Section G – Recursion [Ans 2 Specified Qns from this Section]

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. (**rDigitValue2**) 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. Write the function `rDigitValue2()` that passes the result through the parameter *`result`*. The prototype of the function is given below:

   ```
   void rDigitValue2(int num, int k, 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:
   234567
   Enter k position:
   3
   rDigitValue2(): 5
   ```

(2) Test Case 2:

```
Enter a number:
123
Enter k position:
8
rDigitValue2(): 0
```

A sample program to test the function is given below:

```
#include <stdio.h>
void rDigitValue2(int num, int k, int *result);
int main()
{
   int k;
   int number, digit;

   printf("Enter a number: \n");
   scanf("%d", &number);
   printf("Enter k position: \n");
   scanf("%d", &k);
   rDigitValue2(number, k, &digit);
   printf("rDigitValue2(): %d\n", digit);
   return 0;
}
void rDigitValue2(int num, int k, int *result)
{
   /* Write your code here */
}
```

3. (**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 */
}
```

4. (**rGcd1**) Write a **recursive** C function `rGcd1()` that computes the greatest common divisor and returns the result to the calling function. For example, `rGcd1(4,7)` returns 1, `rGcd1(4,32)` returns 4 and `rGcd1(4,38)` returns 2. The function prototype is given as follows:

```
int rGcd(int num1, int num2);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter 2 numbers:
4 7
rGcd1(): 1
```

(2) Test Case 1:

```
Enter 2 numbers:
32 4
rGcd1(): 4
```

(3) Test Case 2:

```
Enter 2 numbers:
4 38
rGcd1(): 2
```

A sample program to test the function is given below:

```
#include <stdio.h>
int rGcd1(int num1, int num2);
int main()
{
   int n1, n2;

   printf("Enter 2 numbers: \n");
   scanf("%d %d", &n1, &n2);
   printf("rGcd1(): %d\n", rGcd1(n1, n2));
   return 0;
}
int rGcd1(int num1, int num2)
{
   /* Write your code here */
}
```

5. (**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 */
}
```

6. (**rPrintArReverse**) Write a **recursive** C function `rPrintArReverse()` that prints the content of an array of integers in reverse order. For example, if *ar*`[]` is {1, 2, 3, 4, 5} and *size* is 5, then 5, 4, 3, 2, 1 will be printed when the function `rPrintArReverse(ar,5)` is called. The function prototype is given as follows:

```
void rPrintArReverse(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
rPrintArReverse(): 5 4 3 2 1
```

(2) Test Case 2

```
Enter size:
1
Enter 1 numbers:
5
rPrintArReverse(): 5
```

A sample program to test the function is given below:

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

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

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

7. (**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 */
}
```

8. (**rLookupAr**) Write a **recursive** C function called `rLookupAr()` takes in three parameters, *`array`*, *`size`* and *`target`*, and returns the subscript of the **last appearance** of a number in the array. The parameter *`size`* indicates the size of the array. For example, if *`array`* is {2,1,3,2,4} and *`target`* is 3, it will return 2. With the same array, if *`target`* is 2, it will return 3. If the required number is not in the array, the function will return –1. The function prototype is given below:

   ```
   int rLookupAr(int array[], int size, int target);
   ```

   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:
   2 1 3 2 4
   Enter the target number:
   2
   rLookupAr(): 3
   ```

   (2) Test Case 2:

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

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

   ```
   #include <stdio.h>
   int rLookupAr(int array[], int size, int target);
   int main()
   {
      int numArray[80];
      int target, i, size;

      printf("Enter array size: \n");
      scanf("%d", &size);
      printf("Enter %d numbers: \n", size);
      for (i=0; i < size; i++)
         scanf("%d", &numArray[i]);
      printf("Enter the target number: \n");
      scanf("%d", &target);
      printf("rLookupAr(): %d", rLookupAr(numArray, size, target));
      return 0;
   }
   int rLookupAr(int array[], int size, int target)
   {

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

9. (**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
```

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 */
}
```

10. (**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:

```c
#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 */
}
```