

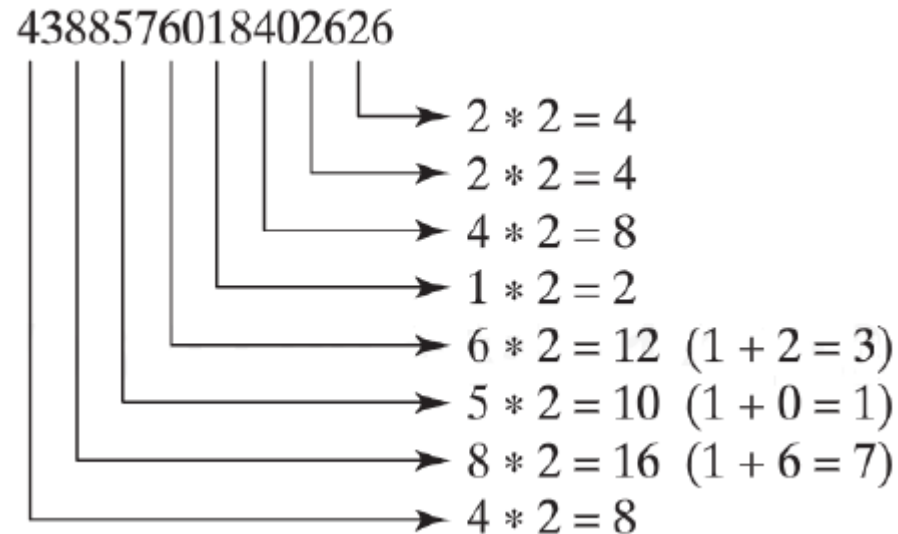
HW #2 (Credit card number validation)

- Design a modular **C++** (or **Java** if you prefer) program to solve the following problem.
- Credit card numbers follow certain patterns. A credit card number must have between 13 and 16 digits. The number must start with the following:
 - 4 for Visa cards
 - 5 for MasterCard cards
 - 37 for American Express cards
 - 6 for Discover cards

HW #2 (2)

- In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or is scanned correctly by a scanner. Almost all credit card numbers are generated following this validity check, commonly known as the *Luhn check* or the *Mod 10 check*. It can be described as follows. (For illustration, consider the card number 4388576018402626.)
1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add the two digits to get a single digit number.

HW #2 (3)



2. Now add all single-digit numbers from Step 1.

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

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

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

HW #2 (4)

4. Sum the results from Step 2 and Step 3.

$$37 + 38 = 75$$

5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid. Likewise, 4226610081169571 and 5490123456789128 are valid numbers as well.

- Write a program that reads a file containing a list of **newline**-delimited **strings**, each representing a credit card number. Display whether the number is valid.

HW #2 (5)

- Also, the file may or may **not** have a **newline** character before **EOF**.
- Moreover, we define in advance four particular strings, “004”, “005”, “0037”, and “006”, for other purpose, which is explained in page 7.

Design your program to use the following functions:

// Return true if the card number is valid

bool isvalid (const string& cardNumber)

HW #2 (6)

// Get the result from Step 2

int sumofDoubleEvenPlace (const string& cardNumber)

// Return this number if it is a single digit, otherwise,

// return the sum of the two digits

int getDigit (int number)

// Return sum of odd-place digits in the card number

int sumOfOddPlace (const string& cardNumber)

HW #2 (7)

// Return true if substr is the prefix for cardNumber
*bool startsWith (const string& cardNumber,
 const string& substr)*

// Return a **valid** credit card number for the given type:
// “004” for Visa cards
// “005” for MasterCard cards
// “0037” for American Express cards
// “006” for Discover cards
string& fakeOne (const string& type)

HW #2 (8)

// One easier way for doing this is you keep generating new
// number combination using a **random number generator**
// until a valid card number of given type is found. There
// may be other better ways....

// You should **time** your code, for example, calling
// *gettimeofday()*, and report timing information in
// **microseconds**.

HW #2 (9)

Here are the main criteria for this problem

- Read the name of the data file from the command line in `argv[1]`.
- Open the file, read the data and save individual credit card number in an array.
- Compute and check whether that credit card number is valid.
- Display the final answer with an annotation.

HW #2 (10)

Program requirements

- Use good program style. This includes (but is not limited to the following).
 1. Short main function with top-level function calls.
 2. Separate interface and implementation files for your globals.
 3. Indentation and whitespace.
 4. Descriptive names for your functions/constants/variables.
 5. Symbolic constants where appropriate.

HW #2 (11)

- A namespace around your globals.
- A namespace alias in main to preface your functions invocations.
- Error checking should trap any potential error, such as too few command line args, then print an appropriate message and quit.

HW #2 (12)

- **Sample Input**

4388576018402626

5490123456789128

004

004

- **Sample Output**

4388576018402626: an invalid Visa card #

5490123456789128: a valid MasterCard card #

Generated a valid Visa card #: 4388576018410707; Timing: 123 microseconds

Generated a valid Visa card #: 4226610081169571; Timing: 168 microseconds

Fake credit card numbers for all major brands

- **Visa:**

4556443985096249
4532490130177027
4872229942657026
4485664397248591
4532716601603095
4929373876667651

- **MasterCard:**

5483429059012878
5124710941822367
5591330937779507
5537538941421471
5285606799287098
5203316333196172

- **American Express (AMEX):**

373025858858430
373868046043179
374893294266032
375577400395277
372988244909889

- **Discover:**

6011818740391627
6011783412179876
6011811675541373
6011800857885708
6011904390073358
6011980675192147