Programming for Engineers Laboratory

EE058IU

Lab 3

Functions - Recursion

Full name + Student ID: ....

....

Class: ....

Group: ....

Date: ....

I. Objectives

1. Understanding the concept of the function method in C programming. Why we need to use Function Method and how to use function method to solve the specific problems.
2. Understanding about the recursion function and applying this method to solve the problems.

II. Pre-Lab Preparation

Students are required to review the theory about the topics before the lab time.

III. In-Lab Procedure

Exercise 1

Write a C program that reads several numbers and rounds each of these numbers to the nearest integer. Print both the original number and the rounded number.

Hint: Include the **`<math.h>`** and Use the C standard library function: ***`floor`*.**

<u>Output:</u>

```
Enter a number: 2.3
Original number: 2.300 and rounded number: 2.000
Enter a number: 81.91222
Original number: 81.912 and rounded number: 82.000
```

Exercise 2

Write a function that takes an integer value and returns the number with its digits reversed. Write a C program to test the function.

<u>Output:</u>

```
Enter a integer: 1234                    Enter a integer: 567891234
Reversed digits: 4321                    Reversed digits: 432198765
```

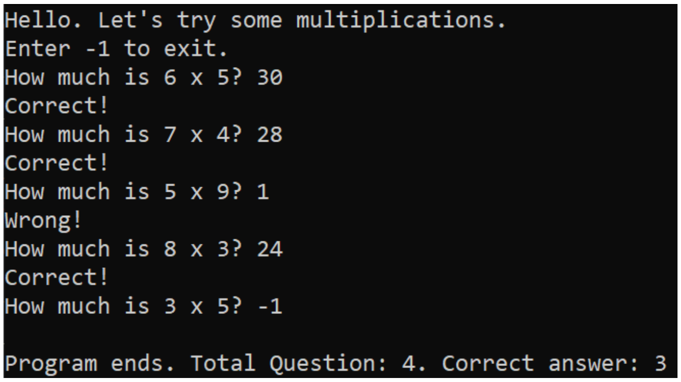
Exercise 3

Write a C program that aid a young student learns multiplication. Use srand function to generate randomly two positive 1-digit integers.

The student then entered the solution. The program should verify whether the message in the answer and response is correct or incorrect. Another question involving multiplication follows. Until the student enters a **-1** to end the program, the procedure should be repeated. The program concludes by showing the number of questions asked and accurate answers.

Note: Modularize your main program by using at least 3 functions.

<u>Output:</u>



```
Hello. Let's try some multiplications.
Enter -1 to exit.
How much is 6 x 5? 30
Correct!
How much is 7 x 4? 28
Correct!
How much is 5 x 9? 1
Wrong!
How much is 8 x 3? 24
Correct!
How much is 3 x 5? -1

Program ends. Total Question: 4. Correct answer: 3
```

Exercise 4

Write a program and a recursive function to find the greatest common divisor (gcd) of integers of integers **x** and **y** (inputted by user). GCD is the largest integer that evenly divides both x and y.

If y is equal to 0, then gcd(x, y) = x; otherwise gcd(x, y) = gcd(y, x % y), where % is the remainder operator

<u>Note: Student must create a recursive function</u>

<u>Output:</u>

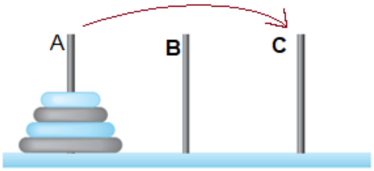
Enter a positive integer x: 5

Enter a positive integer x: 10

GCD of x and y is: 5

Exercise 5 (Optional – Bonus)

Write a C program using recursion function that solve the *Towers of Hanoi (TOH)* puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack from rod A to rod C, as shown in the below diagram.



Moving of disks should follow the following **rules:**

- One disk can be moved at a time
- A disk can only be moved if it is the uppermost disk on a stack.
- No disk can be placed on top of a smaller disk.

Your program must print the series of moves that needed to solve the puzzle.

Hint:

Moving *n* disks can be viewed in terms of moving only *n* – 1 disks (and hence the recursion) as follows:

a) Move *n* – 1 disks from rod A to rod B, using rod C as a temporary holding area.
b) Move the last disk (the largest) from rod A to rod C.
c) Move the *n* – 1 disks from rod B to rod C, using rod A as a temporary holding area.

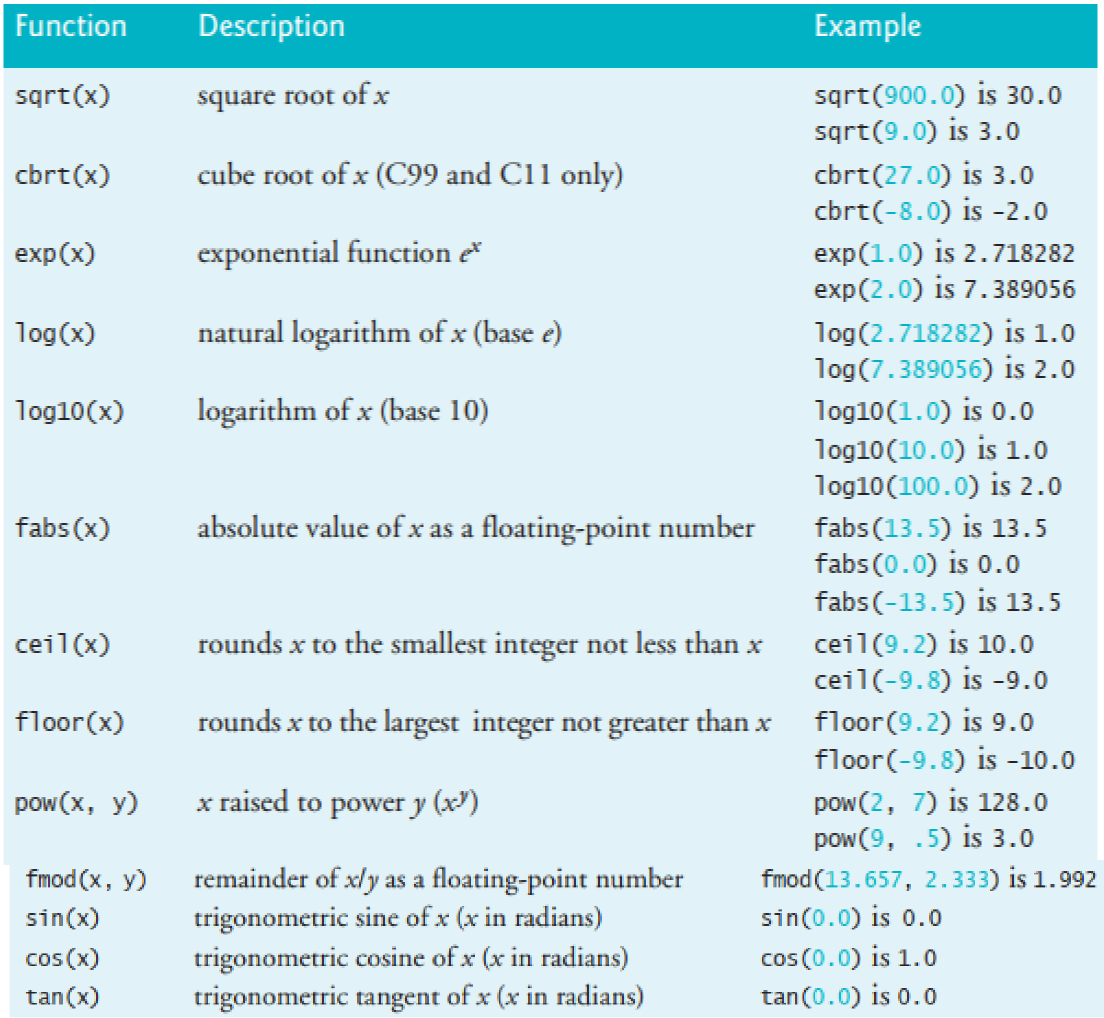
Output:

```
Enter number of disks: 3

move disk from A to C
move disk from A to B
move disk from C to B
move disk from A to C
move disk from B to A
move disk from B to C
```

move disk from A to C

Several Useful C Math Library Functions <math.h>



Function	Description	Example
`sqrt(x)`	square root of *x*	`sqrt(900.0)` is 30.0 `sqrt(9.0)` is 3.0
`cbrt(x)`	cube root of *x* (C99 and C11 only)	`cbrt(27.0)` is 3.0 `cbrt(-8.0)` is -2.0
`exp(x)`	exponential function e^x	`exp(1.0)` is 2.718282 `exp(2.0)` is 7.389056
`log(x)`	natural logarithm of *x* (base *e*)	`log(2.718282)` is 1.0 `log(7.389056)` is 2.0
`log10(x)`	logarithm of *x* (base 10)	`log10(1.0)` is 0.0 `log10(10.0)` is 1.0 `log10(100.0)` is 2.0
`fabs(x)`	absolute value of *x* as a floating-point number	`fabs(13.5)` is 13.5 `fabs(0.0)` is 0.0 `fabs(-13.5)` is 13.5
`ceil(x)`	rounds *x* to the smallest integer not less than *x*	`ceil(9.2)` is 10.0 `ceil(-9.8)` is -9.0
`floor(x)`	rounds *x* to the largest integer not greater than *x*	`floor(9.2)` is 9.0 `floor(-9.8)` is -10.0
`pow(x, y)`	*x* raised to power *y* (x^y)	`pow(2, 7)` is 128.0 `pow(9, .5)` is 3.0
`fmod(x, y)`	remainder of *x*/*y* as a floating-point number	`fmod(13.657, 2.333)` is 1.992
`sin(x)`	trigonometric sine of *x* (*x* in radians)	`sin(0.0)` is 0.0
`cos(x)`	trigonometric cosine of *x* (*x* in radians)	`cos(0.0)` is 1.0
`tan(x)`	trigonometric tangent of *x* (*x* in radians)	`tan(0.0)` is 0.0

THE END