

WEEK 7 ASSIGNMENT – RECURSION

Assignment on Coding Questions

For this assignment – you will need to answer 4 questions (randomly generated via APAS) from the following question list:

1. rAge
2. rGcd
3. rPower
4. rCountZeros
5. rCountEvenDigits
6. rAllEvenDigits
7. rStrLen
8. rStrcmp
9. rFindMaxAr
10. rLookupAr
11. rReverseAr

Questions

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);
```

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

2. (**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

3. (**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

4. (**rCountZeros**) Write a **recursive** C function that counts the number of zeros in a specified positive number (bigger than 0) *num*. For example, if *num* is 105006, then the function will return 3; and if *num* is 1357, the function will return 0. Write the recursive function in two versions. The function `rCountZeros1()` computes and returns the result. The function `rCountZeros2()` passes the result through the pointer parameter *result*. The function prototypes are given as follows:

```
int rCountZeros1(int num);
void rCountZeros2(int num, int *result);
```

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

```
#include <stdio.h>
int rCountZeros1(int num);
void rCountZeros2(int num, int *result);
int main()
{
    int number, result;

    printf("Enter the number: \n");
    scanf("%d", &number);
    printf("rCountZeros1(): %d\n", rCountZeros1(number));
    rCountZeros2(number, &result);
    printf("rCountZeros2(): %d\n", result);
    return 0;
}
int rCountZeros1(int num)
{
    /* Write your program code here */
}
void rCountZeros2(int num, int *result)
{
    /* Write your program code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
Enter the number:
10500
rCountZeros1(): 3
rCountZeros2(): 3

(2) Test Case 2:
Enter the number:
23453
rCountZeros1(): 0
rCountZeros2(): 0

(3) Test Case 3:
Enter the number:
404
rCountZeros1(): 1
rCountZeros2(): 1

5. (**rCountEvenDigits**) Write a recursive C function that counts the number of even digits in a specified positive number (bigger than 0), *num*. For example, if *num* is 105006, then the function will return 4; and if *num* is 1357, the function will return 0. Write the recursive function in two versions. The function **rCountEvenDigits1()** computes and returns the result. The function **rCountEvenDigits2()** passes the result through the pointer parameter, *result*. The function prototypes are given as follows:

```
int rCountEvenDigits1(int num);
void rCountEvenDigits2(int num, int *result);
```

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

```
#include <stdio.h>
int rCountEvenDigits1(int num);
void rCountEvenDigits2(int num, int *result);
int main()
```

```

{
    int number, result;

    printf("Enter the number: \n");
    scanf("%d", &number);
    printf("rCountEvenDigits1(): %d\n", rCountEvenDigits1(number));
    rCountEvenDigits2(number, &result);
    printf("rCountEvenDigits2(): %d\n", result);
    return 0;
}
int rCountEvenDigits1(int num)
{
    /* Write your program code here */
}
void rCountEvenDigits2(int num, int *result)
{
    /* Write your program code here */
}

```

Some sample input and output sessions are given below:

(4) Test Case 1:
Enter the number:
105006
rCountEvenDigits1(): 4
rCountEvenDigits2(): 4

(5) Test Case 2:
Enter the number:
23453
rCountEvenDigits1(): 2
rCountEvenDigits2(): 2

(6) Test Case 3:
Enter the number:
135
rCountEvenDigits1(): 0
rCountEvenDigits2(): 0

6. (**rAllEvenDigits**) 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 even. For example, if the argument num is 2468, then the function should return 1; and if the argument num is 2345, then 0 should be returned. Write the recursive function in two versions. The function rAllEvenDigits1() computes and returns the result. The function rAllEvenDigits2() computes and returns the result through the parameter result using call by reference. The function prototypes are given below:

```

int rAllEvenDigits1(int num);
void rAllEvenDigits2(int num, int *result);

```

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

```

#include <stdio.h>
#define INIT_VALUE 999
int rAllEvenDigits1(int num);
void rAllEvenDigits2(int num, int *result);
int main()
{
    int number, result=INIT_VALUE;

    printf("Enter a number: \n");
    scanf("%d", &number);
    result = rAllEvenDigits1(number);
    if (result == 1)

```

```

        printf("rAllEvenDigits1(): yes\n");
    else if (result == 0)
        printf("rAllEvenDigits1(): no\n");
    else
        printf("rAllevenDigits1(): error\n");
    result=INIT_VALUE;
    rAllEvenDigits2(number, &result);
    if (result == 1)
        printf("rAllEvenDigits2(): yes\n");
    else if (result == 0)
        printf("rAllEvenDigits2(): no\n");
    else
        printf("rAllevenDigits2(): error\n");
    return 0;
}
int rAllEvenDigits1(int num)
{
    /* Write your code here */
}
void rAllEvenDigits2(int num, int *result)
{
    /* Write your code here */
}

```

Some sample input and output sessions are given below:

- (1) Test Case 1:
Enter a number:
5
rAllEvenDigits1(): no
rAllEvenDigits2(): no
- (2) Test Case 2:
Enter a number:
2468
rAllEvenDigits1(): yes
rAllEvenDigits2(): yes
- (3) Test Case 3:
Enter a number:
2345
rAllEvenDigits1(): no
rAllEvenDigits2(): no
- (4) Test Case 4:
Enter a number:
280
rAllEvenDigits1(): yes
rAllEvenDigits2(): yes

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>
#include <string.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 3:
Enter the string:
a
rStrLen(): 1

8. (**rStrcmp**) The **recursive** C function that 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);
```

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

```

#include <stdio.h>
#include <string.h>
#define INIT_VALUE 100
int rStrcmp(char *s1, char *s2);
int main()
{
    char source[40], target[40], *p;
    int result = INIT_VALUE;

    printf("Enter a source string: \n");
    fgets(source, 40, stdin);
    if (p=strchr(source, '\n')) *p = '\0';
    printf("Enter a target string: \n");
    fgets(target, 40, stdin);
    if (p=strchr(target, '\n')) *p = '\0';
    result = rStrcmp(source, target);
    printf("rStrcmp(): %d", result);
    return 0;
}
int rStrcmp(char *s1, char *s2)
{
    /* Write your code here */
}

```

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

9. (**rFindMaxAr**) Write a **recursive** C function that finds 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 integer parameter size indicates the size of the array. The pointer parameter max is used for passing the maximum number to the caller via call by reference. The function prototype is given as follows:

```
void rFindMaxAr(int *ar, int size, int *max);
```

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

```
#include <stdio.h>
void rFindMaxAr(int *a, int size, int *max);
int main()
{
    int ar[50], i, max, 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]);
    max=ar[0];
    rFindMaxAr(ar, size, &max);
    printf("rFindMaxAr(): %d\n", max);
    return 0;
}
void rFindMaxAr(int *ar, int size, int *max)
{
    /* Write your code here */
}
```

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
 rFindMaxAr(): 5

(2) Test Case 2:
 Enter array size:
7
 Enter 7 numbers:
2 5 4 -7 9 10 1
 rFindMaxAr(): 10

(3) Test Case 3:
 Enter array size:
3
 Enter 3 numbers:
-1 -3 -2
 rFindMaxAr(): -1

10. (**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

11. (**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
- (3) Test Case 3:
Enter size:
4
Enter 4 numbers:
1 2 4 5
rReverseAr(): 5 4 2 1