

Object-Oriented Systems Development

CW1 - The BUCKS Centre for the Performing Arts (Design & Implementation)



January 5, 2023

Muhammad Shakeel

21924162

Contents

[Credit card validation code 2](#_Toc123773301)

[Credit card validation explanation 6](#_Toc123773302)

[Test Case for Credit card validation , check month , check year , check CVV 7](#_Toc123773303)

# Credit card validation code

**creditcardvalidation.h**

#include <iostream>

#include <string>

#include <stdlib.h>

using namespace std;

class creditcardvalidation

{

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

// return the sum of the two digits

int getDigit(int number)

{

  if (number < 9)

    return number;

  return number / 10 + number % 10;

}

// Return the number of digits in d

int getSize(std::int64\_t d)

{

  string num = to\_string(d);

  return num.length();

}

// Return the first k number of digits from

// number. If the number of digits in number

// is less than k, return number.

std::int64\_t getPrefix(std::int64\_t number, int k)

{

  if (getSize(number) > k)

  {

    string num = to\_string(number);

    return stol(num.substr(0, k));

  }

  return number;

}

// Return true if the digit d is a prefix for number

bool prefixMatched(std::int64\_t number, int d)

{

  return getPrefix(number, getSize(d)) == d;

}

// Get the result from Step 2

int sumOfDoubleEvenPlace(std::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(std::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;

}

// Return true if the card number is valid

public:bool isValid(std::int64\_t number)

{

  return (getSize(number) >= 13 &&

          getSize(number) <= 16) &&

    (prefixMatched(number, 4) ||

     prefixMatched(number, 5) ||

     prefixMatched(number, 37) ||

     prefixMatched(number, 6)) &&

    ((sumOfDoubleEvenPlace(number) +

      sumOfOddPlace(number)) % 10 == 0);

}

void is\_valid\_Run()

{

// If the credit card number is valid, check the expiration month

// If the credit card number is not valid, prompt the user to enter a new number

    // tested on card number 5196081888500645

 std::int64\_t cardnumber;

std::cout << "Please enter credit card number." << std::endl;

     cin >> cardnumber;

  if(isValid(cardnumber)==true)

  {

    cout << cardnumber << " credit card number is " << "valid"<<std::endl;

    check\_Month();

  }

  else

  {

    cout << cardnumber << " credit card is " << "invalid"<<std::endl;

    cardnumber=0;

    is\_valid\_Run();

  }

}

void check\_Month()

{

  // Prompts the user to enter the expiration month of the credit card

    // If the month is between 1 and 12 (inclusive), check the expiration year

    // If the month is not between 1 and 12, prompt the user to enter a new month

    // checking month between 1 and 12

    std::cout << "Please enter expiry date month." << std::endl;

    int month = 0;

    std::cin >> month;

    if (month >= 1 && month <= 12) {

        std::cout << "The month is accepted." << std::endl;

        check\_Year();

    } else {

        std::cout << "The month is not accepted." << std::endl;

        check\_Month();

    }

}

void check\_Year()

{

  // Prompts the user to enter the expiration year of the credit card

    // If the year is between 23 and 27 (inclusive), check the CVV

    // If the year is not between 23 and 27, prompt the user to enter a new year

    // usialy credit card have 4 to 5 years of expiry

    std::cout << "Please enter expiry date year like 23,24,25 and 26." << std::endl;

    int year = 0;

    std::cin >> year;

    if (year >= 23 && year <= 27) {

        std::cout << "The year is accepted." << std::endl;

        check\_CVV();

    } else {

        std::cout << "The year is not accepted." << std::endl;

        check\_Year();

    }

}

void check\_CVV()

{

  // Prompts the user to enter the CVV of the credit card

    // If the CVV is between 100 and 999 (inclusive), accept the credit card

    // If the CVV is not between 100 and 999, prompt the user to enter a new CVV

    // usialy credit card have length 3 cvv number

    std::cout << "Please enter 3 digit CVV usualy on card back side." << std::endl;

    int cvv = 0;

    std::cin >> cvv;

    if (cvv >= 100 && cvv <= 999) {

        std::cout << "The cvv is accepted." << std::endl;

        std::cout << "Your card successfully accepted." << std::endl;

    } else {

        std::cout << "The cvv is not accepted." << std::endl;

        check\_CVV();

    }

}

};

**Main.cpp**

#include <iostream>

#include "creditcardvalidation.h"

int main() {

creditcardvalidation card;

card.is\_valid\_Run();

    return 0;

}

(GeeksforGeeks,2022)

# Credit card validation explanation

This C++ program is a credit card validation program. It has several functions that are used to validate a credit card number.

The **getDigit** function takes an integer as input and returns either that number or the sum of the digits of the number, depending on whether the number is a single digit or not.

The **getSize** function takes an **int64\_t** type as input and returns the number of digits in that number.

The **getPrefix** function takes an **int64\_t** type and an integer as input and returns the first **k** digits of the number, where **k** is the second input. If the number of digits in the number is less than **k**, it returns the entire number.

The **prefixMatched** function takes an **int64\_t** type and an integer as input and returns **true** if the integer is a prefix for the number, and **false** otherwise.

The **sumOfDoubleEvenPlace** function takes an **int64\_t** type as input and returns the sum of the double of the even-place digits in the number.

The **sumOfOddPlace** function takes an **int64\_t** type as input and returns the sum of the odd-place digits in the number.

The **isValid** function takes an **int64\_t** type as input and returns **true** if the number is a valid credit card number, and **false** otherwise. It checks the length of the number, the prefix, and the sum of the double of the even-place digits and the odd-place digits.

The **is\_valid\_Run** function prompts the user to enter a credit card number and then calls the **isValid** function to check if the number is valid. If the number is valid, it calls the **check\_Month** function to verify the expiration date of the credit card. If the number is not valid, it prompts the user to enter a new credit card number.

The **check\_Month** function prompts the user to enter the expiration date month of the credit card and checks if the month is between 1 and 12. If the month is valid, it calls the **check\_Year** function to verify the expiration date year. If the month is not valid, it prompts the user to enter a new month.

The **check\_Year** function prompts the user to enter the expiration date year of the credit card and checks if the year is between 23 and 27. If the year is valid, it prints a message saying that the credit card is valid and the program ends. If the year is not valid, it prompts the user to enter a new year.

# Test Case for Credit card validation , check month , check year , check CVV

| **Test Case** | **Input** | **Expected Output** |
| --- | --- | --- |
| Valid credit card number | 5196081888500645 | true |
| Invalid credit card number | 1234567890 | false |
| Credit card number with invalid prefix | 1234567890123456 | false |
| Credit card number with invalid length | 123456789012345 | false |
| Credit card number with invalid checksum | 5196081888500644 | false |

Here is a table of test cases that you can use to test the **check\_Year** function:

| **Test Case** | **Input** | **Expected Output** |
| --- | --- | --- |
| Valid year | 25 | true |
| Invalid year | 20 | false |
| Year on lower bound | 23 | true |
| Year on upper bound | 27 | true |
| Year below lower bound | 22 | false |
| Year above upper bound | 28 | false |

Here is a table of test cases that you can use to test the **check\_Month** function:

| **Test Case** | **Input** | **Expected Output** |
| --- | --- | --- |
| Valid month | 6 | true |
| Invalid month | 13 | false |
| Month on lower bound | 1 | true |
| Month on upper bound | 12 | true |
| Month below lower bound | 0 | false |
| Month above upper bound | 13 | false |

Here is a table of test cases that you can use to test the **check\_CVV** function:

| **Test Case** | **Input** | **Expected Output** |
| --- | --- | --- |
| Valid CVV | 123 | true |
| Invalid CVV | 1234 | false |
| CVV with all digits the same | 111 | true |
| CVV with all digits in sequence | 123 | true |

# References:

Program for credit card number validation (2022) GeeksforGeeks. Available at: https://www.geeksforgeeks.org/program-credit-card-number-validation/ (Accessed: January 5, 2023).