ООП Password Manager представяне

Част 0 – Intro

Здравейте, казвам се Лъчезар! Подготвил съм проект за запазване на паролите на потребители за дадени уебсайтове, като паролите са запазени в криптиран вид. Нека започнем с реализацията на програмата.

Част 1 – Шифрите

Йерархията започва от абстрактния клас Cipher. Той изисква пет метода – encrypt, decrypt, name, writeConfig и readConfig. С тези методи подчертавам какво се изисква от всеки един от шифърите а именно: всеки шифър трябва да може да кодира и декодира текст, да се представя еднозначно чрез кратък идентификатор и да знае как да запише и възстанови собствената си конфигурация във файл-поток.

CaesarCipher

Конкретните имплантации на всички шифри имат статична член-данна ID и статична член-функция makeFromArgs. ID и MakefromArgs се използват от CipherCreator-ите за които ще говорим малко по-късно. Сега нека започнем с това как работи този шифър.

Шифъра запазва едно цяло число shift, като при кодиране от позицията на всеки символ в изважда shift, а резултатът се умножава по модул 95, за да остане в допустимия Ascii range.

Дешифрирането е огледален процес с прибавяне на shift.

writeConfig записва shift като сурови 4 байта; readConfig ги възстановява, така че обектът се самоконфигурира при отваряне на файл с пароли.

TextCodeCipher

TextCodeCipher се използва чрез конструктор, който приема път до текстов файл, който се използва като „референтен текст“. Файлът се сканира символ по символ, като първата поява на всеки символ в допустимия Ascii-range получава номер, който всъщност е неговата позиция. По този начин изграждаме две таблици, а именно encode, която по подаден символ ни връща индекса на първата му поява(ако присъства в таблицата) и decode, която по подаден индекс връща символа който съответства на дадения индекс.

Кодирането заменя всеки знак с неговия индекс, изведен като поредица от цифри, разделени с интервали.

Декодирането приема текст, който се очаква да е поредица от цифри разделени с интервал(понеже кодирането връща текст в такъв формат). Начина по, който работи дешифрирането е следния: итерираме през получения текст като проверяваме дали сегашния символ е цифра или интервал. Ако е цифра акумулираме дадената цифра във външна променлива num, по следния начин:

num = num \* 10 + (c - '0');

Иначе ако сегашния символ е интервал, добавяме към резултатния текст символа на който съответства акумулираното число(ако е валидно).

writeConfig извежда броя елементи и самата decode таблица.

readConfig прави обратното и изгражда таблицата endode, чрез прочетената таблица decode.

HillCipher

При създаване той приема квадратна матрица която ще използва като ключ за криптиране и декриптиране, след което веднага изчислява нейната обратна матрица.

Накрая запазва тези две матрици като член данни.

За шифриране и дешифриране използваме функцията process, която приема текст и матрица. Идеята на process е да умножи подадената матрица по числовата версия на подадения текст, като две матрици. Реализира се като първо разбива текста на блокове, след което преобразува символите в числа, умножава ги по подадената матрица и връща резултата като низ от символи.

При шифрирането викаме функцията process, която ни връща криптирания текст.

За дешифриране, отново използваме process, само че с обратната матрица.

writeConfig записва n и всички елементи на матрицата; readConfig ги възстановява и пресмята \_invKey.

Следващата важна тема за проекта е:

**Част 2 – Плъгин-архитектурата**

Архитектурата разчита на класически Factory модел, разширен до plug-in стил, така че добавянето на нов шифър изисква един-единствен нов .cpp/.h чифт и никакви изменения по съществуващия код.

CipherCreator

Първия важен компонент за реализацията на тази архитектура е именно класът CipherCreator, той пази в себе си идентификатор на шифъра, на който е създател, както и две виртуални функции – createFromArgs (когато шифърът се създава чрез подадени параметри от конзолата) и createFromStream (когато шифърът се създава чрез файл съдържащ необходимата конфигурация). Най-важната част от този клас е неговия конструктор, който приема низ с име на шифъра. Този конструктор кара създателя на шифъра с подаденото име да се саморегистрира в CipherFactory.

Следващия важен компонент е именно:

AutoCreator<T>

Това е универсален шаблон, който наследява CipherCreator.

Чрез дефоулт конструктора си той вика конструктора на CipherCreator, като му подава идентификатора на шаблонния тип T::ID. Също така, той реализира двете чисто виртуални функции на CipherCreator, createFromStream и createFromArgs.

createFromArgs просто извиква T::makeFromArgs(args), която връща поинтър към Cipher, именно затова всеки конкретен наследник на Cipher, трябва да реализира тази функция.

Другата реализирана виртуална функция е createFromStream(), тя получава поток за четене, чрез който създаваме шифър. Тази функция вика default конструктора на T с new, след което чете необходимата информация от потока чрез виртуалната фунцкия readConfig и връща новосъздадения поинтър.

По този начин няма нужда всеки шифър да има собствен малък клас-създател; отпада дублиран код, регистрацията се случва по следния начин static AutoCreator<T> \_;. Единственият компромис е изискването за дефолтен конструктор.

Следващия важен компонент на архитектурата е:

CipherFactory

Това е сингълтън клас, който държи вектор от всички регистрирани CipherCreator-и. В него имаме функция за регистриране на шифъри, която просто добавя подадения създател към вектора от създатели.

Имаме също функция getCreator(const std::string& name) за намиране на търсения създател, която просто получава неговото име и обхожда вектора докато не го намери, ако не го намери връща nullptr

Функциите createCipherFromArgs и createCipherFromStream получават име на създател и съответно или аргументи или поток. Тези функции връщат готов за използване шифър като викат съответно createFromArgs или createFromStream на създателя.

Последната част от този проект са

Част 3 – PasswordManager и CommandHandler

Класът PasswordManager управлява чувствителните данни на потребителите и ги синхронизира с двоичен файл. Съдържа член-променливи като:

\_filename – път до файла

\_password – главна парола

\_cipher – указател към шифър

\_entries – вектор с криптирани записи

\_countPos – позиция на броя записи във файла (при оптимизация)

Когато потребителят стартира командата create, мениджърът получава име на файл, избран шифър и главна парола (<filePass>). Той записва тези стойности във вътрешните си член-променливи, занулява вектора с пароли и веднага извиква saveToFile, за да изгради чисто нов двоичен файл, в който вече присъстват криптираната <filePass>, името и конфигурацията на шифъра, плюс броя записи – засега нула.

При open мениджърът първо изисква пътя до файла и въведената <filePass>, след това разчита двоичния поток: взема криптираната <filePass>, името на шифъра и неговата конфигурация. С помощта на CipherFactory и createCipherFromStream пресъздава обекта-шифър, разкодира главната парола и проверява, че съвпада. Ако всичко е наред, зарежда броя записи и ги копира в \_entries.

savePassword приема сайт, потребител и декриптирана парола; шифрова паролата чрез \_cipher->encrypt, създава запис, добавя го към \_entries и синхронизира файла. Функцията избягва пълно презаписване, като добавя новия запис в края и актуализира числото-брояч.

updatePassword търси в паметта комбинацията сайт + потребител, разкодира текущата стойност и, ако новата парола е различна, я криптира, подменя я в entries и после презаписва файла.

deletePassword премахва първия срещнат запис със съвпадащи сайт и потребител, след което отново запазва промените във файла; ако не намери съвпадение, методът просто връща отрицателен резултат.

deleteWebsite работи по същия принцип, но изхвърля всички записи за въпросния website; при поне една изтрита парола записът на диска се подновява, иначе остава непроменен.

Последния клас който ще разгледаме е:

Класът `CommandHandler` той играе ролята на посредник между потребителя и основната логика на програмата. Неговото предназначение е да приема команди от потребителя, да ги разчита, валидира и пренасочва към съответните функции на `PasswordManager`.

Командите се въвеждат чрез метода `run()`, който чете вход от потребителя и извиква `executeCommand()`. Там се разпознава какъв тип операция се изисква, като за целта се използва функцията ‚split‘ която разделя входа на потребителя на команди – например `create`, `open`, `save`, `load`, `update` или `delete`. Всяка от тези команди се обработва от отделна функция (`handleCreate`, `handleOpen` и т.н.), която проверява броя и валидността на аргументите, създава нужния шифър чрез фабриката и извиква конкретния метод на `PasswordManager`.