WEEK 7 - RECURSIVE FUNCTIONS

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

Questions 1-3

You may use the program template in Figure 1 to test your recursive functions developed in this lab. The program contains a `main()` which includes a switch statement so that the following functions can be tested by the user. Write the code for each function and use the suggested test cases to test your code for correctness.

```
#include <stdio.h>

/* function prototypes */
int rNumDigits1(int num);
void rNumDigits2(int num, int *result);
int rDigitPos1(int num, int digit);
void rDigitPos2(int num, int digit, int *pos);
int rSquare1(int num);
void rSquare2(int num, int *result);

int main()
{
   int choice;
   int number;
   int digit, result=0;

   do {
      printf("\nPerform the following functions ITERATIVELY:\n");
      printf("1:  rNumDigits1()\n");
      printf("2:  rNumDigits2()\n");
      printf("3:  rDigitPos1()\n");
      printf("4:  rDigitPos2()\n");
      printf("5:  rSquare1()\n");
      printf("6:  rSquare2()\n");
      printf("7:  quit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter the number: \n");
            scanf("%d", &number);
            printf("rNumDigits1(): %d\n", rNumDigits1(number));
            break;
         case 2:
            printf("Enter the number: \n");
            scanf("%d", &number);
            rNumDigits2(number, &result);
            printf("rNumDigits2(): %d\n", result);
```

```
            break;
         case 3:
            printf("Enter the number: \n");
            scanf("%d", &number);
            printf("Enter the digit: \n");
            scanf("%d", &digit);
            printf("rDigitPos1(): %d\n", rDigitPos1(number,
digit));
            break;
         case 4:
            printf("Enter the number: \n");
            scanf("%d", &number);
            printf("Enter the digit: \n");
            scanf("%d", &digit);
            rDigitPos2(number, digit, &result);
            printf("rDigitPos2(): %d\n", result);
            break;
         case 5:
            printf("Enter the number: \n");
            scanf("%d", &number);
            printf("rSquare1(): %d\n", rSquare1(number));
            break;
         case 6:
            printf("Enter the number: \n");
            scanf("%d", &number);
            rSquare2(number, &result);
            printf("rSquare2(): %d\n", result);
            break;
         default: printf("Program terminating .....\n");
            break;
      }
   } while (choice < 7);
   return 0;
}
int rNumDigits1(int num)
{
   if (num < 10)
      return 1;
   else
      return rNumDigits1(num/10) + 1;
}
void rNumDigits2(int num, int *result)
{
   /* Write your program code here */
}
int rDigitPos1(int num, int digit)
{
   /* Write your program code here */
}
void rDigitPos2(int num, int digit, int *pos)
{
   if (num % 10 == digit)
      *pos = 1;
   else if (num < 10)
      *pos = 0;
   else {
      rDigitPos2(num/10, digit, pos);
      if (*pos > 0)
         *pos += 1;
      else
         *pos = 0;
   }
}
int rSquare1(int num)
```

```
{
   /* Write your program code here */
}
void rSquare2(int num, int *result)
{
   /* Write your program code here */
}
```

Figure 1

1. (**rNumDigits**) Write a **recursive** function that counts the number of digits for a non-negative integer. For example, 1234 has 4 digits. Write two versions of the function. The function `rNumDigits1()` returns the result. The function `rNumDigits2()` returns the result through the parameter *`result`*. The function prototypes are:

```
int rNumDigits1(int num);
void rNumDigits2(int num, int *result);
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter the number:
5
rNumDigits1(): 1
rNumDigits2(): 1
```

(2) Test Case 2:
```
Enter the number:
13579
rNumDigits1(): 5
rNumDigits2(): 5
```

(3) Test Case 3:
```
Enter the number:
12
rNumDigits1(): 2
rNumDigits2(): 2
```

(4) Test Case 4:
```
Enter the number:
2468
rNumDigits1(): 4
rNumDigits2(): 4
```

For separate program testing: The following sample program template is given below:

```
#include <stdio.h>
int rNumDigits1(int num);
void rNumDigits2(int num, int *result);
int main()
{
   int number, result=0;

   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("rNumDigits1(): %d\n", rNumDigits1(number));
   rNumDigits2(number, &result);
   printf("rNumDigits2(): %d\n", result);
   return 0;
}
int rNumDigits1(int num)
{
   /* Write your program code here */
```

```
}
void rNumDigits2(int num, int *result)
{
   /* Write your program code here */
}
```

2. (**rDigitPos**) Write a **recursive** function 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. Write two versions of the function. The function `rDigitPos1()` returns the result. The function `rDigitPos2()` returns the result through the pointer parameter *pos*. The function prototypes are:

```
int rDigitPos1(int num, int digit);
void rDigitPos2(int num, int digit, int *pos);
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter the number:
1234567
Enter the digit:
6
rDigitPos1(): 2
rDigitPos2(): 2
```

(2) Test Case 2:
```
Enter the number:
1234567
Enter the digit:
8
rDigitPos1(): 0
rDigitPos2(): 0
```

(3) Test Case 3:
```
Enter the number:
1357
Enter the digit:
3
rDigitPos1(): 3
rDigitPos2(): 3
```

(4) Test Case 4:
```
Enter the number:
6
Enter the digit:
6
rDigitPos1(): 1
rDigitPos2(): 1
```

For separate program testing: The following sample program template is given below:

```
#include <stdio.h>
int rDigitPos1(int num, int digit);
void rDigitPos2(int num, int digit, int *pos);
int main()
{
   int number, digit, result=0;

   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("Enter the digit: \n");
   scanf("%d", &digit);
   printf("rDigitPos1(): %d\n", rDigitPos1(number, digit));
```

```
    rDigitPos2(number, digit, &result);
    printf("rDigitPos2(): %d\n", result);
    return 0;
}
int rDigitPos1(int num, int digit)
{
    /* Write your program code here */
}
void rDigitPos2(int num, int digit, int *pos)
{
    /* Write your program code here */
}
```

3. (**rSquare**) 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. Write two versions of the function. The function `rSquare1()` returns the result. The function `rSquare2()` returns the result through the parameter *result*. The function prototypes are:

```
int rSquare1(int num);
void rSquare2(int num, int *result);
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter a number:
4
rSquare1(): 16
rSquare2(): 16
```

(2) Test Case 2:
```
Enter a number:
1
rSquare1(): 1
rSquare2(): 1
```

(3) Test Case 3:
```
Enter a number:
12
rSquare1(): 144
rSquare2(): 144
```

(4) Test Case 4:
```
Enter a number:
5
rSquare1(): 25
rSquare2(): 25
```

For separate program testing: The following sample program template is given below:

```
#include <stdio.h>
int rSquare1(int num);
void rSquare2(int num, int *result);
int main()
{
    int number, result=0;

    printf("Enter the number: \n");
    scanf("%d", &number);
    printf("rSquare1(): %d\n", rSquare1(number));
    rSquare2(number, &result);
    printf("rSquare2(): %d\n", result);
```

```
    return 0;
}
int rSquare1(int num)
{
    /* Write your program code here */
}
void rSquare2(int num, int *result)
{
    /* Write your program code here */
}
```

4. (**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) Test Case 1:

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

(2) Test Case 2:

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

(3) Test Case 3:

```
Enter array size:
1
Enter 1 numbers:
5
Enter the target number:
5
rCountArray(): 1
```

(4) Test Case 4:

```
Enter array size:
7
Enter 5 numbers:
1 2 3 3 4 3 3
Enter the target number:
3
rCountArray(): 4
```

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 number: \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 program code here */
}
```

Section B – Recursion

1. (**rAge**) Assume that the youngest student is 10 years old. The age of the next older student can be computed by adding 2 years to the age of the previous younger student. The students are arranged in ascending order according to their age with the youngest student as the first one. Write a **recursive** function `rAge()` that takes in the rank of a student *`studRank`* and returns the age of the student to the calling function. For example, if *`studRank`* is `4`, then the age of the corresponding student `16` will be returned. The function prototype is given as follows:

```
int rAge(int studRank);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter student rank:
5
rAge(): 18
```

(2) Test Case 2:
```
Enter student rank:
1
rAge(): 10
```

A sample program to test the function is given below:

```
#include <stdio.h>
int rAge(int studRank);
int main()
{
   int studRank;

   printf("Enter student rank: \n");
   scanf("%d",&studRank);
   printf("rAge(): %d\n", rAge(studRank));
   return 0;
}
int rAge(int studRank)
{
   /* Write your code here */
}
```

2. (**rPower1**) Write a **recursive** function that computes the power of a number *`num`*. The power *`p`* may be any integer value. The function `rPower1()` returns the computed result. The function prototype is given as follows:

```
float rPower1(float num, int p);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter the number and power:
2 3
rPower1(): 8.00
```

(2) Test Case 2:
```
Enter the number and power:
```

```
2 -4
rPower1(): 0.06
```

(3) Test Case 3:

```
Enter the number and power:
2 0
rPower1(): 1.00
```

A sample program to test the function is given below:

```
#include <stdio.h>
float rPower1(float num, int p);
int main()
{
   int power;
   float number;

   printf("Enter the number and power: \n");
   scanf("%f %d", &number, &power);
   printf("rPower1(): %.2f\n", rPower1(number, power));
   return 0;
}
float rPower1(float num, int p)
{
   /* Write your code here */
}
```

3. (**rCountZeros2**) 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 function `rCountZeros2()` passes the result through the pointer parameter *result*. The function prototype is given as follows:

```
void rCountZeros2(int num, int *result);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter the number:
10500
rCountZeros2(): 3
```

(2) Test Case 2:

```
Enter the number:
23453
rCountZeros2(): 0
```

(3) Test Case 3:

```
Enter the number:
0
rCountZeros2(): 1
```

A sample program to test the functions is given below:

```
#include <stdio.h>
void rCountZeros2(int num, int *result);
int main()
{
   int number, result;

   printf("Enter the number: \n");
```

```
    scanf("%d", &number);
    rCountZeros2(number,&result);
    printf("rCountZeros2(): %d\n", result);
    return 0;
}
void rCountZeros2(int num, int *result)
{
    /* Write your program code here */
}
```

4. (**rAllOddDigits1**) The **recursive** function `rAllOddDigits1()` returns either 1 or 0 according to whether or not all the digits of the positive integer argument number *num* are odd. For example, if the argument *num* is 1357, then the function should return 1; and if the argument *num* is 1234, then 0 should be returned. The function prototype is given below:

```
int rAllOddDigits1(int num);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter a number:
3579
rAllOddDigits1(): 1
```

(2) Test Case 2:
```
Enter a number:
3578
rAllOddDigits1(): 0
```

A sample program to test the function is given below:

```
#include <stdio.h>
int rAllOddDigits1(int num);
int main()
{
    int number;

    printf("Enter a number: \n");
    scanf("%d", &number);
    printf("rAllOddDigits1(): %d\n", rAllOddDigits1(number));
    return 0;
}
int rAllOddDigits1(int num)
{
    /* Write your code here */
}
```

5. (**rReverseDigits**) Write a **recursive** C function `rReverseDigits()` that takes an non-negative integer argument *num* and returns an integer whose digits are obtained by reversing those of the argument number. The result is passed to the calling function through a pointer variable *result*. For example, if *num* is 1234, then the function should return 4321 through the pointer variable. If *num* is 10, then the function should return 1. The function prototype is given below:

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

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter a number:
1234
rReverseDigits(): 4321
```

(2) Test Case 2:

```
Enter a number:
10
rReverseDigits(): 1
```

(3) Test Case 1:

```
Enter a number:
12934
rReverseDigits(): 43921
```

A sample program to test the function is given below:

```
#include <stdio.h>
void rReverseDigits(int num, int *result);
int main()
{
   int result=0, number;

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

6. **(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) Test Case 1:

```
Enter the string:
abcde
rStrLen(): 5
```

(2) Test Case 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 program code here */
}
```

7. (**rStrcmp**) The **recursive** C function `rStrcmp()` 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 string library functions. The function prototype is given as follows:

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

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter a source string:
abc
Enter a target string:
abc
rStrcmp(): 0
```

(2) Test Case 2:

```
Enter a source string:
abcdef
Enter a target string:
abc123
rStrcmp(): 1
```

(3) Test Case 3:

```
Enter a source string:
abc123
Enter a target string:
abcdef
rStrcmp(): -1
```

(4) Test Case 4:

```
Enter a source string:
abc123
Enter a target string:
abc123f
rStrcmp(): -1
```

A sample program to test the function is given below:

```
#include <stdio.h>
#define INIT_VALUE 10
int rStrcmp(char *s1, char *s2);
int main()
{
    char source[40], target[40];
    int result=INIT_VALUE;

    printf("Enter a source string: \n");
    gets(source);
    printf("Enter a target string: \n");
    gets(target);
    result = rStrcmp(source, target);
```

```
    printf("rStrcmp(): %d", result);
    return 0;
}
int rStrcmp(char *s1, char *s2)
{
    /* Write your code here */
}
```

8. (**rFindMaxAr**) Write a **recursive** C function `rFindMaxAr()` that finds the index position of the maximum number in an array of integer numbers. In the function, the parameter *ar* accepts an array passed in from the calling function. The pointer parameter *index* is used for passing the maximum number's index position to the caller via call by reference. The function prototype is given as follows:

```
void rFindMaxAr(int *ar, int size, int i, int *index);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter array size:
5
Enter 5 numbers:
1 2 3 4 5
Max number: 5
Index position: 4
```

(2) Test Case 2:

```
Enter array size:
7
Enter 7 numbers:
2 5 4 7 9 10 1
Max number: 10
Index position: 5
```

A sample program to test the functions is given below:

```
#include <stdio.h>
void rFindMaxAr(int *ar, int size, int i, int *index);
int main()
{
    int ar[50],i,maxIndex=0,size;

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

9. (**rReverseAr**) Write a **recursive** function whose arguments are an array of integers *ar* and an integer *size* 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) Test Case 1:

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

(2) Test Case 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");
   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 program code here */
}
```