**CREDIT CARD VALIDATION**

A CAPSTONE PROJECT REPORT

# (Object Oriented Programming with C++ in Advanced Topics- DSA0121)

***Submitted to***

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

***In partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING IN COMPUTER SCIENCE & ENGINEERING**

***By***

1. **Pramod Reddy (192111335)**

***Course Faculty***

**Ms.K.Divya**



**SAVEETHA SCHOOL OF ENGINEERING, SIMATS, CHENNAI - 602105**

**MARCH-2024**

**SAVEETHA SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUE OF MEDICAL AND TECHNICAL SCIENCES,CHENNAI - 602105**

**BONAFIDE CERTIFICATE**

Certified that this project report **“CREDIT CARD VALIDATION”** is the Bonafide work of **“G.Pramod Reddy”** who carried out the project work under my supervision.

**Submitted to**

**Ms.K.Divya**

(Course Faculty)

Department of Deep Learning, Saveetha School of Engineering, SIMATS

## SIGNATURE of Course Faculty

**ACKNOWLEDGEMENT**

This project work would not have been possible without the contribution of many people. It gives me immense pleasure to express my profound gratitude to our Honorable Chancellor **Dr. N. M. Veeraiyan**, Saveetha Institute of Medical and Technical Sciences, for his blessings and for being a source of inspiration. I sincerely thank our Director of Academics **Dr. Deepak Nallaswamy,** SIMATS, for his visionary thoughts and support. I am indebted to extend my gratitude to our Director **Dr. Ramya Deepak,** Saveetha School of Engineering, for facilitating us all the facilities and extended support to gain valuable education and learning experience.

I register my special thanks to **Dr. B. Ramesh,** Principal, Saveetha School of Engineering for the support given to me in the successful conduct of this project. I wish to express my sincere gratitude to my Course faculty **Ms.K.Divya**, for his inspiring guidance, personal involvement and constant encouragement during the entire course of this work.

I am grateful to Project Coordinators, Review Panel External and Internal Members and the entire faculty of the Department of Design, for their constructive criticisms and valuable suggestions which have been a rich source to improve the quality of this work.

**STUDENT NAME’s**

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **CHAPTER**  **NO** | **TITLE** |
| **1** | **Introduction** |
| **2** | **Project Description and Goals:** |
| **3** | **Technical Specifications:** |
| **4** | **Design Approach and Details** |
| **5** | **Schedule, Tasks, and Milestones:** |
| **6** | **Project Demonstration:** |
| **7** | **Cost Analysis:** |
| **8** | **Result:** |
| **9** | **Discussion:** |
| **10** | **Conclusion:** |

**Introduction:**

Designing a credit card validator involves creating a program that can verify whether a given credit card number is valid or not. A valid credit card number typically adheres to certain rules defined by the issuer and the Luhn algorithm.

:****Understanding Credit Card Number Structure****: Before designing the validator, it's crucial to understand the structure of credit card numbers. Different credit card issuers have different formats and lengths for their card numbers. For example, Visa cards typically start with a '4', Mastercard with '5', American Express with '3', etc. Additionally, each card has a specific length and may include other identifying digits like a card verification code (CVV).

****Researching Validation Algorithms****: The Luhn algorithm, also known as the modulus 10 or mod 10 algorithm, is commonly used to validate credit card numbers. It involves a simple checksum formula that validates a variety of identification numbers, including credit card numbers. Understanding how the Luhn algorithm works is essential for building a credit card validator.

****Defining Input and Output****: Determine how the validator will receive credit card numbers for validation. It could be through a command-line interface, a graphical user interface, or as input parameters in a function. Similarly, decide how the validation result will be presented, whether through a simple text message, a status indicator, or as part of a larger application.

****Designing the Validator Logic****: Based on the knowledge gained from steps 1 and 2, design the logic for the credit card validator. This involves implementing the Luhn algorithm to perform checksum validation and applying any additional rules specific to the card issuer. The validator should be able to handle different card types and lengths.

****Error Handling****: Implement error handling mechanisms to handle invalid input, such as non-numeric characters, incorrect lengths, or

unsupported card types. Provide meaningful error messages to guide users in providing correct input.

****Testing and Validation****: Thoroughly test the credit card validator with various test cases, including valid and invalid credit card numbers, different card types, and edge cases. Validate the validator's accuracy by comparing its results with known valid and invalid credit card numbers.

****User Interface (Optional)****: Depending on the application's requirements, you may design a user-friendly interface for the credit card validator. This could include graphical elements, input validation feedback, and clear presentation of validation results.

****Integration and Deployment****: Once the credit card validator is complete, integrate it into your application or system. Ensure proper deployment, documentation, and maintenance to support ongoing use and updates.

**Project Description and Goals:**

The goal of designing a credit card validator is to create a program that can verify the validity of credit card numbers entered by users. Credit card numbers are typically 16-digit numeric strings, and their validity can be checked using the Luhn algorithm, also known as the "modulus 10" or "mod 10" algorithm. The Luhn algorithm is widely used to validate credit card numbers and other identification numbers.

**Functionality:**

The functionality of a credit card validator revolves around the verification of credit card numbers entered by users.

\* Input handling

\* Format validation

\* Luhn algorithm

**User-Friendly Interface:**

An intuitive interface will be designed with clear navigation and user-friendly controls to ensure ease of use customers.

**Accurate Calculation:**

Luhn algorithm is implemented to ensure accurate card numbers and the calculations of the card numbers given by the user.

**Error Handling:**

Robust error handling mechanisms will be incorporated to detect and manage exceptions gracefully, with informative error messages provided to assist users.

**Cross-Platform Compatibility:**

The system will be developed using technologies that ensure compatibility across different platforms, websites**.**

**Documentation and Support:**

Comprehensive documentation, including user manuals and troubleshooting guides, will be provided. Support channels such as FAQs and email support will be available for user assistance.

**Testing and Validation:**

Testing and validation are crucial methods in the credit card validation process as before we release this on the websites to the customers we need to test in several times as its crucial to not get errors. We test and validate by using :

\* Unit testing

\* Integration testing

\* Boundary testing

\* Regression testing

\* Security testing

\* Real world testing

**Technical Specifications:**

Technical specifications foer a credit card validator would typically be programming language, libraries, fonts, input/output formtas, oerformance requirements, and any other revelant considerations.

**Platform Compatibility:**

The platform compatibility of a credit card validator depends on the programming language, operating systems, frameworkand libraries, deployment of environment, mobile platforms, web compatibity and cloud platforms are used to develop it.

**Design Approach and Details:**

A modular and scalable design approach will be adopted to facilitate future enhancements and modifications. Object-oriented design principles will be utilized to promote code reusability and maintainability.

**Schedule, Tasks, and Milestones:**

**Planning Phase:**

Define project scope, objectives, and requirements.

Determine the scope of credit card validator.

Identify the target paltforms.

**Research and Design Phase:**

Research existing credit crad validation algorithms.

Design architecture of the credit card validator.

Define class structures and relationships.

Create wireframes or mockups for the user interface.

**Project Demonstration:**

A demonstration will be organized to showcase system features and functionalities, gathering feedback for further refinement and improvement.

**Cost Analysis:**

Project costs, including development resources, licenses, and infrastructure, will be estimated and compared with expected benefits and returns on investment.

**Result:**

The Credit card validation system stands as a comprehensive solution for the replacement of card redares difficultiness to the intricate demands of modern-day usage. It successfully integrates all the functionalities and enhancing overall efficiency. The system's implementation adheres to industry standards, ensuring robustness and reliability in day-to-day activities.

**Discussion:**

A credit card validator is a crucial component in any system that deals with processing credit card payments. It serves as a first line of defense against fraudulent transactions by verifying the validity of credit card numbers entered by users. The primary purpose of a credit card validator is to ensure that the credit card numbers entered by users are legitimate and adhere to the format specified by credit card issuers. By validating credit card numbers, businesses can reduce the risk of processing fraudulent transactions and protect themselves from financial losses. The Luhn algorithm, also known as the modulus 10 or mod 10 algorithm, is the most commonly used method for validating credit card numbers. It involves a series of mathematical operations on the digits of the credit card number to determine its validity. The algorithm is designed to catch common errors in credit card numbers, such as typos or transcription errors, while also detecting some types of fraudulent numbers.

Overall, a credit card validator is a critical tool for businesses that handle credit card payments, helping to prevent fraud, ensure data security, and maintain trust with customers. By implementing an effective credit card validator, businesses can mitigate risks associated with fraudulent transactions and provide a secure payment experience for their customers.

**Summary:**

A credit card validator is a software component designed to verify the validity of credit card numbers entered by users. It plays a crucial role in ensuring the integrity of financial transactions and protecting against fraudulent activities. Key points are purpose, Luhn algorithm, Functionality, Components, Testing and Validation, Deployment, Compliance.

**Conclusion:**

In conclusion, designing a credit card validation system is a fundamental aspect of ensuring the security and reliability of financial transactions in various industries. Through the implementation of sophisticated algorithms such as the Luhn algorithm and rigorous testing methodologies, credit card validators play a crucial role in preventing fraudulent activities and safeguarding the interests of both businesses and consumers. The design process involves careful consideration of factors such as input handling, validation logic, error handling, and compliance with industry standards and regulations. By breaking down the project into manageable components, setting clear objectives and timelines, and conducting thorough testing and validation, developers can create robust credit card validators that accurately verify the validity of credit card numbers across different platforms and environments.

**Code:**

#include <QtWidgets>

// Function to validate credit card number using Luhn algorithm

bool validateCreditCard(const QString& cardNumber) {

int sum = 0;

bool alternate = false;

for (int i = cardNumber.length() - 1; i >= 0; --i) {

int digit = cardNumber[i].digitValue();

if (alternate) {

digit \*= 2;

if (digit > 9)

digit = (digit % 10) + 1;

}

sum += digit;

alternate = !alternate;

}

return (sum % 10 == 0);

}

int main(int argc, char \*argv[]) {

QApplication app(argc, argv);

// Create main window

QMainWindow window;

window.setWindowTitle("Credit Card Validator");

// Create central widget

QWidget \*centralWidget = new QWidget(&window);

window.setCentralWidget(centralWidget);

// Create layout

QVBoxLayout \*layout = new QVBoxLayout(centralWidget);

// Create label for credit card number

QLabel \*cardNumberLabel = new QLabel("Enter your credit card number:", centralWidget);

layout->addWidget(cardNumberLabel);

// Create line edit for credit card number input

QLineEdit \*cardNumberLineEdit = new QLineEdit(centralWidget);

layout->addWidget(cardNumberLineEdit);

// Create button to validate credit card number

QPushButton \*validateButton = new QPushButton("Validate", centralWidget);

layout->addWidget(validateButton);

// Create label for result

QLabel \*resultLabel = new QLabel(centralWidget);

layout->addWidget(resultLabel);

// Connect button click event to validation function

QObject::connect(validateButton, &QPushButton::clicked, [&]() {

QString cardNumber = cardNumberLineEdit->text();

if (!cardNumber.isEmpty()) {

bool isValid = validateCreditCard(cardNumber);

if (isValid) {

resultLabel->setText("Credit card number is valid.");

} else {

resultLabel->setText("Credit card number is invalid.");

}

} else {

resultLabel->setText("Please enter a credit card number.");

}

});

// Set layout for central widget

centralWidget->setLayout(layout);

// Set window size and show

window.resize(400, 200);

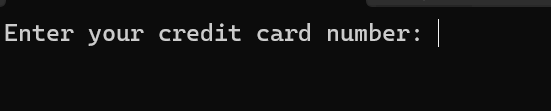
window.show();

return app.exec();

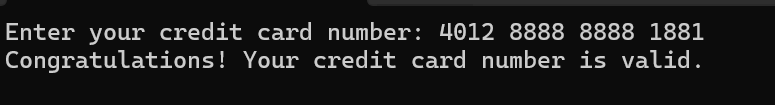
}

**OUTPUT:**

**//** This is where the user needs to enter the card number



**//** If entered number is validated you will get output as:



// If enterd output is invalid:

