CS29003 Algorithms Laboratory
Assignment 1: Logarithmic vs Linear vs Exponential Growth of Functions

General instruction to be followed strictly

1. Do not use any global variable unless you are explicitly instructed so.
2. Do not use Standard Template Library (STL) of C++.
3. Use proper indentation in your code and comment.
4. Name your file as `<roll_no>_<assignment_no>`. For example, if your roll number is 14CS10001 and you are submitting assignment 3, then name your file as `14CS10001_3.c` or `14CS10001_3.cpp` as applicable.
5. Write your name, roll number, and assignment number at the beginning of your program.

Consider the situation of a spread of a viral disease. In the 0-th month, there was no infected person. The disease begins with one infected person in the first month. Once a person gets infected, he/she remains infected for two months. Every infected person infects two healthy person in the first month and one healthy person in the second month. Let $\mathcal{I}_n$ denote the number of infected person in the n-th month. Then we have the following.

$$\mathcal{I}_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ 2\mathcal{I}_{n-1} + \mathcal{I}_{n-2} & \text{if } n \geqslant 2 \end{cases}$$

In this assignment, we will compute the number $\mathcal{I}_n$ of infected people in the n-th month in various ways. For this assignment, you can assume that n is at most 40. You do not need to verify this. Simply take the value of n as input from the user (using keyboard) and output $\mathcal{I}_n$.

Method I: Compute $\mathcal{I}_n$ iteratively

You define 3 variables called `current`, `previous`, and `next` of data type **double**. Although $\mathcal{I}_n$ is always an integer, its value grows so quickly that you will overflow the capacity of int or long even for small values of n. You initialize and update these 3 variables appropriately in a loop which runs for at most n iterations to find the value of $\mathcal{I}_n$. So the running time of your algorithm is $\mathcal{O}(n)$. Implement this algorithm in a function whose prototype is given below. You need to follow the prototype strictly.

double compute_iterative(int);

Method II: Compute $\mathcal{I}_n$ recursively

Another approach to compute $\mathcal{I}_n$ would be to write a recursion whose pseudocode is as follows.

```
compute_recursive(n){
    if n=0, then return 0
    else if n=1, then return 1
    else if n>1, then return 2*compute_recursive(n-1)+compute_recursive(n-2);
}
```

What is the running time of this algorithm? Implement this algorithm in a function whose prototype is given below. You need to follow the prototype strictly.

double compute_recursive(int);

Method III: Compute $\mathcal{I}_n$ using formula

One can easily verify that the following formula for $\mathcal{I}_n$ satisfies the recurrence relation.

$$\mathcal{I}_n = \frac{\left(1+\sqrt{2}\right)^n - \left(1-\sqrt{2}\right)^n}{2\sqrt{2}}$$

Write a function for computing x^n for a double x and int n which should take $\mathcal{O}(\log n)$ time. **You should not use pow function.** Use the following prototype.

double power(double, int);

Now you compute $\mathcal{I}_n$ using the above formula. Use the following function prototype.

double compute_formula(int);

Method IV: Compute $\mathcal{I}_n$ using matrix multiplication

Method III takes $\mathcal{O}(\log n)$ time to compute $\mathcal{I}_n$ but it requires computing square root and division which are comparatively slow. Another problem of method III is that, although mathematically correct, C program based on this method fails to compute correct value of $\mathcal{I}_n$ for large enough value of n since it involves division by double type variables. Method IV avoids these operations with compromising running time. This method uses the following formula.

$$\begin{pmatrix} \mathcal{I}_{n+1} & \mathcal{I}_n \\ \mathcal{I}_n & \mathcal{I}_{n-1} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^n$$

Now you compute $\mathcal{I}_n$ using the above formula. Your algorithm should run in $\mathcal{O}(\log n)$ time. Use the following function prototype.

double compute_matrix(int);

main()

1. Read n from the user.
2. Compute $\mathcal{I}_n$ by all the four methods by calling appropriate functions. Also output the amount of time (in seconds) taken by each method.

One possible way to compute execution time of some piece of code is the following. You are allowed to use any other method.

```
#include <stdio.h>
#include <time.h>       // for clock_t, clock(), CLOCKS_PER_SEC

int main()
{
// to store execution time of code
double time_spent = 0.0;

clock_t begin = clock();

// do some stuff here

clock_t end = clock();

// calculate elapsed time by finding difference (end - begin) and
// dividing the difference by CLOCKS_PER_SEC to convert to seconds
time_spent += (double)(end - begin) / CLOCKS_PER_SEC;

printf("Time elpased is %f seconds", time_spent);

return 0;
}
```

Submit a single .c or .cpp file. Your code should get compiled properly by gcc or g++ compiler.

Sample Output

```
Write n: 10
I_10 (computed using iterative method) = 2378.000000
Time taken in iterative method = 0.000076 seconds

I_10 (computed using recursive method) = 2378.000000
Time taken in recursive method = 0.000027 seconds

I_10 (computed using formula) = 2378.000000
Time taken in formula = 0.000022 seconds

I_10 (computed using matrix multiplication) = 2378.000000
Time taken in matrix = 0.000026 seconds
```

```
Write n: 40
I_40 (computed using iterative method) = 723573111879672.000000
Time taken in iterative method = 0.000062 seconds
```

```
I_40 (computed using recursive method) = 723573111879672.000000
Time taken in recursive method = 0.777038 seconds

I_40 (computed using formula) = 723573111879670.750000
Time taken in formula = 0.000003 seconds

I_40 (computed using matrix multiplication) = 723573111879672.000000
Time taken in matrix = 0.000002 seconds
```
