigitValue1(int num, int k)
   {
      /* Write your code here */
   }
   ```

2. **(rSquare2)** Write a recursive function that returns the square of a positive integer number *num*, by computing the sum of odd integers starting with 1. The result is returned to the calling function. For example, if *num* = 4, then $4^2 = 1 + 3 + 5 + 7 = 16$ is returned; if *num* = 5, then $5^2 = 1 + 3 + 5 + 7 + 9 = 25$ is returned. The recursive function **rSquare2()** returns the result through the parameter *result*. The function prototype is:

```
void rSquare2(int num, int *result);
```

Some sample input and output sessions are given below:

(1)

```
Enter the number:
4
rSquare2(): 16
```

(2)

```
Enter the number:
1
rSquare2(): 1
```

A sample program to test the functions is given below:

```
#include <stdio.h>
void rSquare2(int num, int *result);
int main()
{
   int x, result;

   printf("Enter the number: \n");
   scanf("%d", &x);
   rSquare2(x, &result);
   printf("rSquare2(): %d\n", result);
   return 0;
}
void rSquare2(int num, int *result)
{
   /* Write your code here */
}
```

3. (**rCountZeros1**) Write a recursive C function that counts the number of zeros in a specified positive number num. For example, if num is 105006, then the function will return 3; and if num is 1357, the function will return 0. The recursive function **rCountZeros1()** returns the result. The function prototype is given as follows:

```
int rCountZeros1(int num);
```

Write a C program to test the functions.

Some sample input and output sessions are given below:

(1)

```
Enter the number:
10500
rCountZeros1(): 3
```

(2)

```
Enter the number:
23453
rCountZeros1(): 0
```

(3)

```
Enter the number:
0
rCountZeros1(): 1
```

A sample program to test the functions is given below:

```
#include <stdio.h>
int rCountZeros1(int num);
int main()
{
   int number;

   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("rCountZeros1(): %d\n", rCountZeros1(number));
   return 0;
}
int rCountZeros1(int num)
```

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

4. **(rStrLen)** The **recursive** function rStrLen() accepts a character string s as parameter, and returns the length of the string. For example, if s is "abcde", then the function rStrLen() will return 5. The function prototype is:

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

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)
```
Enter the string:
abcde
rStrLen(): 5
```
(2)
```
Enter the string:
a
rStrLen(): 1
```

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

```
#include <stdio.h>
int rStrLen(char *s);
int main() {
   char str[80];
   printf("Enter the string: \n");
   gets(str);
   printf("rStrLen(): %d\n", rStrLen(str));
   return 0;
}
int rStrLen(char *s)
{
   /* Write your code here */
}
```

5. **(rReverseAr)** Write a recursive function whose arguments are an array of integers and an integer specifying the size of the array and whose task is to reverse the contents of the array. The result is returned to the caller through the array parameter. The code should not use another, intermediate, array. The function prototype is given as follows:

```
void rReverseAr(int ar[], int size);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)
```
Enter size:
5
Enter 5 numbers:
1 2 3 4 5
rReverseAr(): 5 4 3 2 1
```
(2)
```
Enter size:
1
Enter 1 numbers:
3
rReverseAr(): 3
```

A sample program to test the function is given below:

```
#include <stdio.h>
void rReverseAr(int ar[], int size);
int main()
```

```
{
   int array[80];
   int size, i;

   printf("Enter size: \n", &size);
   scanf("%d", &size);
   printf("Enter %d numbers: \n", size);
   for (i = 0; i < size; i++)
      scanf("%d", &array[i]);
   printf("rReverseAr(): ");
   rReverseAr(array, size);
   for (i = 0; i < size; i++)
      printf("%d ", array[i]);
   printf("\n");
   return 0;
}
void rReverseAr(int ar[], int size)
{
   /* Write your code here */
}
```

6. (**rCountArray**) Write a **recursive** C function **`rCountArray()`** that returns the number of times the integer **a** appears in the array which has **n** integers in it. Assume that **n** is greater than or equal to 1. The function prototype is:

```
int rCountArray(int array[], int n, int a);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1)

```
Enter array size:
10
Enter 10 numbers:
1 2 3 4 5 5 6 7 8 9
Enter the target:
5
rCountArray(): 2
```

(2)

```
Enter array size:
5
Enter 5 numbers:
1 2 3 4 5
Enter the target:
8
rCountArray(): 0
```

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

```
#include <stdio.h>
#define SIZE 20
int rCountArray(int array[], int n, int a);
int main()
{
   int array[SIZE];
   int index, count, target, size;

   printf("Enter array size: \n");
   scanf("%d", &size);
   printf("Enter %d numbers: \n", size);
   for (index = 0; index < size; index++)
      scanf("%d", &array[index]);
   printf("Enter the target: \n");
   scanf("%d", &target);
   count = rCountArray(array, size, target);
   printf("rCountArray(): %d\n", count);
   return 0;
}
int rCountArray(int array[], int n, int a)
```

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