Week 6 Practice Questions – Recursive Functions

1. (**rSumup**) A function **rSumup()** is defined as:

   rSumup(1) = 1
   rSumup(n) = n + rSumup(n-1)   if n > 1

   Implement `rSumup()` in two versions. The function `rSumup1()` computes and returns the result. The function `rSumup2()` computes and returns the result through the parameter `result`. The function prototypes are given as follows:

```
int rSumup1(int n);
void rSumup2(int n, int *result);
```

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

```
#include <stdio.h>
int rSumup1(int n);
void rSumup2(int n, int *result);
int main()
{
   int n, result;

   printf("Enter a number: \n");
   scanf("%d", &n);
   printf("rSumup1(): %d\n", rSumup1(n));
   rSumup2(n, &result);
   printf("rSumup2(): %d\n",result);
   return 0;
}
int rSumup1(int n)
{
   /* Write your code here */
}
void rSumup2(int n, int *result)
{
   /* Write your code here */
}
```

   Some sample input and output sessions are given below:

   (1) Test Case 1:
```
Enter a number:
5
rSumup1(): 15
rSumup2(): 15
```

   (2) Test Case 2:
```
Enter a number:
10
rSumup1(): 55
rSumup2(): 55
```

2. (**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);
```

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

```
#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 */
}
```

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
```

3. (**rGcd**) Write a **recursive** C function that computes the greatest common divisor and returns the result to the calling function via call by reference. For example, if `num1` is 4 and `num2` is 7, then `result` is 1; and if `num1` is 4 and `num2` is 32, then `result` is 4. Write the recursive function in two versions. The function `rGcd1()` computes and returns the result. The function `rGcd2()` computes and returns the result through the parameter `result` using call by reference. The function prototypes are given as follows:

```
int rGcd1(int num1, int num2);
void rGcd2(int num1, int num2, int *result);
```

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

```
#include <stdio.h>
int rGcd1(int num1, int num2);
void rGcd2(int num1, int num2, int *result);
int main()
{
   int n1, n2, result;

   printf("Enter 2 numbers: \n");
   scanf("%d %d", &n1, &n2);
   printf("rGcd1(): %d\n", rGcd1(n1, n2));
   rGcd2(n1, n2, &result);
   printf("rGcd2(): %d\n", result);
   return 0;
}
int rGcd1(int num1, int num2)
{
   /* Write your code here */
}
```

```
void rGcd2(int num1, int num2, int *result)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter 2 numbers:
4 7
rGcd1(): 1
rGcd2(): 1
```

(2) Test Case 2:
```
Enter 2 numbers:
32 4
rGcd1(): 4
rGcd2(): 4
```

(3) Test Case 3:
```
Enter 2 numbers:
4 38
rGcd1(): 2
rGcd2(): 2
```

(4) Test Case 4:
```
Enter 2 numbers:
32 38
rGcd1(): 2
rGcd2(): 2
```

4. (**rDigitValue**) 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` is12348567 and `k` is 3, the function will return 5 and if `num` is 1234 and `k` is 8, the function will return 0. Write the recursive function in two versions. The function `rDigitValue1()` computes and returns the result. The function `rDigitValue2()` computes and returns the result through the parameter `result` using call by reference. The function prototypes are given below:

```
int rDigitValue1(int num, int k);
void rDigitValue2(int num, int k, int *result);
```

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

```
#include <stdio.h>
int rDigitValue1(int num, int k);
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);
   printf("rDigitValue1(): %d\n", rDigitValue1(number, k));
   rDigitValue2(number, k, &digit);
   printf("rDigitValue2(): %d\n", digit);
   return 0;
}
int rDigitValue1(int num, int k)
{
   /* Write your code here */
}
void rDigitValue2(int num, int k, int *result)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter a number:
2348567
Enter k position:
3
rDigitValue1(): 5
rDigitValue2(): 5
```

(2) Test Case 2:
```
Enter a number:
123
Enter k position:
4
rDigitValue1(): 0
rDigitValue2(): 0
```

(3) Test Case 3:
```
Enter a number:
12456
Enter k position:
1
rDigitValue1(): 6
rDigitValue2(): 6
```

(4) Test Case 4:
```
Enter a number:
82345
Enter k position:
5
rDigitValue1(): 8
rDigitValue2(): 8
```

5. (**rPower**) Write a **recursive** function that computes the power of a number `num`. The power *p* may be any integer value. Write the recursive function in two versions. The function `rPower1()` computes and returns the result. The function `rPower2()` computes and returns the result through the parameter `result` using call by reference. The function prototypes are given as follows:

```
float rPower1(float num, int p);
void  rPower2(float num, int p, float *result);
```

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

```
#include <stdio.h>
float rPower1(float num, int p);
void  rPower2(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("rPower1(): %.2f\n", rPower1(number, power));
   rPower2(number, power, &result);
   printf("rPower2(): %.2f\n", result);
   return 0;
}
float rPower1(float num, int p)
{
   /* Write your code here */
}
void rPower2(float num, int p, float *result)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter the number and power:
2 3
rPower1(): 8.00
rPower2(): 8.00
```

(2) Test Case 2:
```
Enter the number and power:
2 -4
rPower1(): 0.06
rPower2(): 0.06
```

(3) Test Case 3:
```
Enter the number and power:
2 0
rPower1(): 1.00
rPower2(): 1.00
```

6. (**rAllOddDigits**) The **recursive** function that 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. Write the recursive function in two versions. The function `rAllOddDigits1()` computes and returns the result. The function `rAllOddDigits2()` computes and returns the result through the parameter `result` using call by reference. The function prototypes are given below:

```
int rAllOddDigits1(int num);
void rAllOddDigits2(int num, int *result);
```

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

```
#include <stdio.h>
int rAllOddDigits1(int num);
void rAllOddDigits2(int num, int *result);
int main()
{
   int number, result=-1;

   printf("Enter a number: \n");
   scanf("%d", &number);
   printf("rAllOddDigits1(): %d\n", rAllOddDigits1(number));
   rAllOddDigits2(number, &result);
   printf("rAllOddDigits2(): %d\n", result);
   return 0;
}

int rAllOddDigits1(int num)
{
   /* Write your code here */
}
void rAllOddDigits2(int num, int *result)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter a number:
3579
rAllOddDigits1(): 1
rAllOddDigits2(): 1
```

(2) Test Case 2:
```
Enter a number:
3578
rAllOddDigits1(): 0
rAllOddDigits2(): 0
```

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

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

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

```
#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 */
}
```

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:
abc de
rStrLen(): 6
```

(3) Test Case 2:
```
Enter the string:
a
rStrLen(): 1
```

8. (**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 function prototype is given as follows:

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

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

```
#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 */
}
```

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
```

9. (**rLookupAr**) Write a **recursive** C function that 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);
```

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

```
#include <stdio.h>
int rLookupAr(int array[], int size, int target);
int main()
{
   int numArray[80];
   int target, i, size;
   int result=-999;

   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);
   result = rLookupAr(numArray, size, target);
   printf("rLookupAr(): %d", result);
   return 0;
}
int rLookupAr(int array[], int size, int target)
{
   /* Write your code here */
}
```

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
```

(3) Test Case 3:
```
Enter array size:
7
Enter 7 numbers:
7 9 10 1 2 3 4
Enter the target number:
3
rLookupAr(): 5
```

(4) Test Case 4:
```
Enter array size:
10
Enter 10 numbers:
7 9 1 1 2 3 4 1 2 3
Enter the target number:
1
rLookupAr(): 7
```