



C++ PROGRAMMING

(335)

REGIONAL 2023

APPLICATION KNOWLEDGE:

Test Score Statistics Reporter _____ (350 points)

TOTAL POINTS _____ (350 points)

Test Time: 90 minutes

Credit Card Verification

All credit card readers need to verify that a credit card is valid before an attempt is made to pull money from the bank account. In this exercise, you will create a C++ console application that performs credit card verification using the Luhn Algorithm.

The process to verify credit card numbers first involves verifying that the value is between 13 and 16 digits – note that this will require 64bits when stored as an integer value. Additionally, proper prefix must be present. The list of valid prefixes are as follows: “4”, “5”, “37”, and “6”, and will indicate what company the card is from.

The Luhn algorithm is a Mod 10 based verification process, that involves grouping each digit based on whether it is in an even or odd position in the number, starting from right and going left. Digits in an even position are doubled (if this results in a 2-digit number, add the digits together such that it is a single-digit value again) prior to adding them together, and odd position are simply added together.

The final step of the check is to add both sums together, and then verify that the result is divisible by 10 – if it is, then the card number is valid.

```
Enter a Credit Card number to check validity for (-1 to exit application): test
Invalid input - only enter numerical values that can be contained in a 64bit signed value (max 9223372036854775807).
Enter a Credit Card number to check validity for (-1 to exit application): o
Invalid input - only enter numerical values that can be contained in a 64bit signed value (max 9223372036854775807).
Enter a Credit Card number to check validity for (-1 to exit application): 0
0 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 4670009360252821
4670009360252821 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 374319177519123
374319177519123 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 5329858139046159
5329858139046159 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 6670009760252821
6670009760252821 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 6670009960252821
6670009960252821 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): -1
Closing application.
Press any key to continue . . .
```

Figure 1: Sample output showing various valid credit card numbers and application flow.

Requirements:

1. You must create a C++ console application named CPP_335_ContestantNumber, where ContestantNumber is your BPA assigned contestant number (including dashes). For example, CPP_335_01_2345_6789.
2. Your contestant number must appear as a comment at the top of the main source code file.
3. When application starts, the user shall be prompted to input a credit card number to check validity for
 - a. Entering '-1' shall cause the application to be closed
 - b. After validity is reported, the user shall be prompted for an additional number to check
 - c. The user's input shall also be validated to ensure that it is not text input
4. Application shall properly employ the described credit card verification method using the Luhn algorithm
 - a. Verify card number is between 13 and 16 digits long
 - b. Starting right to left, sum all even position digits (double them first) and sum all odd position digits
 - c. Sum results from step b (odd summation and doubled even summation)
 - d. Verify result from step c is divisible by 10 (valid if so)
 - e. Report validity as shown

Your application will be graded on the following criteria:

Solution and Project

The project is present on the flash drive	_____ 10 pts
The project is named according to the naming conventions	_____ 10 pts

Program Execution

Code copied to USB drive and the program runs from USB	_____ 10 pts
--	--------------

If the program does not execute, then the remaining items in this section receive a score of zero.

Application displays error message if invalid text input	_____ 20 pts
Application displays error message if invalid numerical input	_____ 20 pts
Application successfully reports validity for test values	_____ 50 pts
Application properly loops to allow additional inputs to be entered	_____ 30 pts
Application pauses once execution is complete	_____ 20 pts

Source Code Review

Code is commented at the top, for each function, and as needed	_____ 20 pts
Code uses reasonable and consistent variable naming conventions	_____ 20 pts
Data types are handled in a logical and consistent manner	_____ 30 pts
Application user input logic is constructed properly	_____ 30 pts
The specified functions are present and are constructed in a logical manner	_____ 40 pts
Algorithm is constructed in manner described	_____ 40 pts

Total Points: 350 pts

Expected Output from Key Input, using provided bat file

```
C:\bpa_2022_audits\regionals\regionals\x64\Debug>regionals.exe 0<numbers.txt
Enter a Credit Card number to check validity for (-1 to exit application): 4670009360252821 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 4670006833278045 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 4670006645749514 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 5154787999388577 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 5236687425499786 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 5161238656200631 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 5420162200623808 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 5329858139046159 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 374319177519123 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 374713834758605 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 374317658739574 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 4670002360252821 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 4670003833278045 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 4670005645749514 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 5154786999388577 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 5236681425499786 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 5161233656200631 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 5420161200623808 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 5329855139046159 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 374319277519123 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 374713534758605 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 374317358739574 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): Closing application.
Press any key to continue . . .

C:\bpa_2022_audits\regionals\regionals\x64\Debug>pause
Press any key to continue . . .
```

Completed Source Code

```
#include <iostream>
#include <string>

using namespace std;

// Return this number if it is a single digit, otherwise,
// return the sum of the two digits
int getDigit(int64_t number)
{
    if (number < 9)
        return static_cast<int>(number);
    return static_cast<int>(number / 10 + number % 10);
}

// Return the number of digits in d
int getSize(int64_t d)
{
    string num = to_string(d);
    return static_cast<int>(num.length());
}

// Return the first k number of digits from
// number. If the number of digits in number
// is less than k, return number.
long getPrefix(int64_t number, int k)
{
    if (getSize(number) > k)
```

```

    {
        string num = to_string(number);
        return stol(num.substr(0, k));
    }
    return static_cast<long>(number);
}

// Return true if the digit d is a prefix for number
bool prefixMatched(int64_t number, int d)
{
    return getPrefix(number, getSize(d)) == d;
}

// Get the result from Step 2
int sumOfDoubleEvenPlace(int64_t number)
{
    int sum = 0;
    string num = to_string(number);
    for (int i = getSize(number) - 2; i >= 0; i -= 2)
        sum += getDigit(int(num[i] - '0') * 2);

    return sum;
}

// Return sum of odd-place digits in number
int sumOfOddPlace(int64_t number)
{
    int sum = 0;
    string num = to_string(number);
    for (int i = getSize(number) - 1; i >= 0; i -= 2)
        sum += num[i] - '0';

    return sum;
}

// Verifies that the sum of odd and even digits is divisible by 10
bool verifySum(int64_t number)
{
    int sum = sumOfDoubleEvenPlace(number) + sumOfOddPlace(number);

    return sum % 10 == 0; // divisible by 10 - no remainder
}

bool verifyPrefix(int64_t number)
{
    return prefixMatched(number, 4) ||
        prefixMatched(number, 5) ||
        prefixMatched(number, 37) ||
        prefixMatched(number, 6);
}

// Return true if the card number is valid

```

```

bool isValid(int64_t number)
{
    bool valid = false;

    if (getSize(number) >= 13 && getSize(number) <= 16)
    {
        if (verifyPrefix(number))
        {
            if (verifySum(number))
            {
                valid = true;
            }
        }
    }
}

return valid;
}

bool is_number(const string& s)
{
    return(strspn(s.c_str(), "-.0123456789") == s.size());
}

// Entry point
int main()
{
    int64_t num = 0L; // Defined as long long (64bit signed - value is signed because -1 is exit criteria for
    application)
    string input;

    do
    {
        cout << "Enter a Credit Card number to check validity for (-1 to exit application): ";

        getline(cin, input);

        if (!is_number(input) || input.size() > 19) // guards stoll call to ensure value is valid
        {
            cout << "Invalid input - only enter numerical values that can be contained in a 64bit signed value (max
            9223372036854775807)." << endl;
        }
        else
        {
            num = stoll(input.c_str());

            if (num != -1)
            {
                cout << num << " is " << (isValid(num) ? "valid" : "invalid") << endl;
            }
            else
            {
                cout << "Closing application." << endl;
            }
        }
    } while (num != -1);
}

```

```
        }  
    }  
} while (num != -1);  
  
system("pause");  
return 0;  
}
```