**The Luhn Algorithm**

**What is Luhn Algorithm?**

Created in the year 1954 by an IBM scientist Hans Peter Luhn, the Luhn Algorithm is a check digit system used for various number identification and verification purposes, for instance, verifying credit card numbers, IMEI numbers, National Provider Identifier number in the US, South African ID numbers and many more. The Lunh Algorithm is also known as the “Modulus 10 Algorithm” and is an essential component in the electronics payments system.

The biggest advantage of the Luhn Algorithm is that it can instantly check whether or not the online transactions are made using a valid credit card as it can easily calculate if the number entered is valid using a simple program or formula.

**Advantages of Luhn Algorithm in day-to-day life**

1. As mentioned above, the Luhn Algorithm checks if the entered credit card number is valid or not. This will ensure that the user gets notified about the incorrect input. This saves time and effort as the user can instantly recheck the input values and correct the mistake made before sending the transaction details for further inspection. In simple words, it helps to validate credit card numbers to ensure that there are no payment errors when processing a transaction.
2. The Luhn Algorithm also helps prevent people from trying to use fraudulent or fake credit cards.
3. The Luhn Algorithm is just an added layer of an ever improved security to give the customers more peace of mind when making purchases.

The usage of Luhn Algorithm does not necessarily have to be linked with online transactions only. It is as much used in physical mode of payment as in online mode.

**Limitations of Luhn Algorithm**

Despite being widely used in global security, the Luhn Algorithm poses few limitations.

1. Generation of check digit in Luhn Algorithm is a disclosed public knowledge, so anyone can generate a fake card number satisfying the formula to bypass the credit card field.
2. It only uses one check digit, so while creating any official number at random, it has 10% chance to be correct without even checking it.
3. It may detect single-digit error, as well as almost all transpositions of adjacent digits, but it will not detect transposition of the two-digit sequence like 04 and 40.

**How Luhn Algorithm Works?** **Steps to validate a given credit card number**

1. Write down the 16-digit credit card number in a piece of paper.
2. Compute the weighted sum with the weight pattern 2, 1, 2, 1, 2, 1… such that starting with 0th and ending with 15th index number, the evenly placed numbers get multiplied with 2 and the oddly placed numbers get multiplied by 1. If multiplying a digit by 2 gives a two-digit number, those two digits get added up to form a single-digit number.
3. Add up all the results and if the resultant number R is a multiple of 10 (), then the ID number is valid.

**Example 1:** Use Luhn Algorithm to determine if the credit card number 4847 3529 8926 3094 is valid or not.

Solution.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Digit** | **4** | **8** | **4** | **7** | **3** | **5** | **2** | **9** | **8** | **9** | **2** | **6** | **3** | **0** | **9** | **4** |
|  | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* |
| **Weight** | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 |
|  | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| **Product** | 8 | 8 | 8 | 7 | 6 | 5 | 4 | 9 | 16 | 9 | 4 | 6 | 6 | 0 | 18 | 4 |
|  |  |  |  |  |  |  |  |  | 1+6 |  |  |  |  |  | 1+8 |  |
| **Result** | 8 | 8 | 8 | 7 | 6 | 5 | 4 | 9 | 7 | 9 | 4 | 6 | 6 | 0 | 9 | 4 |

Summation of the resultant numbers, R = 8 + 8 + 8 + 7 + 6 + 5 + 4 + 9 + 7 + 9 + 4 + 6 + 6 + 0 + 9 + 4

= 100

100 mod 10 = 0.

Therefore, the given credit card number is valid.

**Example 2:** Find the check digit for the credit card number 3379 5135 6110 879X.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Digit** | **3** | **3** | **7** | **9** | **5** | **1** | **3** | **5** | **6** | **1** | **1** | **0** | **8** | **7** | **9** | **X** |
|  | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* |
| **Weight** | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 |
|  | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| **Product** | 6 | 3 | 14 | 9 | 10 | 1 | 6 | 5 | 12 | 1 | 2 | 0 | 16 | 7 | 18 | X |
|  |  |  | 1+4 |  | 1+0 |  |  |  | 1+2 |  |  |  | 1+6 |  | 1+8 |  |
| **Result** | 6 | 3 | 5 | 9 | 1 | 1 | 6 | 5 | 3 | 1 | 2 | 0 | 7 | 7 | 9 | X |

Sum, R = 6 + 3 + 5 + 9 + 1 + 1 + 6 + 5 + 3 + 1 + 2 + 0 + 7 + 7 + 9 + X

= 65 + X

So, for (65 + X ) mod 10 = 0,

X = 5.

Therefore, the check digit is 5.

And the Valid Credit Card Number do formed is: 3379 5135 6110 8795.

**Implementation of Luhn Algorithm in Programming.**

**Algorithm**

Step I : START

Step II : Input Credit Card Number  
Step III : Check if the number is 16-digit numeric {

If No,

Step IV : Return Invalid Input

If Yes,

Step V : Check every digit for odd or even (Digit % 2 == 0) {

If odd,

Step VI : Multiply every odd number by 1

If even,

Step VII : Multiply every even number by 2.

Step VIII : Check if the obtained multiplied number is of 2 digit, i.e., Is Num >= 10?{

If Yes,

Step IX : Num = Num % 10 + 1

If No,

Step X : Skip to next step.

}

}

Step XI : Add every number obtained after multiplication.

Step XII : Check if the obtained sum result is multiple of 10 (i.e, if Sum % 10 == 0) {

If Yes,

Step XIII : Display the credit card number is valid.

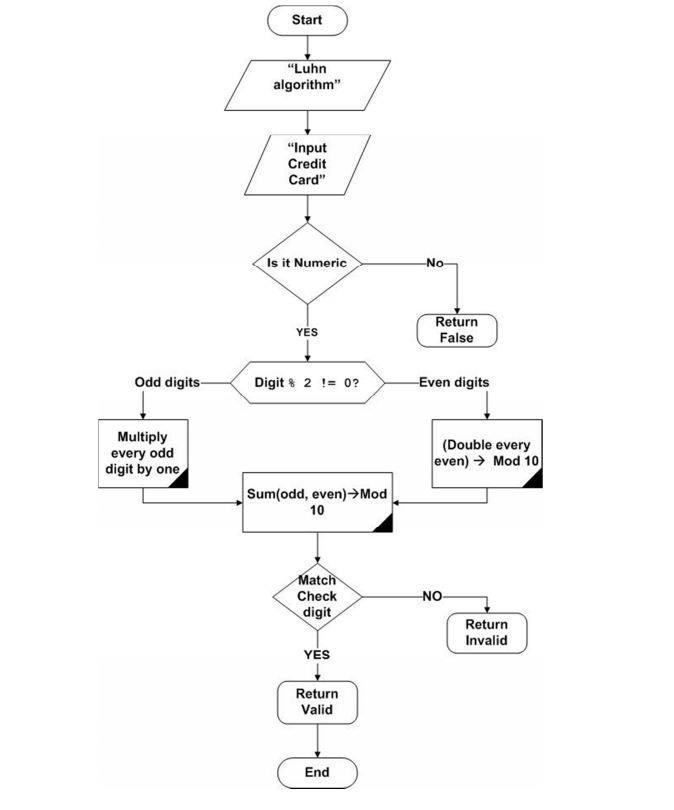
If No,

Step XIV : Display the credit card number is invalid.

}

Step XV : TERMINATE

**Flowchart**



**C Program**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void ValidityCheck ()

{

    int input[16], product, summation = 0;

    char TempInput;

    printf ("\nENTER THE 16-BIT NUMBER WHOSE VALIDITY YOU WISH TO CHECK:\n> ");

    for (int i=0; i<16; ++i)

    {

        TempInput = getche ();

        input [i] = TempInput - '0';

        if ((i+1) % 4 == 0)

            printf ("\t");

    }

    for (int i = 1; i < 16; i = i+2)

    {

        summation += input [i];

    }

    for (int i = 0; i<15; i = i+2)

    {

        product = input [i] \* 2;

        if (product > 9)

            product = (product % 10) + 1;

        summation += product;

    }

    if (! (summation % 10))

        printf ("\nVALID CARD\n");

    else

        printf ("\nINVALID CARD\n");

}

void GenerateCheckDigit ()

{

    int input[16], product, summation = 0, CheckDigit;

    char TempInput;

    printf ("\nENTER THE FIRST 15-DIGITS OF THE NUMBER YOU WISH TO GENERATE:\n> ");

    for (int i=0; i<15; ++i)

    {

        TempInput = getche ();

        input [i] = TempInput - '0';

        if ((i+1) % 4 == 0)

            printf ("\t");

    }

    for (int i = 1; i < 14; i = i+2)

    {

        summation += input [i];

    }

    for (int i = 0; i<15; i = i+2)

    {

        product = input [i] \* 2;

        if (product > 9)

            product = (product % 10) + 1;

        summation += product;

    }

    CheckDigit = ((summation / 10)\*10 + 10) - summation;

    input [15] = CheckDigit;

    printf ("\nThe Check Digit is %d", CheckDigit);

    printf ("\nThe Valid Credit Card Number so-formed is ");

    for (int i=0; i<16; ++i)

    {

        printf ("%d", input[i]);

        if ((i+1)%4 == 0)

            printf ("\t");

    }

}

int main()

{

    printf ("##########################\n");

    printf ("#### LUHN ALGORITHM ######\n");

    printf ("##########################\n");

    printf ("\nSelect one of the options: ");

    printf ("\n1. Check Validity of the Credit Card Number");

    printf ("\n2. Generate the check digit");

    printf ("\n3. Exit\n\n> ");

    int option;

    scanf ("%d", &option);

    switch (option)

    {

        case 1: ValidityCheck();

                break;

        case 2: GenerateCheckDigit();

                break;

        default: printf ("Exiting...");

                exit(0);

    }

    return 0;

}

**Output**

