FUNCTIONS AND POINTERS

Practice questions:

1. computePay
2. computeSalary
3. power
4. perfectProd
5. printPattern2
6. printPattern2
7. computeTotal
8. sumSqDigits
9. countEvenDigits
10. allEvenDigits
11. extEvenDigits
12. 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:

```
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:

```
#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. **(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
```

4. (**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. If there is no perfect number for the input number, then the function should return 1. 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
```

5. (**printPattern2**) Write a function that accepts an integer number `height` (between 1 and 9) as its parameter, and prints a pattern with the specified `height`. For example, if `height` = 4, the following pattern will be printed:

```
A
BA
ABA
BABA
```

If `height` = 7, then the following pattern will be printed:

```
A
BA
ABA
BABA
ABABA
BABABA
ABABABA
```

The function prototype is given below:

```
void printPattern2(int height);
```

A sample program to test the function is given below:

```
#include <stdio.h>
void printPattern2(int height);
int main()
{
   int height;

   printf("Enter the height: \n");
   scanf("%d", &height);
   printf("printPattern2(): \n");
   printPattern2(height);
   return 0;
}
void printPattern2(int height)
{
   /* Write your code here */
}
```

Sample input and output sessions are given below:

(1) Test Case 1:

```
Enter the height:
3
printPattern2():
A
BA
ABA
```

(2) Test Case 2:

```
Enter the height:
8
printPattern2():
A
BA
ABA
BABA
ABABA
BABABA
ABABABA
BABABABA
```

6. (**printPattern3**) Write a function that accepts an integer number `height` (between 1 and 9) as its parameter, and displays a pattern according to the value of `height`. For example, if `height` is 9, then the following pattern should be printed:

```
1
2 3
3 4 5
4 5 6 7
5 6 7 8 9
6 7 8 9 0 1
7 8 9 0 1 2 3
8 9 0 1 2 3 4 5
9 0 1 2 3 4 5 6 7
```

The function prototype is given below:

```
void printPattern3(int height);
```

A sample program to test the function is given below:

```c
#include <stdio.h>
void printPattern3(int height);
int main()
{
   int height;

   printf("Enter the height: \n");
   scanf("%d", &height);
   printf("printPattern3(): \n");
   printPattern3(height);
   return 0;
}
void printPattern3(int height)
{
   /* Write your code here */
}
```

Sample input and output sessions are given below:

(3) Test Case 1:

```
Enter the height:
5
printPattern3():
1
23
345
4567
56789
```

(4) Test Case 2:

```
Enter the height:
7
printPattern3():
1
23
345
4567
56789
678901
7890123
```

7. (**computeTotal**) Write a function that accepts an integer `numOfLines` as parameter, reads in specified number of lists of numbers according to `numOfLines`, one list per line, computes the total for each list and displays the total on the screen. For each list of numbers, the first number indicates how many elements are in the list. Your function does not need to check user input errors.

   The function prototype is given below:

```c
void computeTotal(int numOfLines);
```

   A sample program is given below to test the function:

```c
#include <stdio.h>
void computeTotal(int numOfLines);
int main()
{
   int numOfLines;

   printf("Enter number of lines: \n");
   scanf("%d", &numOfLines);
```

```
    computeTotal(numOfLines);
    return 0;
}
void computeTotal(int numOfLines)
{
    /* Write your code here */
}
```

Sample input and output sessions are given below:

(1) Test Case 1:

```
Enter number of lines:
2
Enter line 1:
3 2 3 4
Total: 9
Enter line 2:
4 1 2 3 4
Total: 10
```

(2) Test Case 2:

```
Enter number of lines:
3
Enter line 1:
3 2 3 4
Total: 9
Enter line 2:
4 1 2 3 4
Total: 10
Enter line 3:
2 1 2
Total: 3
```

8. **(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
```

9. (**countEvenDigits**) Write a C function to count the number of even digits, i.e. digits with values 0,2,4,6,8 in a positive number (>0). 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
```

10. (**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
```

11. (**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. You do not need to consider the input number such as 0, 20, 3033, etc. 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
```

12. (**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
```