WEEK 3 ASSIGNMENT – FUNCTIONS AND POINTERS

Assignment on Coding Questions

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

1. computePay
2. computeSalary
3. sumSqDigits
4. countEvenDigits
5. allEvenDigits
6. divide
7. power
8. gcd
9. perfectProd
10. extEvenDigits
11. reverseDigits

Questions

1. (**computePay**) Write a C function that determines the gross pay for an employee in a company. The company pays straight-time for the first 160 hours worked by each employee for four weeks and pays time-and-a-half for all hours worked in excess of 160 hours. The function takes in the number of hours each employee worked for the four weeks, and the hourly rate of each employee. Write the function in two versions. The function `computePay1()` returns the gross pay to the calling function, while the function `computePay2()` returns the gross pay to the calling function through the pointer parameter, `grossPay`. The function prototypes for the function are given as follows:

```
double computePay1(int noOfHours, int payRate);
void computePay2(int noOfHours, int payRate, double *grossPay);
```

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

```
#include <stdio.h>
double computePay1(int noOfHours, int payRate);
void computePay2(int noOfHours, int payRate, double *grossPay);
int main()
{
   int noOfHours, payRate;
   double grossPay;

   printf("Enter number of hours: \n");
   scanf("%d", &noOfHours);
   printf("Enter hourly pay rate: \n");
   scanf("%d", &payRate);
   printf("computePay1(): %.2f\n", computePay1(noOfHours, payRate));
   computePay2(noOfHours, payRate, &grossPay);
   printf("computePay2(): %.2f\n", grossPay);
   return 0;
}
double computePay1(int noOfHours, int payRate)
{
   /* Write your code here */
}
void computePay2(int noOfHours, int payRate, double *grossPay)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter number of hours:
176
Enter hourly pay rate:
6
computePay1(): 1104.00
computePay2(): 1104.00
```

(2) Test Case 2:
```
Enter number of hours:
34
Enter hourly pay rate:
6
computePay1(): 204.00
computePay2(): 204.00
```

(3) Test Case 3:
```
Enter number of hours:
156
Enter hourly pay rate:
8
computePay1(): 1248.00
computePay2(): 1248.00
```

2. (**computeSalary**) Write a C program that determines the gross pay for each employee in a company. The company pays "straight-time" for the first 160 hours worked by each employee for four weeks and pays "time-and-a-half" for all hours worked in excess of 160 hours. You are given a list of employee Ids (an integer), the number of hours each employee worked for the four weeks, and the hourly rate of each employee. The program should input this information for each employee, then determine and display the employee's gross pay. The sentinel value of –1 is used for the employee *id* to indicate the end of input. Your program should include three functions, apart from the main() function, to handle the input, and the computation of the gross pay. The function prototypes for the functions are given as follows:

```c
void readInput(int *id, int *noOfHours, int *payRate);
double computeSalary1(int noOfHours, int payRate);
void computeSalary2(int noOfHours, int payRate, double *grossPay);
```

The function **computeSalary1()** uses call by value for returning the result to the calling function. The function **computeSalary2()** uses call by reference to pass the result through the pointer parameter, grossPay, to the caller.

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

```c
#include <stdio.h>
void readInput(int *id, int *noOfHours, int *payRate);
double computeSalary1(int noOfHours, int payRate);
void computeSalary2(int noOfHours, int payRate, double *grossPay);
int main()
{
   int id = -1, noOfHours, payRate;
   double grossPay;

   readInput(&id, &noOfHours, &payRate);
   while  (id != -1) {
      printf("computeSalary1(): ");
      grossPay = computeSalary1(noOfHours, payRate);
      printf("ID %d grossPay %.2f \n", id, grossPay);
      printf("computeSalary2(): ");
      computeSalary2(noOfHours, payRate, &grossPay);
      printf("ID %d grossPay %.2f \n", id, grossPay);
```

```
         readInput(&id, &noOfHours, &payRate);
      }
      return 0;
   }
   void readInput(int *id, int *noOfHours, int *payRate)
   {
      /* Write your code here */
   }
   double computeSalary1(int noOfHours, int payRate)
   {
      /* Write your code here */
   }
   void computeSalary2(int noOfHours, int payRate, double *grossPay)
   {
      /* Write your code here */
   }
```

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter ID (-1 to end):
11
Enter number of hours:
155
Enter hourly pay rate:
8
computeSalary1(): ID 11 grossPay 1240.00
computeSalary2(): ID 11 grossPay 1240.00
Enter ID (-1 to end):
12
Enter number of hours:
165
Enter hourly pay rate:
8
computeSalary1(): ID 12 grossPay 1340.00
computeSalary2(): ID 12 grossPay 1340.00
Enter ID (-1 to end):
-1
```

(2) Test Case 2:

```
Enter ID (-1 to end):
11
Enter number of hours:
155
Enter hourly pay rate:
8
computeSalary1(): ID 11 grossPay 1240.00
computeSalary2(): ID 11 grossPay 1240.00
Enter ID (-1 to end):
12
Enter number of hours:
160
Enter hourly pay rate:
8
computeSalary1(): ID 12 grossPay 1280.00
computeSalary2(): ID 12 grossPay 1280.00
Enter ID (-1 to end):
13
Enter number of hours:
200
Enter hourly pay rate:
8
computeSalary1(): ID 13 grossPay 1760.00
computeSalary2(): ID 13 grossPay 1760.00
```

```
Enter ID (-1 to end):
-1
```

(3) Test Case 3:

```
Enter ID (-1 to end):
11
Enter number of hours:
165
Enter hourly pay rate:
8
computeSalary1(): ID 11 grossPay 1340.00
computeSalary2(): ID 11 grossPay 1340.00
Enter ID (-1 to end):
-1
```

3. **(sumSqDigits)** Write a function that takes an integer argument `num` and returns the sum of the squares of the digits of the integer. For example, given the number 3418, the function should return 90, i.e. 3*3+4*4+1*1+8*8. Write two iterative versions of the function. The function **`sumSqDigits1()`** returns the computed result, while **`sumSqDigits2()`** passes the result through the pointer parameter `result`. The function prototypes are given as follows:

```
int sumSqDigits1(int num);
void sumSqDigits2(int num, int *result)
```

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

```
#include  <stdio.h>
int sumSqDigits1(int num);
void sumSqDigits2(int num, int *result);
int main()
{
   int num, result;

   printf("Enter a number: \n");
   scanf("%d", &num);
   printf("sumSqDigits1(): %d\n", sumSqDigits1(num));
   sumSqDigits2(num, &result);
   printf("sumSqDigits2(): %d\n", result);
   return 0;
}
int sumSqDigits1(int num)
{
   /* Write your code here */
}
void sumSqDigits2(int num, int *result)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1

```
Enter a number:
24
sumSqDigits1(): 20
sumSqDigits2(): 20
```

(2) Test Case 2

```
Enter a number:
3418
sumSqDigits1(): 90
sumSqDigits2(): 90
```

(3) Test Case 3

```
Enter a number:
102
sumSqDigits1(): 5
sumSqDigits2(): 5
```

4. (**countEvenDigits**) Write a C function to count the number of even digits, i.e. digits with values 0,2,4,6,8 in a non-negative number. For example, if `number` is 1234567, then 3 will be returned. Write the function in two versions. The function `countEvenDigits1()` returns the result, while the function `countEvenDigits2()` returns the result via the pointer parameter, `count`. The function prototypes are given below:

```
int countEvenDigits1(int number);
void countEvenDigits2(int number, int *count);
```

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

```
#include <stdio.h>
int countEvenDigits1(int number);
void countEvenDigits2(int number, int *count);
int main()
{
   int number, result;

   printf("Enter a number: \n");
   scanf("%d", &number);
   printf("countEvenDigits1(): %d\n", countEvenDigits1(number));
   countEvenDigits2(number, &result);
   printf("countEvenDigits2(): %d\n", result);
   return 0;
}
int countEvenDigits1(int number)
{
   /* Write your code here */
}
void countEvenDigits2(int number, int *count)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter a number:
1234567
countEvenDigits1(): 3
countEvenDigits2(): 3
```

(2) Test Case 2:

```
Enter a number:
2468
countEvenDigits1(): 4
countEvenDigits2(): 4
```

(3) Test Case 3:

```
Enter a number:
1357
countEvenDigits1(): 0
countEvenDigits2(): 0
```

5. (**allEvenDigits**) Write a function that returns either 1 or 0 according to whether or not all the digits of the positive integer argument number are even. For example, if the input argument is 2468, then

the function should return 1; and if the input argument is 1234, then 0 should be returned. Write the function in two versions. The function `allEvenDigits1()` returns the result to the caller, while `allEvenDigits2()` passes the result through the pointer parameter, `result.` The function prototypes are given below:

```
int allEvenDigits1(int num);
void allEvenDigits2(int num, int *result);
```

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

```
#include  <stdio.h>
int allEvenDigits1(int num);
void allEvenDigits2(int num, int *result);
int main()
{
   int number, p=999, result=999;

   printf("Enter a number: \n");
   scanf("%d", &number);
   p = allEvenDigits1(number);
   if (p == 1)
      printf("allEvenDigits1(): yes\n");
   else if (p == 0)
      printf("allEvenDigits1(): no\n");
   else
      printf("allEvenDigits1(): error\n");
   allEvenDigits2(number, &result);
   if (result == 1)
      printf("allEvenDigits2(): yes\n");
   else if (result == 0)
      printf("allEvenDigits2(): no\n");
   else
      printf("allEvenDigits2(): error\n");
   return 0;
}
int allEvenDigits1(int num)
{
   /* Write your code here */
}
void allEvenDigits2(int num, int *result)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter a number:
2468
allEvenDigits1(): yes
allEvenDigits2(): yes
```

(2) Test Case 2:
```
Enter a number:
1357
allEvenDigits1(): no
allEvenDigits2(): no
```

(3) Test Case 3:
```
Enter a number:
24
allEvenDigits1(): yes
allEvenDigits2(): yes
```

(4) Test Case 4:
```
Enter a number:
245
allEvenDigits1(): no
allEvenDigits2(): no
```

6. (**divide**) Write a function that performs integer division by subtraction. The function computes the quotient and remainder of dividing *m* by *n*. Both *m* and *n* are positive integers. Write the function in two versions, and the function prototypes are given below:

```
int divide1(int m, int n, int *r);
void divide2(int m, int n, int *q, int *r);
```

In the first version **divide1()**, the function returns the quotient of dividing *m* by *n*, and the remainder is passed to the caller through the pointer parameter *r*. In the second version **divide2()**, the pointer variable *q* is used to store the quotient which will be returned to the caller, and the remainder is passed to the caller through the pointer parameter *r*. Please note that in this question, you are not allowed to use the division (/) and modulus (%) operators.

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

```
#include <stdio.h>
int divide1(int m, int n, int *r);
void divide2(int m, int n, int *q, int *r);
int main()
{
   int m, n, q, r;

   printf("Enter two numbers (m and n): \n");
   scanf("%d %d", &m, &n);
   q = divide1(m, n, &r);
   printf("divide1(): quotient %d remainder %d\n", q, r);
   divide2(m, n, &q, &r);
   printf("divide2(): quotient %d remainder %d\n", q, r);
   return 0;
}
int divide1(int m, int n, int *r)
{
   /* Write your code here */
}
void divide2(int m, int n, int *q, int *r)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter two numbers (m and n):
10 3
divide1(): quotient 3 remainder 1
divide2(): quotient 3 remainder 1
```

(2) Test Case 2:
```
Enter two numbers (m and n):
3 5
divide1(): quotient 0 remainder 3
divide2(): quotient 0 remainder 3
```

(3) Test Case 3:
```
Enter two numbers (m and n):
```

```
32 7
divide1(): quotient 4 remainder 4
divide2(): quotient 4 remainder 4
```

(4) Test Case 4:
```
Enter two numbers (m and n):
0 7
divide1(): quotient 0 remainder 0
divide2(): quotient 0 remainder 0
```

7. **(power)** Write a function that computes the power `p` of a positive number `num`. 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`. In this question, you should not use any functioms from the standard math library. The function prototypes are given below:

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

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

```
#include <stdio.h>
float power1(float num, int p);
void power2(float num, int p, float *result);
int main()
{
   int power;
   float number, result=-1;

   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 */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter the number and power:
2 3
power1(): 8.00
power2(): 8.00
```

(2) Test Case 2:
```
Enter the number and power:
2 -4
power1(): 0.06
power2(): 0.06
```

(3) Test Case 3:
```
Enter the number and power:
2 0
power1(): 1.00
```

```
power2(): 1.00
```

8. (**gcd – greatest common divisor**) Write a C function 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);
```

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

```
#include <stdio.h>
int gcd1(int num1, int num2);
void gcd2(int num1, int num2, int *result);
int main()
{
   int x,y,result=-1;

   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 */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter 2 numbers:
4 7
gcd1(): 1
gcd2(): 1
```

(2) Test Case 2:
```
Enter 2 numbers:
32 4
gcd1(): 4
gcd2(): 4
```

(3) Test Case 3:
```
Enter 2 numbers:
4 38
gcd1(): 2
gcd2(): 2
```

9. (**perfectProd**) A perfect number is one that is equal to the sum of all its factors (excluding the number itself). For example, 6 is perfect because its factors are 1, 2, and 3, and 6 = 1+2+3; but 24 is not perfect because its factors are 1, 2, 3, 4, 6, 8, 12, but 1+2+3+4+6+8+12 = 36. Write a function that takes a number, `num`, prints the perfrct numbers that are less than the number, and returns the product of all perfect numbers. For example, if the number is 100, the function should

return 168 (which is 6*28), as 6 and 28 are the only perfect numbers less than 100. Write the function in two versions. The function **perfectProd1()** returns the result to the caller, while **perfectProd2()** passes the result through the pointer parameter `result`. The function prototypes are given as follows:

```
int perfectProd1(int num);
void perfectProd2(int num, int *prod);
```

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

```
#include <stdio.h>
int perfectProd1(int num);
void perfectProd2(int num, int *prod);
int main()
{
   int number, result=0;

   printf("Enter a number: \n");
   scanf("%d", &number);
   printf("Calling perfectProd1() \n");
   printf("perfectProd1(): %d\n", perfectProd1(number));

   printf("Calling perfectProd2() \n");
   perfectProd2(number, &result);
   printf("perfectProd2(): %d\n", result);
   return 0;
}
int perfectProd1(int num)
{
   /* Write your code here */
}
void perfectProd2(int num, int *prod)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter a number:
100
Calling perfectProd1()
Perfect number: 6
Perfect number: 28
perfectProd1(): 168
Calling perfectProd2()
Perfect number: 6
Perfect number: 28
perfectProd2(): 168
```

(2) Test Case 2:
```
Enter a number:
500
Calling perfectProd1()
Perfect number: 6
Perfect number: 28
Perfect number: 496
perfectProd1(): 83328
Calling perfectProd2()
Perfect number: 6
Perfect number: 28
Perfect number: 496
perfectProd2(): 83328
```

(3) Test Case 3:
```
Enter a number:
1000
Calling perfectProd1()
Perfect number: 6
Perfect number: 28
Perfect number: 496
perfectProd1(): 83328
Calling perfectProd2()
Perfect number: 6
Perfect number: 28
Perfect number: 496
perfectProd2(): 83328
```

10. (**extEvenDigits**) Write a function that extracts the even digits from a positive number, and combines the even digits sequentially into a new number. The new number is returned to the calling function. If the input number does not contain any even digits, then the function returns -1. For example, if the input number is 1234567, then 246 will be returned; and if the input number is 13, then –1 will returned. Write the function in two versions. The function `extEvenDigits1()` returns the result to the caller, while the function `extEvenDigits2()` returns the result through the pointer parameter, `result`. The function prototypes are given as follows:

```
int extEvenDigits1(int num);
void extEvenDigits2(int num, int *result);
```

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

```
#include <stdio.h>
#define INIT_VALUE 999
int extEvenDigits1(int num);
void extEvenDigits2(int num, int *result);
int main()
{
   int number, result = INIT_VALUE;

   printf("Enter a number: \n");
   scanf("%d", &number);
   printf("extEvenDigits1(): %d\n", extEvenDigits1(number));
   extEvenDigits2(number, &result);
   printf("extEvenDigits2(): %d\n", result);
   return 0;
}
int extEvenDigits1(int num)
{
   /* Write your code here */
}
void extEvenDigits2(int num, int *result)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter a number:
1234
extEvenDigits1(): 24
extEvenDigits2(): 24
```

(2) Test Case 2:
```
Enter a number:
```

```
1357
extEvenDigits1(): -1
extEvenDigits2(): -1
```

(3) Test Case 3:

```
Enter a number:
2468
extEvenDigits1(): 2468
extEvenDigits2(): 2468
```

(4) Test Case 4:

```
Enter a number:
6
extEvenDigits1(): 6
extEvenDigits2(): 6
```

11. **(reverseDigits)** Write a C function that takes in a positive integer number, reverses its digits and returns the result to the calling function. You may assume that the last digit of the input number is not 0, i.e. the number will not be in the form of 1110, 1200, etc. Write two iterative versions of the function. The function **reverseDigits1()** returns the computed result, while **reverseDigits2()** passes the result through the pointer parameter `result`. The function prototypes are given as follows:

```
int reverseDigits1(int num);
void reverseDigits2(int num, int *result);
```

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

```
#include <stdio.h>
int reverseDigits1(int num);
void reverseDigits2(int num, int *result);
int main()
{
   int num, result=999;

   printf("Enter a number: \n");
   scanf("%d", &num);
   printf("reverseDigits1(): %d\n", reverseDigits1(num));
   reverseDigits2(num, &result);
   printf("reverseDigits2(): %d\n", result);
   return 0;
}
int reverseDigits1(int num)
{
   /* Write your code here */
}
void reverseDigits2(int num, int *result)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter a number:
12045
reverseDigits1(): 54021
reverseDigits2(): 54021
```

(2) Test Case 2:

```
Enter a number:
123
reverseDigits1(): 321
```

```
reverseDigits2(): 321
```

(3) Test Case 3:

```
Enter a number:
8
reverseDigits1(): 8
reverseDigits2(): 8
```