Week 5: In-Class Exercises

***Please download the resource from** http://blue.smu.edu.sg/cs101/ice5-resource.zip*

1. **[Difficulty: *]** Implement the `is_even` function. The `is_even` function takes in a number `n`(type: `int`) as its parameter. It will return `true` if `n` is even, otherwise it returns `false`.

2. **[Difficulty: *]** Implement the following functions:
   a. `max2()` that takes as input two integers, and returns the maximum of the two integers.
   b. `max3()` that takes as input 3 integers, and returns the maximum of the 3 integers. The function should call `max2()`.

3. **[Difficulty: *]** Implement the `print_even_numbers` function. This function accepts a "maximum" number as an argument and prints all the even numbers from 2 up to that maximum (inclusive), enclosed by the curly braces.

4. **[Difficulty: *]** Implement the `get_num_holes` function. It takes in a number `n`(type: `int`) and return the hole count of the number based on the number of holes in **EACH** digit:

digit(s)	Number of holes
0,4,6,9	1
1, 2, 3, 5, 7	0
8	2

   For example, 12008 has 4 holes. Each 0 has 1 hole, 8 has 2 holes.

5. **[Difficulty: *]** Implement the `get_grade` function. It takes in a `score`(type:`int`) and returns the grade letter (type: `char`). The grading scheme is as follows:

Score	Letter Grade
90 <= score <= 100	A
80 <= score < 90	B
70 <= score < 80	C
50 <= score < 70	D
score < 50	F

6. **[Difficulty: *]** Implement the get_num_unique function. It takes three integers as parameters and that returns the number of unique integers among the three.
   a. get_num_unique(1, 2, 4) should return 3 (1, 2 & 4).
   b. get_num_unique(2, 5, 2) should return 2 (2 & 5).
   c. get_num_unique(1, 1, 1) should return 1 (1).

7. **[Difficulty: **]** Implement the `is_binary_number` function. A binary number is made up of only 0s and 1s digits. This method takes in `value`(type:`int`) and returns `True` if it is a binary number, otherwise it returns `False`.

8. **[Difficulty: **]** Implement the `swap_digit_pairs` function. It takes in an integer n (type: int), and returns an integer whose value is the pairwise digit swap of n.
   a. `swap_digit_pair(1234)` will return 2143. 1 and 2 are swapped. 3 & 4 are swapped.
   b. `swap_digit_pair(56789)` will return 57698. 6 & 7 are swapped. 8 & 9 are swapped. 5 (the first digit) is left alone.

9. **[Difficulty: *]** A common convention in C programs is to write a header file (with .h suffix) for each source file (.c suffix) that you link to your main source code. The logic is that the .c source file contains all of the code and the header file contains the function prototypes, that is, just a declaration of which functions can be found in the source file.
   Given the following add function:

```
// add.c
int add(int x, int y) {
    return x + y;
}
```

The header file will be as follows:

```
// add.h
int add(int x, int y);
```

To use the function we have defined in add.c, we will include the add.h header file in q9.c.
Note: A set of double quotes ("") are used instead of the inequality signs (< and >) because the header file is located in the same directory.

```
#include <stdio.h>
#include "add.h"

int main(void) {
    int x;
    printf("Enter x:");
    scanf("%d", &x);

    int y;
    printf("Enter y:");
    scanf("%d", &y);

    printf("%d + %d = %d\n", x, y, add(x, y));
}
```

To compile the program, you just need to specify the c files required.

```
clang -o add add.c q9.c
```

.

The **make** command is a Unix tool for managing and maintaining computer programs that consist of many component files. It reads in rules (specified as a list of target entries) from a user created Makefile. The make command will only re-compile the pieces that have been modified since the last time the objects or executables were built.

A simple Makefile for our program is as follows:

```
q9: q9.c add.c
# do note that the next line starts with a tab character
    clang -o q9 q9.c add.c
```

Note: You may need to install make.

```
sudo apt install make
```

To build the program, you just need to type make.

```
make
```

10. **[Difficulty: *]** Implement the `get_largest` function that takes as input an array and its length, and returns the largest number in the array.

OPTIONAL

11. **[Difficulty: **]** Implement the `is_perfect_number` function. It takes in a `num`(type:`int`) and returns the true if num is a perfect number, otherwise it returns false. A perfect number is a positive number which is equal to the sum of all its divisors excluding itself.

12. **[Difficulty: *]** Implement a function `are_all_prices_higher_than`. It takes in three parameters: `numbers, len`(size of the numbers array) and `min_value`, and returns `True` if all the values in `numbers` are greater than the `min_value`. Otherwise, this function returns `False`.

13. **[Difficulty: ***]** Implement a function called `print_calendar`. This function takes in 2 parameters
    a. num_days (type: int): This specifies the number of days in a month
    b. first_sun (type:int): the date of the first Sunday in that month

    It then prints the calendar.

    E.g. 1: If the function is invoked like this:

    ```
    print_calendar(30, 2);
    ```

    the statement generates the following output:

```
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

c. Write a function `day_of_week()` that takes as input a day, a month, and a year, representing a date, and returns the day of the week of that date. The days of the week are represented by integer values from 0 for Sunday, to 6 for Saturday. For example, `days_of_week(1, 1, 2000)` returns `6`. Then implement a `print_calendar(day, month, year)` function.
Reference: https://artofmemory.com/blog/how-to-calculate-the-day-of-the-week-4203.html

14. **[Difficulty Level: ****] A function can not only be called by other functions but also by itself. A function that calls itself is a *recursive* function. If you have watched the movie "Inception" (2010) then *recursion* is not a new concept to you. Just like you can dream inside a dream, you can call a method inside itself. But you have to take caution not to get trapped in an infinite loop. The key is to provide a "stopping criterion" that establishes a base case.

Study the following program (`q14demo.c`). The function `compute_sum` is supposed to compute the sum from 1 to n where n is a parameter. Look at Line 06. Instead of using a for-loop to compute the sum, the method calls itself to compute the sum from 1 to (n – 1) and then adds n to the sum. This is generally correct if you think about it. However, when n is 1, it does not make sense to compute the sum from 1 to (n – 1) because (n – 1) is 0 in this case. So when n is 1, we have a base case that we have to handle. In this case, we should simply return 1 as the sum. Lines 06 to 10 use an if/else statement to check and handle this base case. Trace the code below to make sure you know how `compute_sum(5)` is executed.

```
01  // Author: Dom Cobb
02  #include <stdio.h>
03
04  int compute_sum(n) {
05      int sum;
06      if (n == 1) { // stopping criterion
07          sum = 1;
08      } else {
09          sum = compute_sum(n - 1) + n;
10      }
11      return sum;
12  }
13
14  int main(void) {
15      printf("%d\n", compute_sum(5));
16  }
```

A. Now try to write a recursive method that computes the factorial of n. You do not need a for-loop in your method. You do need if/else to check for the base case.

B. Now write another recursive method that takes in a positive integer and prints out the digits of the number line by line. A sample run of a program that uses this method is shown below.
Try solving it with the use of the modulo (%) operator!

```
cwarrior:/mnt/c/cs101/ice11$./a.out
Enter a positive integer:139748
1
3
9
7
4
8
D:\ice5>
```

C. Solve the fibonacci question using recursion. Your function should behave as follows:

```
# the parameter is the "n"-th to generate
printf("%d\n", fibonacci(1));  // 0
printf("%d\n", fibonacci(2));  // 1
printf("%d\n", fibonacci(3));  // 2
printf("%d\n", fibonacci(4));  // 3
printf("%d\n", fibonacci(5));  // 5
printf("%d\n", fibonacci(6));  // 8
printf("%d\n", fibonacci(7));  // 13
```