WEEK 3 - FUNCTIONS AND POINTERS - ASSIGNMENT

You are required to submit your code for the following assignment questions to the Automated Programming Assessment System (APAS) for marking and grading. Please follow strictly the following guidelines on submission and grading.

- **Submission Deadline** - For example, if your lab session is held on the week of 15 January 2018 (any day of that week), then the deadline for your submission will be on Friday the following week 26 January 2018 (Friday), at 11.59pm.

- **Late Submission** – Late submission **will not** be accepted for grading. If you have any problems on submitting the code using the APAS system, please submit the code to our lab staff Ms Eng Hui Fang (ashfeng@ntu.edu.sg) or me via email before the deadline. Again, late submission after the deadline will not be accepted for grading. Please remember to save and submit the code you have developed for each question in your assignment.

- **Automated Grading** – Your submitted assignment code will be graded automatically by the APAS system. Please note that the grading is carried out based on test cases using program input/output with "**exact string matching**". Therefore, it is extremely important to pay special attention to the program input and output (including letter cases) when you write your programs (a simple mistake on letter case will cause your program to be marked as incorrect.). It is **your responsibility** to follow the question problem specification including the requirements for program input and output when writing your programs. As mentioned, the APAS system will only mark the correctness of the programs based on test cases, and program input/output errors will cause your programs to fail in the test cases. We will not accept any review for such errors.

- **Request for Review** – For each assignment, after your code has been marked and graded, you may submit a request to Ms Eng (ashfeng@ntu.edu.sg) for reviewing your code if you found that the grading is not correct. We will then review your submitted code accordingly. Please note that the period for your request is within one week after the assignment has been graded and returned to you (i.e. the deadline will be the following Friday after your assignment submission deadline). Please also note that **late request** beyond the request period will not be accepted.

Assignment Questions for Submission for Week 3

1. (**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 the following functions, apart from the main() function, to handle the input, the output 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);
void printOutputs(int id, double grossPay);
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.

   Write a C program to test the functions.

   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
```

(4) Test Case 4:

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

A sample program to test the function is given below:

```
#include <stdio.h>
void readInput(int *id, int *noOfHours, int *payRate);
void printOutputs(int id, double grossPay);
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);
      printOutputs(id, grossPay);
      printf("computeSalary2(): ");
      computeSalary2(noOfHours, payRate, &grossPay);
      printOutputs(id, grossPay);
      readInput(&id, &noOfHours, &payRate);
   }
   return 0;
}
void readInput(int *id, int *noOfHours, int *payRate)
{
   /* Write your program code here */
}
void printOutputs(int id, double grossPay)
{
   /* Write your program code here */
}
double computeSalary1(int noOfHours, int payRate)
{
   /* Write your program code here */
}
void computeSalary2(int noOfHours, int payRate, double *grossPay)
{
   /* Write your program code here */
}
```

2. (**divide**) Write the function **divide()** 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);
int 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.

Write a C program to test the functions.

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
```

A sample program to test the function is given below:

```
#include <stdio.h>
int divide1(int m, int n, int *r);
int 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 program code here */
}
int divide2(int m, int n, int *q, int *r)
{
   /* Write your program code here */
}
```

3. (**allEvenDigits**) The function allEvenDigits() 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);
```

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
allEvenDigits1(): Not All Even
allEvenDigits2(): Not All Even
```

(2) Test Case 2:

```
Enter a number:
2468
allEvenDigits1(): All Even
allEvenDigits2(): All Even
```

(3) Test Case 3:

```
Enter a number:
1357
allEvenDigits1(): Not All Even
allEvenDigits2(): Not All Even
```

(4) Test Case 4:

```
Enter a number:
0
allEvenDigits1(): All Even
allEvenDigits2(): All Even
```

A sample program to test the function is given below:

```
#include <stdio.h>
#define INIT_VALUE -1
int allEvenDigits1(int num);
void allEvenDigits2(int num, int *result);
int main()
{
   int number, result = INIT_VALUE;

   printf("Enter a number: \n");
   scanf("%d", &number);
   printf("allEvenDigits1(): ");
   result = allEvenDigits1(number);
   if (result == 1)
      printf("All Even\n");
   else if (result == 0)
      printf("Not All Even\n");
   else
      printf("Error\n");
   allEvenDigits2(number, &result);
   printf("allEvenDigits2(): ");
   if (result == 1)
      printf("All Even\n");
   else if (result == 0)
      printf("Not All Even\n");
   else
      printf("Error\n");
   return 0;
}
int allEvenDigits1(int num)
{
   /* Write your program code here */
}
void allEvenDigits2(int num, int *result)
{
   /* Write your program code here */
}
```

4. (**extOddDigits**) Write a function extOddDigits1() that extracts the odd digits from a positive number, and combines the odd digits sequentially into a new number. The new

number is returned to the calling function. If the input number does not contain any odd digits, then the function returns -1. For example, if the input number is 1234567, then 1357 will be returned; and if the input number is 28, then –1 will returned. Write the function in two versions. The function extOddDigits1() returns the result to the caller, while the function extOddDigits2() returns the result through the pointer parameter, *result*. The function prototype is given as follows:

```
int extOddDigits1(int num);
void extOddDigits2(int num, int *result);
```

Write a C program to test the functions.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter a number:
1234
extOddDigits1(): 13
extOddDigits2(): 13
```

(2) Test Case 2:

```
Enter a number:
2468
extOddDigits1(): -1
extOddDigits2(): -1
```

(3) Test Case 3:

```
Enter a number:
1357
extOddDigits1(): 1357
extOddDigits2(): 1357
```

(4) Test Case 4:

```
Enter a number:
7
extOddDigits1(): 7
extOddDigits2(): 7
```

A sample program to test the function is given below:

```
#include <stdio.h>
#define INIT_VALUE 0
int extOddDigits1(int num);
void extOddDigits2(int num, int *result);
int main()
{
   int number, result = INIT_VALUE;

   printf("Enter a number: \n");
   scanf("%d", &number);
   printf("extOddDigits1(): %d\n", extOddDigits1(number));
   extOddDigits2(number, &result);
   printf("extOddDigits2(): %d\n", result);
   return 0;
}
int extOddDigits1(int num)
{
   /* Write your program code here */
}
void extOddDigits2(int num, int *result)
{
```

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