HCMC UNIVERSITY OF TECHNOLOGY – VNU-HCM
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

PROGRAMMING FUNDAMENTALS PROJECT - CO1031

ASSIGNMENT

QR CODE PAYMENTS TOOL

HO CHI MINH CITY, 09/2021

ASSIGNMENT'S SPECIFICATION

Version 1.0

1. Introduction

Currently, the payment trend in the economy is shifting toward using a cashless payment approach. This method brings many advantages to the users:

- **Not keeping a huge amount of cash**: Users don't worry about stealing, counterfeit money, or fire damage.
- **No face-to-face meetings**: Users can save travel time.
- **Detail transaction:** Compared to using cash, people have recorded what they bought on papers or notes. This can be time-consuming and difficult to remember each expenditure clearly. If these notes are lost, transaction information will be lost. Cashless payment instead automatically stores transactions information on record whenever they're established.
- **Earn interest from banks**: People can put their idle money into a saving account to earn extra income.

Realizing the above benefits, banks began to deploy technologies to support cashless payments. Technologies typically used are E-Banking and Mobile Banking. With them, the payment transfer is done by entering the recipient's account number and bank name. Then, the system will check this account number and display the corresponding account name for the

sender to identify if the recipient is correct. The sender enters the amount of money and the message, then sends it to the recipient.

Thanks to the Napas National Payment Gateway[1], all transactions can be made directly, like paying cash. However, when it comes to direct payments (such as payments at cashier counters in stores and supermarkets), online payments still have limitations in involved steps. The sender needs to have the recipient's account number to make the payment, which is often a long and difficult number to remember. In addition, the sender may enter this number incorrectly, resulting in re-entry, which is quite annoyed for the sender. In some cases, if the senders don't check carefully, they will transfer money to the wrong account.

Perhaps, there is still a way to improve the interaction and allow users to make interbank cashless payments more easily, quickly and with fewer errors from user input. This way can be achieved by using a QR code scanning application.

The QR code payment solution is almost implemented as follows:

1. The seller will generate a QR code containing the recipient's account information and give it to the buyer.
2. Buyers will use their smartphones to scan the above QR code and make a money transfer on their Mobile Banking application.

Smart mobile is an essential item for every individual today. Therefore, payment by QR code is a solution that can replace the current cash payment.

Smart mobile is an essential item for every individual today. Therefore, payment by QR code is a solution that can replace the current cash payment. This payment can be applied to small

[1] CÔNG TY CỔ PHẦN THANH TOÁN QUỐC GIA VIỆT NAM - NAPAS

convenience stores when dealing with customers and minimizing communication. It is possible to reduce the risk of spreading the epidemic in the COVID-19 pandemic. Shopping and payment will be safer for people, especially in Ho Chi Minh City and the southern provinces. The distribution of goods will not be congested, difficult and reduce the overall economic loss of the localities.

In this assignment, we will implement a way to support money transfer by QR code between different banks. Typically, when using a smartphone to scan a QR code, we usually get a link; more generally, we get a string of characters. In the context of QR codes, this string is called a message. Then, different methods apply to this message to generate the QR code. These methods are complex and are not the main goal of this assignment so we will ignore them. Summary, we can create a QR code from a message and retrieve the message from the QR code. One problem in interbank money transfers is the inability to understand other's QR codes because the format of each message is different for each bank. Therefore, the bank's application cannot parse the message into the necessary information and perform the interbank transfer. This information is the information related to a bank account to identify the recipient's account.

Students need to implement 2 class in this assignment: BankAccount and BankQR. The detail requirements will be described in two following sections.

2. Implement class BankAccount

Class BankAccount has attributes and methods related to a bank account:

- **Initial number**: A literal string as **"00020101021"**
- **Account number**: A string only contains numbers, its length is from 8 to 15 long depending on each bank
- **Account name**: A string only contains alphabetical letters (can be in uppercase or lowercase). An account name can have one or more words. In each word, the first letter must be in uppercase, the following letters must be in lowercase. A word must contain at least one uppercase letter and one lowercase letter. The words in the name are written consecutively, which means that there are no spaces between them.
    - Examples of correct account names:
        - “TranTrungTin”
    - Examples of incorrect account names:

  - "Tran Trung Tin": there is 1 space between "Tran" and "Trung", and 1 space between "Trung" and "Tin".
  - "TRANTRUNGTIN": all letters are in uppercase, we can separate the name into some valid words.
  - "TranTrungT": the last word (T) doesn't have any lowercase letters.
- **Bank name**: one of three strings: "VCBKB", "OCBKB", "ACBKB".

Note: In the following requirements, if the description does not specify where to do the implementation, students should write the answers to file QRBank.cpp.

Requirement 1.

Inspect class BankAccount in file QRBank.hpp and make sure you understand all attributes. No need to code in this requirement.

Requirement 2.

Implement two Constructors of class BankAccount:

- No-parameter Constructor: initialize all attributes with the value of "" (empty string).
- 4-parameters Constructor: initialize each attribute with the parameter that has the same name with it, given that all parameters will have valid values.

Requirement 3.

Implement 4 get methods and 4 set methods of class BankAccount. Note that you must check if the value is valid for the set methods before assigning it to the attribute. You should refer to the description at the beginning of this section for validation. You also need to read Requirement 5 about the specified length of the account number. The set methods should return true if the input value is valid; otherwise, return false.

Requirement 4.

The account number and account name should be encrypted when needed. The encryption is as follows:

a) Encryption of Account number

The method used to encrypt account number is similar to Vigenère Cipher. You can read about it at the link Mật mã Vigenère – Wikipedia tiếng Việt. The difference is we will use 10 digits (0-9) instead of 26 alphabetical letters (A-Z) to build the Vigenère Table.

Vigenère Cipher needs another information, keyword. Let *s* be the sum of all digits of account number. Let f = s % 3 + 3. We divide the account number into segments with equal size of f digits, starting from right to left. If the size of the last segment is less than f, we add some digits similar to the first digit until reaching f digits. Let k be the sum of all numbers in each segment. k is the keyword for encryption.

Example of finding keyword:

Account number	2129082021
Find s	(2+1+2+9+0+8+2+0+2+1)=27
Find f	(27 % 3) + 3 = 3
Divide account number into segments of f=3	|2|129|082|021|
Add number 2 to first segment	|222|129|082|021|
Find k	k = 222 + 129 + 82 + 21 = 454

You need to implement the method **getEncodedAccountNumber**:

`std::string getEncodedAccountNumber() const;`	
Input	+ No.
Output	+ Return: encrypted account number as above description.

b) Encryption of Account name

The method used to encrypt account number is similar to Caesar Cipher. You can read about it at the link Mật mã Caesar – Wikipedia tiếng Việt.

Caesar Cipher needs the shift d to do the encryption. Let s be the sum of all digits of the encrypted account number. Let p is the smallest prime and not less than s. Then:

$$d = p \ \% \ 52$$

We also use another alphabet:

A	a	B	b	C	c	…	Z	z

Example of encryption:

Given that d = 2	
Plaintext	ZzzBaaCzab
Encrypted text	AaaCbbDabc

You need to implement the method **getEncodedAccountName**:

`std::string getEncodedAccountName() const;`

Input	+ No
Output	+ Return: encrypted account name as above description.

Requirement 5.

In this requirement, students need to build the message for QR Code based on the information of the account and the created time of message. The following describes the detailed format of each bank.

- **VCBKB bank**

  Initial NumberBank NameAccount NumberAccount NameTime to create message

  With:

  - **Bank name**: "VCBKB"
  - **Account number**: Length of 13 numbers
  - **Time to create message**: A string of day, month and year in format **ddmmyyy**. E.g. date 22/08/2021 will be converted to "22082021"

  Example of VCBKB's messages:

  - 00020101021VCBKB0271001142475TranMinhHoang22082021
  - 00020101021VCBKB0491009260891NguyenVanBa26082021
  - 00020101021VCBKB0882003729910DangVanThanh18072020

- **OCBKB bank**

  Initial numberTime to create messageBank nameAccount numberAccount name

  With:

  - **Bank name**: "OCBKB"
  - **Account number**: Length of 15 numbers
  - **Time to create message**: A string of day, month, year in format **mmddyyyy**. E.g. date 22/08/2021 will be converted to "08222021"

  Examples of OCBKB's messages:

  - 0002010102108222021OCBKB012100002512112VoThiQuynhNhu
  - 0002010102107312021OCBKB013900001913490NguyenThinhToan

- 000201010211220202020OCBKB012100000801332HoangCongAnh

- **ACBKB bank**

Initial number	**Account Name**	**Account number**	**Time to create message**	**Bank name**

With:

- **Bank name**: "ACBKB"
- **Account number**: Length of 8 numbers
- **Time to create message**: A string of day, month and year in format **yymmdd** (yy is last two number of year). E.g. date 22/08/2021 will be converted to "210822".

Examples of ACBKB messages:

- 00020101021TrinhThiKhanhDuyen29136412200101ACBKB
- 00020101021HuynhNgocPhu14226317200706ACBKB
- 00020101021DangVanBinh15883012201130ACBKB

Implement method **buildMessageForQR** to build the message as above descriptions.

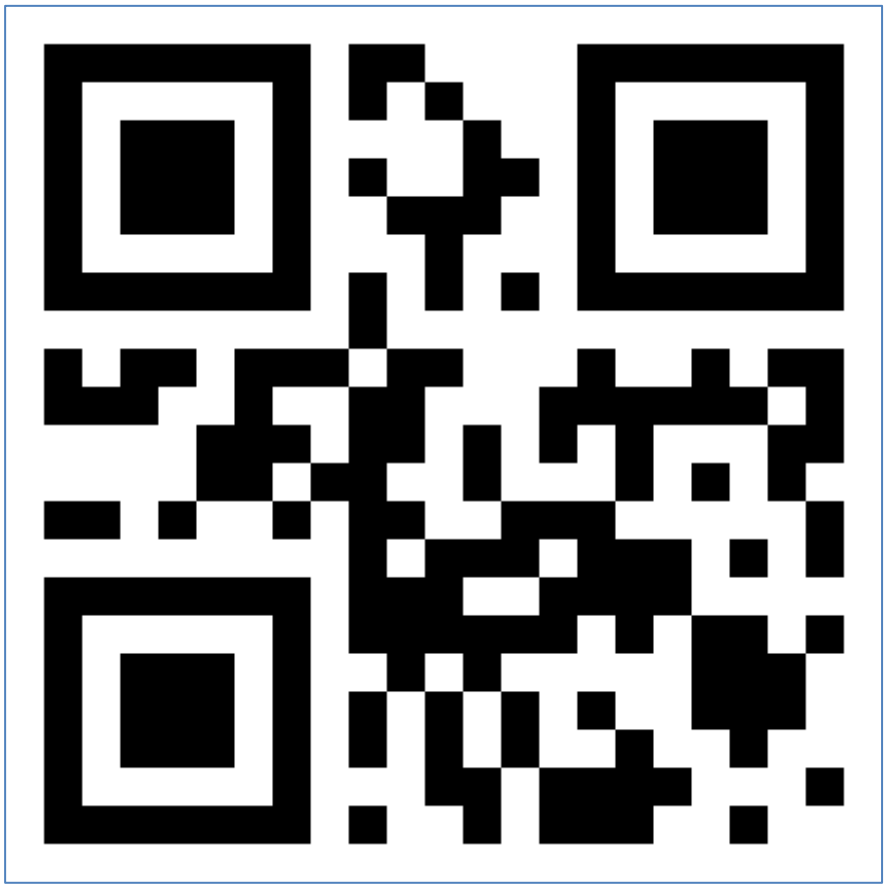
`std::string buildMessageForQR(const std::string & createdAt) const;`	
Input	+ createdAt: Time to create the message, the time is in the format of dd-mm-yyyy
Output	+ Return: Message
Note	• Students need to convert the value of createdAt to match the format of **Time to create message** of each bank. • For simplicity, the above examples illustrate with plain account number and account name. But these information need to be encrypted (as in Requirement 4) before assembling the message. Refer to the below cell for an example with encrypted information.
Ví dụ	An account has • Bank name: **VCBKB** • Initial number: **00020101021** • Account name: **TranTrungTin** • Account number: **1234567890123** Time to create message: **31-08-2021**
	Then we should have below results:

	• Encrypted account name: **cCcIVaQMV** • Encrypted account number: **2159659309374** • Method **buildMessageForQR** should return: **00020101021VCBKB2159659309374cCcIVaQMV31082021**

3. Implement class BankQR

Class BankQR has attributes and methods related to a QR. Assume that we only focus on the following information:

- Message to create the QR.
- Size of QR. QR code is a square, so we only need one value to represent its size.
- A two-dimensional array to represent the QR. Each element of the array is a pixel of white or black.



Requirement 6.

Inspect class BankQR in file QRBank.hpp. In access modifier private, you need to provide attributes for this class. Note that the requirements after this one are to implement methods. Here, you can give any attributes as you want, but you need to ensure that the

methods work correctly as the specification. You can write your proposed attributes, then revising them later.

Requirement 7.

Implement the Constructor of class BankQR.

`BankQR(string message)`	
Input	+ message: message to create QR Code.
Output	+ Return: (no)
Note	Initialize each attribute with a suitable value. The conversion from message to QR Code is implemented in function encodeQR in the directory helper/helper.hpp. Students should use the function in this task.

Requirement 8.

Implement Destructor of class BankQR. The Destructor must deallocate any Heap memory allocated by the Constructor.

`~BankQR()`	
Input	+ No
Output	+ Return: no

Requirement 9.

Implement method **saveToPNG** that writes a QR code to a PNG image.

`bool saveToPNG(std::string filename) const;`	
Input	+ filename: file name of the image (ends with ".png")
Output	+ Return: true if the file is written successfully, false otherwise. You should use function writeQRToPNG in the directory helper/helper.hpp for this task.

After completing this requirement, you can use a smartphone and scan the saved image to see the message of the QR code.

Requirement 10.

Implement method **toString**:

`string toString() const`	
Input	+ No
Output	+ Return: A string represents the QR code.
Description	• A white pixel is represented by a string with 2 spaces "", a black pixel is represented by a string with 2 hashtags "##". • There are no extra characters at the end of each line and the end of the returned string.
Example	Create an object of class BankQR with the message **"Hello world!!!"** The method toString should return

```
##############  ####        ##############
##          ##  ##  ##      ##          ##
##  ######  ##      ##    ##  ######  ##
##  ######  ##  ##    ####  ##  ######  ##
##  ######  ##    ######    ##  ######  ##
##          ##      ##      ##          ##
##############  ##  ##  ##  ##############
                ##
##  ####  ######  ####      ##    ##  ####
######    ##    ####      ############  ##
        ######  ####  ##  ##  ##      ####
        ####  ####    ##      ##  ##  ##
####  ##    ##  ####    ######          ##
                ##  ######  ######  ##  ##
##############  ######    ########
##          ##  ############  ##  ####  ##
##  ######  ##    ##  ##          ######
##  ######  ##  ##  ##  ##  ##    ######
##  ######  ##  ##  ##  ##    ##    ##
##          ##      ####  ########      ##
##############  ##    ##  ######    ##
```

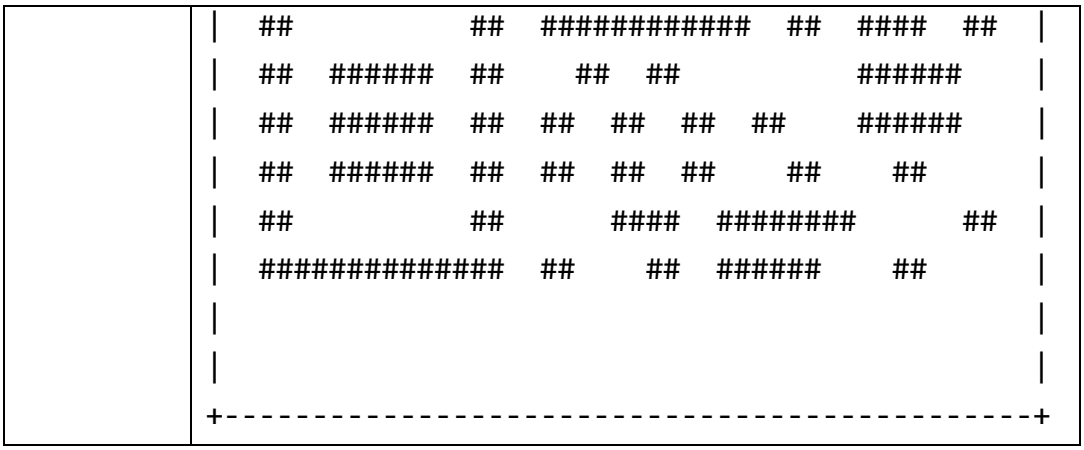
Requirement 11.

Implement the method **toString(int margin)**:

`string toString(int margin) const`

Input	+ margin: An integer represents the distance from each QR code edge to a border surrounding the QR. Testcases will ensure that the value of margin is not less than 0.
Output	+ Return: A string represents the QR code.
Description	The returned string is similar to the one in Requirement 10. But, there is a border surrounding the QR code (see Example). • ***Border***: Each character of the above border and below border is '-'. In left border and right border, each character is '|'. Except, 4 characters in 4 corners are '+'. • ***Distance to borders (margin)***: above border and above edge (of QR) are margin rows apart; below border and below edge are margin rows apart; left border and left edge are margin columns apart; and the same for right border and right edge; with each column corresponds to 1 space.
Example	• Create an object of class BankQR with the message **"Hello world!!!"** • margin = 2 This method should return as below

```
+-----------------------------------------------+
|                                               |
|                                               |
|  ###############  ####         ###############  |
|  ##           ##  ##  ##       ##           ##  |
|  ##  ######   ##        ##     ##  ######   ##  |
|  ##  ######   ##  ##     ####  ##  ######   ##  |
|  ##  ######   ##    ######     ##  ######   ##  |
|  ##           ##       ##      ##           ##  |
|  ###############  ##  ##  ##  ###############  |
|                   ##                          |
|  ##  ####  ######  ####       ##    ##  ####   |
|  ######    ##     ####       ############  ##  |
|          ######  ####  ##  ##  ##       ####   |
|          ####  ####     ##       ##  ##  ##     |
|  ####  ##    ##  ####    ######           ##   |
|                   ##  ######  ######  ##  ##   |
|  ###############  ######    ########           |
```



Requirement 12.

Implement method **saveStringTo** that writes the represented string of QR to a file.

`bool saveStringTo(std::string filename, int margin=0) const;`	
Input	+ filename: name of the written file. + margin: An integer represents the distance from each QR code edge to a border surrounding the QR. Testcases will ensure that the value of margin is not less than 0.
Output	+ Return: true if the file is written successfully, false otherwise.
Description	Write to file a representation string similar to Requirement 11.

Requirement 13.

Implement method **decodePNG** that decodes a PNG image of QR code to the message.

`static std::string decodePNG(std::string filename);`	
Input	+ filename: Name of the reading image (ends with ".png")
Output	+ Return: Message of QR code.

You need to explore class QRCodeDetector of OpenCV (version 4.5.2) to complete this method.

Node: The installation of OpenCV on Windows from the Internet often instructs you to install using Visual Studio. We want to introduce you to another way: find out MSYS2 and install OpenCV by it.

Requirement 14.

Implement method **decodeInfoOfMessage** that extracts the information from the message and displays them.

`static std::string decodeInfoOfMessage(const std::string & message);`	
Input	+ message: Message needs to be extracted.
Output	+ Return: A string looks like the below cell.
Description	Ngan hang: <**Bank name**> So tai khoan: <**Encrypted account number**> Ten chu tai khoan: <**Encrypted account name**> Thoi diem tao thong diep QR: <**Time to create message**>
	Lưu ý: • **Bank name**, **Encrypted account number**, **Encrypted account name** is as in Implement class BankAccount. • **Time to create message** is in the format of dd/mm/yyyy, e.g.: 29/08/2021 • If the time only has 2 digits representing for year part, two digits **20** will be added to the beginning to create an entire year. E.g. If the time in the message is "220821", the corresponding string for time will be "21/08/2022" • There is a space after the colon (:) in the returned string.
Ví dụ	• We need to extract information from the following message: **00020101021VCBKB2159659309374cCcIVaQMV01092021** This method should return as below cell

	`Ngan hang: VCBKB`
	`So tai khoan: 2159659309374`
	`Ten chu tai khoan: cCcIVaQMV`
	`Thoi diem tao thong diep QR: 01/09/2021`

Requirement 15.

Most mobile banking apps only recognize the QR of the same bank. Then, they can't support transferring money from a bank to the different one. In this requirement, you need to convert the original QR code to a new QR code that can be read from the target bank.

`static BankQR convert(const BankQR & inQR, const std::string & targetBank);`	
Input	+ inQR: QR code need to be converted. + targetBank: Name of the bank that this QR code needs to be

	converted to its format.
Output	+ Return: New QR code in the format of target bank.

4. Other instructions

- You are not allowed to include any extra libraries except the existing ones in the initial codes.
- You are not allowed to change the first four (4) lines of file QRBank.hpp, and the first nine (9) lines of file QRBank.cpp.
- Given that the name of excutable file is *main.cpp*, the compilation command is:

```
g++ -std=c++17 -o main -lstdc++fs -lopencv_objdetect -
lopencv_imgcodecs -lopencv_core -I <directory to OpenCV
header> main.cpp helper/*.cpp helper/libs/*.cpp -I
helper/*.hpp -I helper/libs/*.hpp
```

Then, we can run the program by: **`./main`**

5. Submission

Students download the file *Do_an_KTLT_assigment.zip* from the course site and extract it, you will see the following files:

[FP-Project]Assignment-QRCode.pdf		Assignment description
Folder *initial_code*	*QRBank.hpp*	The header file contains the declaration of class BankAccount and BankQR
	QRBank.cpp	Implementation file
	Folder *helper*	Some necessary functions and libraries.

Students submit 2 files on site e-Learning of course: **`QRBank.hpp`** and **`QRBank.cpp`**.

The deadline for submission is announced at the submission site. By the deadline for submission, the link will be locked automatically, so students will not be able to submit them late. To avoid possible risks at the submission time, students MUST submit their files at least **one hour** before the deadline.

6. Handling fraud

Assignment must be done BY YOURSELF. Students will be considered fraudulent if:

- There is an unusual similarity between the source code of the submissions. In this case, ALL submissions are considered fraudulent. Therefore, students must protect the source code of their assignments.
- Students do not understand the source code written by themselves, except for the parts of the code provided in the initialization program. Students can consult from any source, but make sure they understand the meaning of all the lines they write. In the case of not understanding the source code of the place they refer to, students are especially warned NOT to use this source code; instead use what has been learned to write programs.
- Mistakenly submit another student's assignment on your personal account.

In the case of cheating, students will get a 0 for the entire subject (not just the assignment).

DO NOT ACCEPT ANY INTERPRETATION AND NO EXCEPTION!

After each major assignment has been submitted, a number of students will be called for random interviews to prove that the assignment has been done by themselves.

---END---