SYNCHRONOUS MULTITREADING

This project consists of writing a **Synchronous Multithreaded Program** that implements *Luhn Algorithm* for Credit card number validation.

Specifics:

- Create multiple threads, each of which run in parallel creating **Thread-level parallelism**
- This involves examining the algorithm steps to find areas that can be divided into separate, **concurrent** tasks. Tasks (threads) are independent of one another and thus can run in parallel on multiple computing cores.
- The data accessed by the threads must be examined for dependencies between two or more threads. When one thread depends on data from another, ensure that the execution of the tasks is **synchronized** to accommodate the data dependency
- In **Synchronous Threading**, parent thread creates multiple children (or worker threads) and then must wait for all of its children to terminate before it resumes. The threads created by the parent perform work concurrently, but the parent cannot continue until this work has been completed. Once each thread has finished its work, it terminates and joins (example: pthread_join(), or Java's thrd.join()) with its parent. Only after all of the children have joined can the parent resume execution.
- Synchronous threading involves significant data sharing among threads. The parent thread combines the results calculated by its various children and outputs the final results.

Luhn Algorithm

The *Luhn Algorithm*- also known as the *Modulus 10 check* - is a formula that is used to determine whether the identification number provided by a user is accurate. The formula is widely used in validating credit card numbers. The algorithm is used to validate a variety of identification numbers, such as National Provider ID, Canadian SIN, Israeli ID, South African ID, Greek SSN, and International Mobile Equipment ID.

Luhn Algorithm for Credit card number validation

Program Steps:

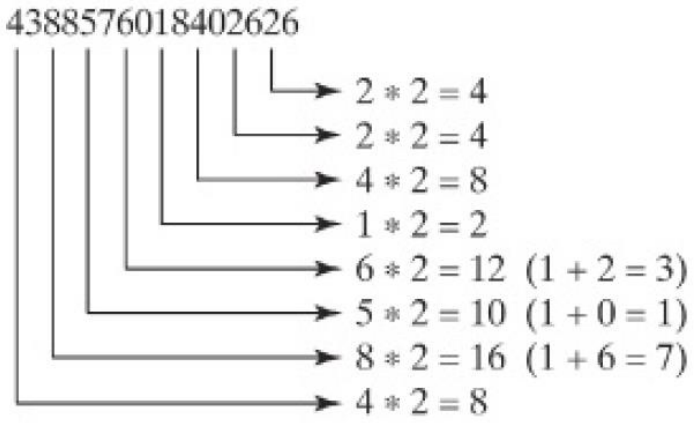
Credit Card numbers follow certain patterns. Verify the following requirements:

I. A credit card number must have between 13 and 19 digits.
II. Check if it starts with the major industry identifier of the Issuer identification number
    - 3 for American Express cards
    - 6 for Discover cards
    - 5 for Master cards
    - 4 for Visa cards

Issuing network	IIN ranges	Length	Validation
American Express	34, 37	15	Luhn algorithm
Discover Card	6	16–19	Luhn algorithm
Mastercard	51–55	16	Luhn algorithm
Visa	4	13. 16	Luhn algorithm

Card numbers are generated following this validity check which can be described as follows (for illustration, consider the card number 4388576018402626):

III. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single digit number.



2. Now add all single-digit numbers from Step III.

4+4+8+2+3+1+7+8=37

IV. Add all digits in the odd places from right to left in the card number.

6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38

V. Sum the results from Step III and Step IV.

37+38=75

VI. If the result from Step V is divisible by 10, the card number is valid; otherwise, it is invalid.

Test Data:

The number 4388576018402626 is invalid

The number 4388576018410707 is valid.

Implementation:

1. This program will be passed a series of credit card numbers on the command line. Card numbers must not be hard-coded in the program. Embedding data directly into the source code demands recompiling.
2. Allocate each worker thread a computational task of the algorithm. Parent will create separate worker threads for program steps 1, II, III, and IV.
3. Once the worker threads have exited, the parent thread will output the following results:
    a. Display which of the the credit card numbers are valid or and which are invalid.
    b. If valid, display if the card is issued by American Express, Discover, Mastercard, or Visa.

Language:

1. You can opt for C, C++, or Java
2. You may solve this using POSIX Pthreads, Windows API, or Java thread API

Execution Efficiency:

To measure the efficiency of multithreading, we will compare the execution time of the algorithm execution with and without threads.

1. Measure the program execution times.

    In Linux, just write `time` before what you would usually write to run your program from the terminal command line. In the output, the '`user`' gives the CPU time.

Submission

After completion, your program is to report the following:

1. A two or three page project report. The report must be in your own words.
   The report details:
    a. your approach,
    b. valid and invalid data used,
    c. task allocations,
    d. functionality and outputs of each thread,
    e. methods (not the code, but details),
    f. thread creation,
    g. thread join,
    h. difficulties you overcome, and
    i. sources you have used for your project.
   You may use resource. You must include bibliographic citations for all references..

2. Source codes exclusively in txt, pdf, or word format
3. Screenshots of results
4. Screenshots of execution time.