**CRYPTOGRAPHY TERM PROJECT BITS F463**

**Blockchain based Credit Scoring**

### @Team Members

Atishya Gupta - 2017A7PS1482H

V Rachana Srikruthi - 2017A7PS0086H V Chandrahaas - 2015A7PS0010H

R Hariharan - 2017A7PS0065H

**Source file:** Crypto.cpp

## Functions:

1. verifyTransaction()
2. mineBlock()
3. createBlock()
4. viewUser()

# verifyTransaction():

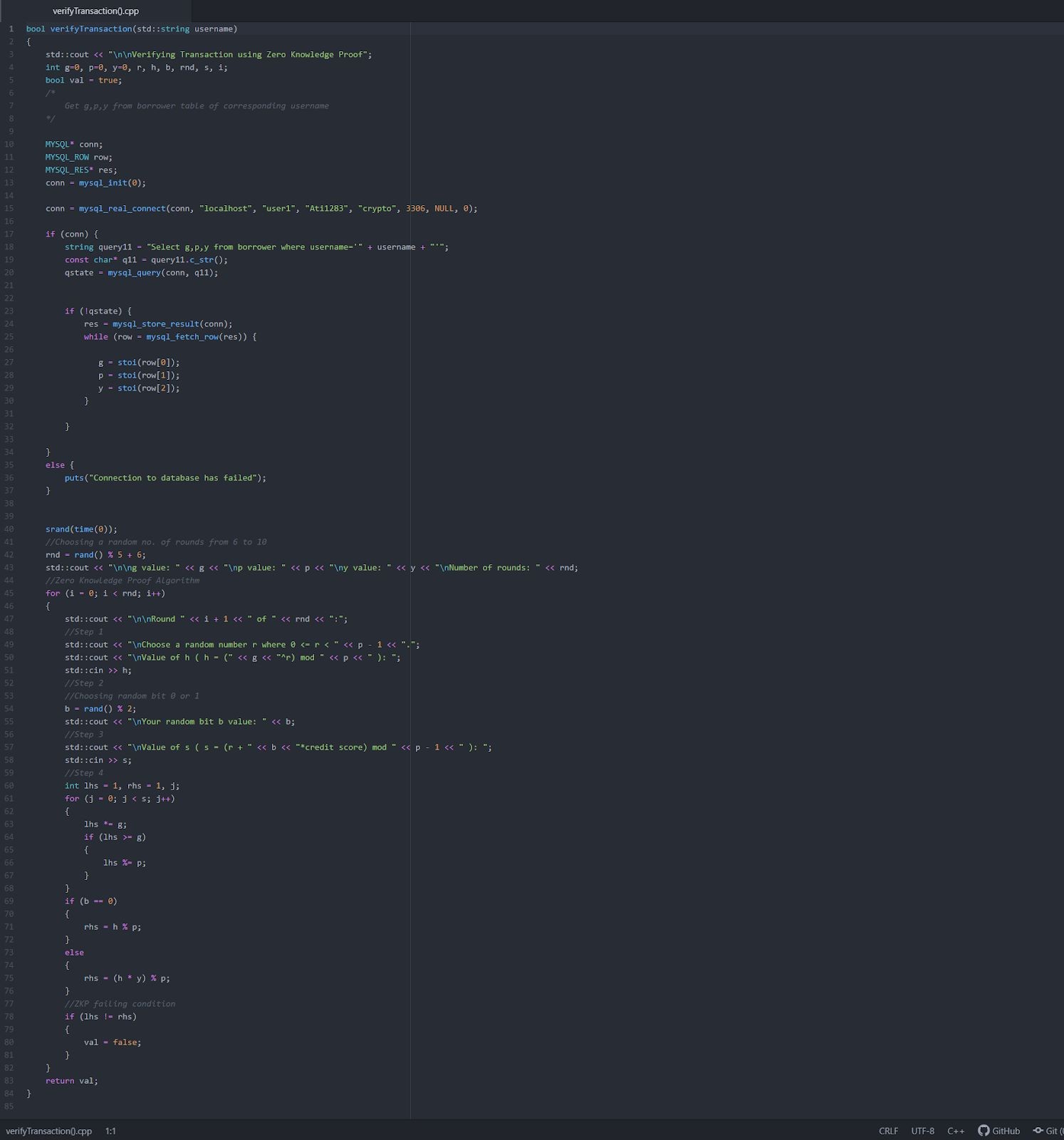
The function is used for the Zero Knowledge Proof(ZKP) to verify the credit score of the borrower without having him to reveal it to everyone hence making the overall transaction process safe.

For ZKP: g, p are primes where g is the generator of prime p. y = gxmod(p) where x is the credit score of the borrower.

g, p and y are chosen by the user at the time of registration.

The numbers g,p,y are displayed to the borrower when he requests a new loan in order to verify through ZKP.

* A random number between 6 to 10 is generated for doing multiple rounds of ZKP to make sure of the correctness.
* Borrower has to choose a random number between **0<=r<p-1** and sends it as **h=grmod(p)**
* A random bit either **0** or **1** is returned back.
* Borrower sends **s**= (r+bx)mod(p-1)
* Number **gsmod(p)** is computed which should be equal to **hybmod(p)** else it means that borrower is not aware of his credit score or is fake and after completion of rounds **security breach** will be printed and program exits without feeding the request of borrower.



# mineBlock():

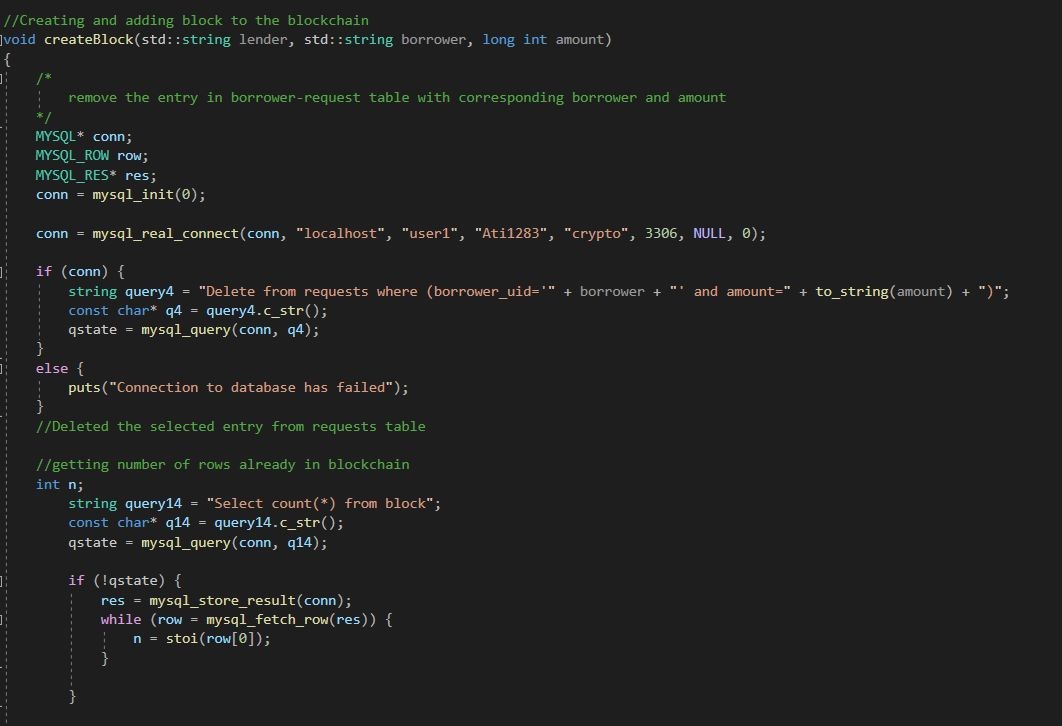
This function is called after the **ZKP** has been successfully completed. The borrower has to decrypt a puzzle which is a hash value and is displayed right after ZKP is completed, the borrower has to decrypt (solve) the hash value in order to get a successfully transacted block.



The input-hash pairs are given in this file: <https://drive.google.com/file/d/17aWz7Wm3AMp9QSRNSJjzGSlhlv_FIA5q/view?usp=sharing> Input is the decrypted hash value which can be used to check the correctness of proof of work.

# createBlock():

The function is called after the lender has chosen the request of the borrower to grant the loan. The request which has been chosen by the lender is deleted from the requests table which is necessary as the transaction for that request is successfully completed and is no longer pending.

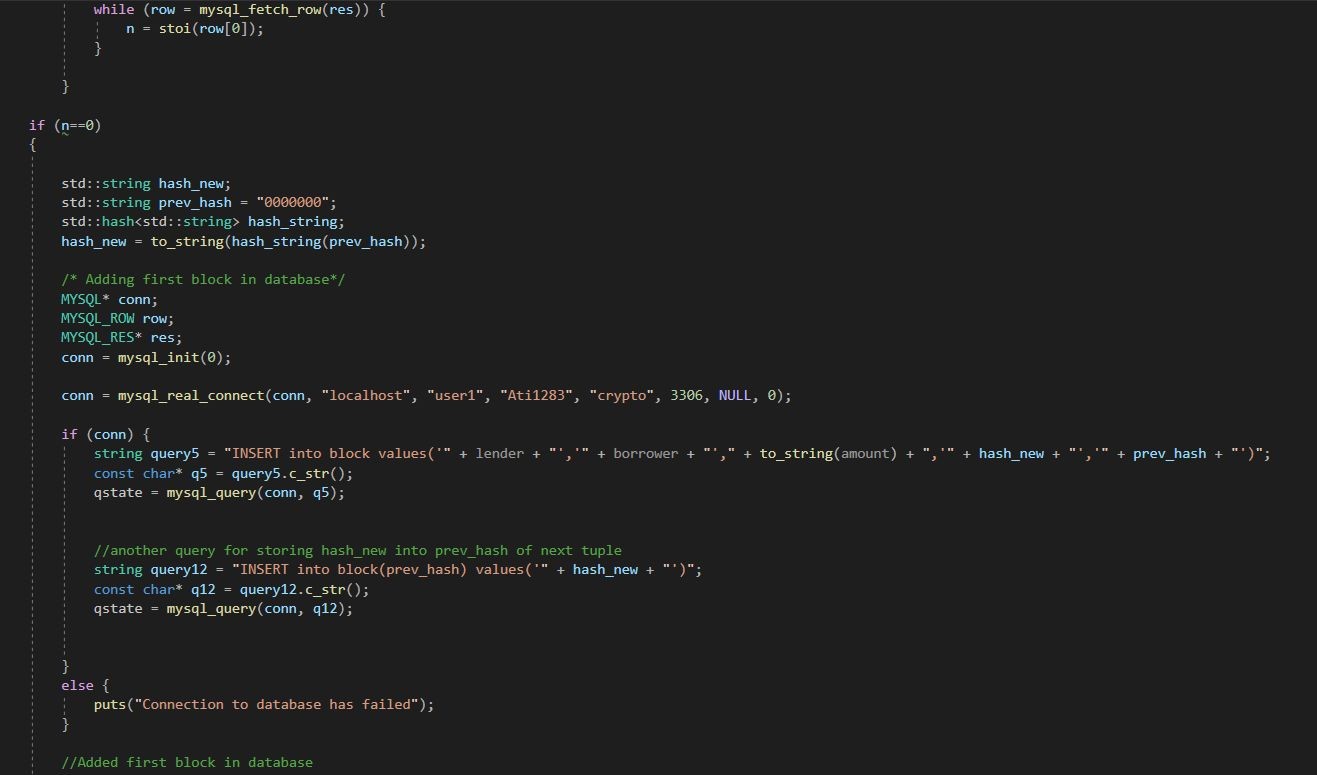


Now, after deleting entry from request table, to add a block in the blockchain: There are two cases:

* 1. when the blockchain is empty and there is no transaction occurred before. For this the prev\_hash value is taken as ‘0000000’ to generate the hash value as there is no past block from which the prev\_hash value can be fetched. Now, for future transaction, the hash value of this created block can be used as prev\_hash.
  2. For any transaction after the first block has been created the hash value of the last transaction in the blockchain can be used as prev\_hash to generate its hash value which then can be used as prev\_hash for the next block.

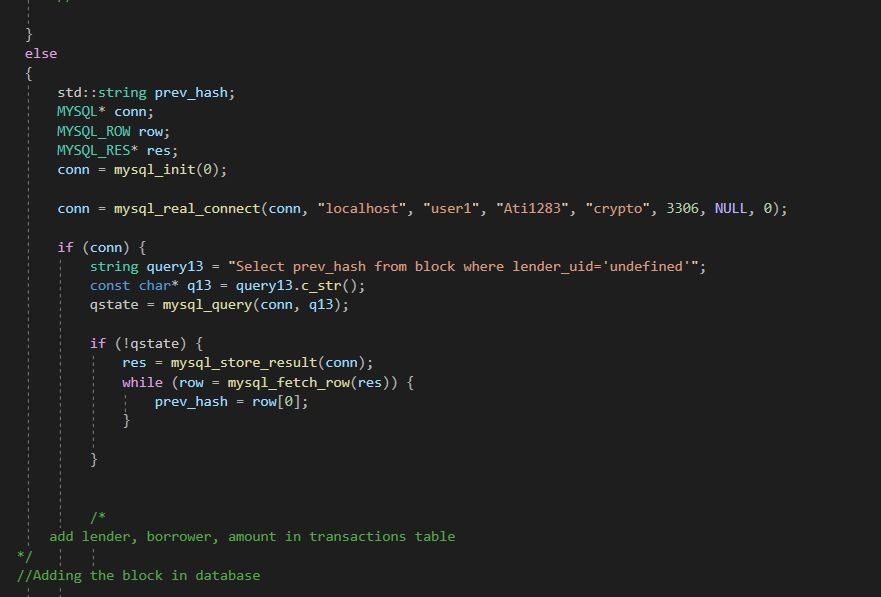
**Case I**. code snippet:

\*\*when no transactions occurred ‘0000000’ taken as prev\_hash to generate hash value.



**Case II**. code snippet:

\*\*To get the hash value of previous block



\*\* Calculating hash value from prev\_hash and to feed the values of the current transaction

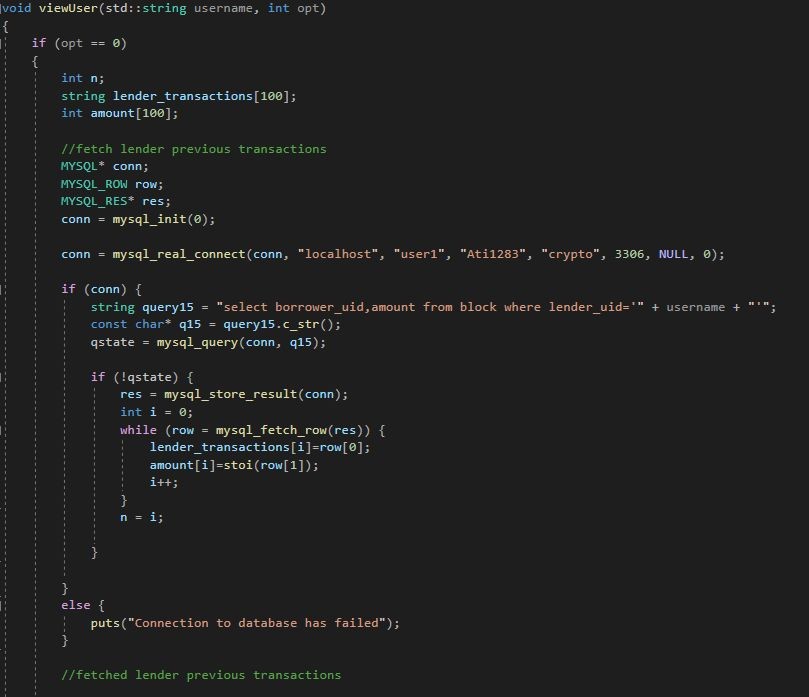


* **viewUser()**

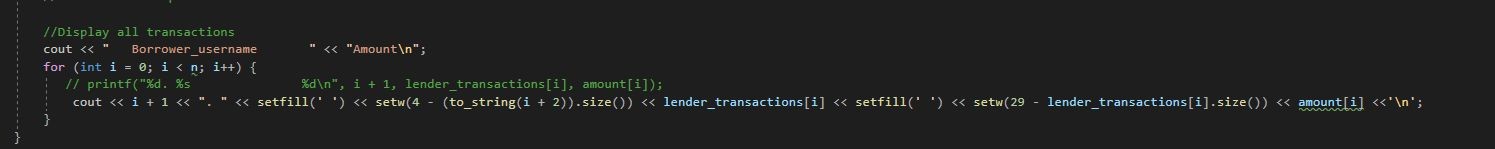
This function is used to view the previous transactions of borrower or lender. Both lender and borrower are provided the functionalities to view their previous transactions. This is done by using SQL query in the **blocks** table and then displaying the info fetched.

Note: [opt=0 means lender wants to view transaction and opt opt=1 means borrower wants to view previous transaction]

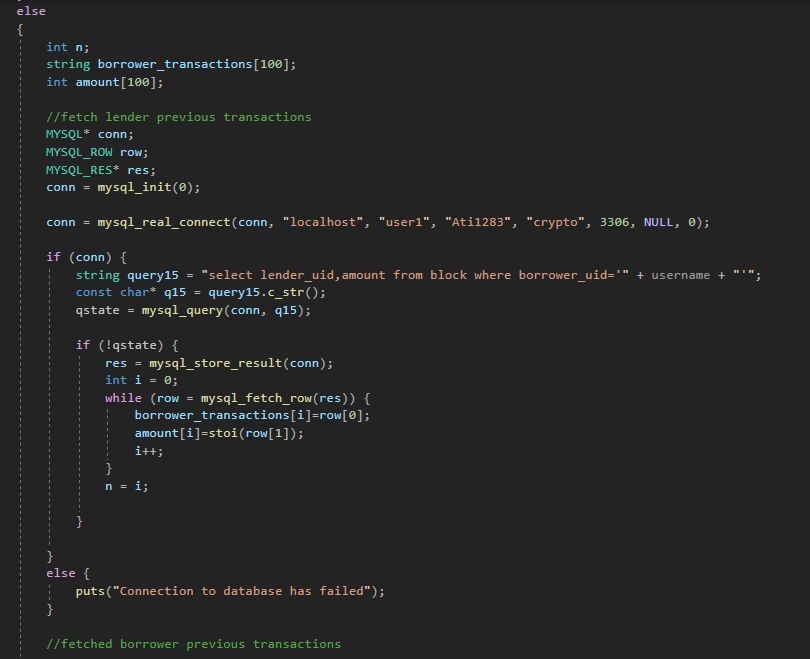
**For lender:**

\*\*Querying the blocks table to get lender previous transactions

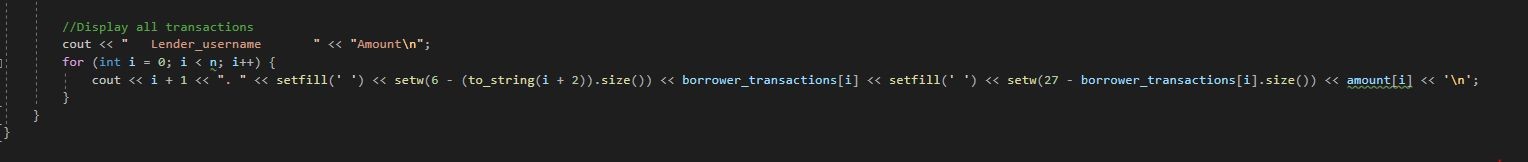
\*\*Displaying the transactions



**For borrower:**

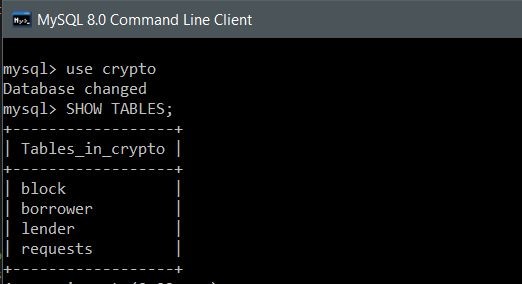
\*\*Querying the blocks table to get borrower previous transactions

\*\*Displaying the transactions fetched.

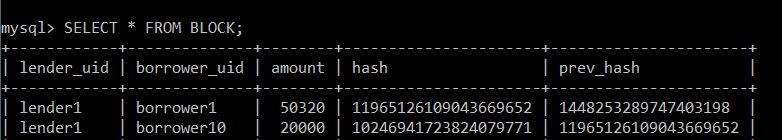


## DATABASE TABLES USED

Following database tables has been used:

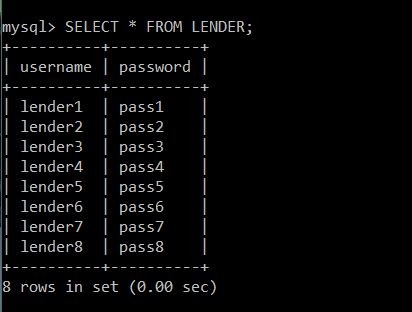
1. Block
2. Lender
3. Borrower
4. Requests
5. BLOCK

Block table stores the transactions that are completed successfully. It stores the information of the **lender username** (which grants the loan), **borrower username**(whom the loan has been granted), **amount**(Loan amount), **hash** (this keeps the data intact and safe from getting modified by anyone, if any one of the past transaction hash value is changed then it causes ripple effect and the hashes of all further blocks are affected because hash value of one block is used for the generation of hash value for the next block, Prev\_hash (the hash value of previous block which is used to generate hash value for current block)



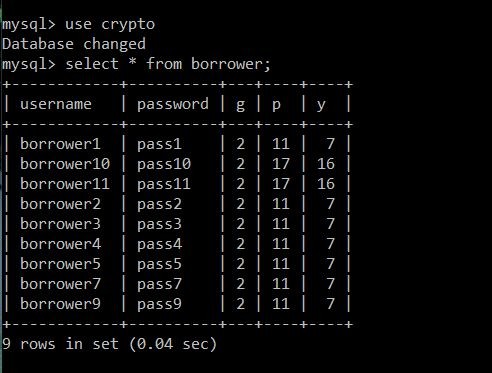
1. LENDER

Lender table stores the login credentials as **username** and **password** of the customers of the blockchain that are interested in lending money. Any new registrations as a lender are added to this table.



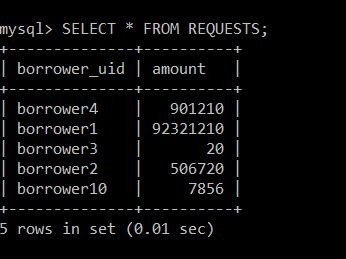
1. BORROWER

Borrower table stores the login credentials as **username** and **password** of the customer of the blockchain who needs loan and any new registrations as borrower are added to this table.



1. REQUESTS

Requests table stores the information of the pending loan requests created by the borrowers that are to be displayed to the lenders. It stored the **borrower\_uid** who has created the loan request and the **amount** demanded.



**FUNCTIONALITIES**

After building and running the source code the **Blockchain Loan Interface** is displayed which asks the user to login/register as: 1. Lender

2. Borrower

The user has to choose code as **‘**1**’** or ‘2’ depending on if the user is a lender or borrower.

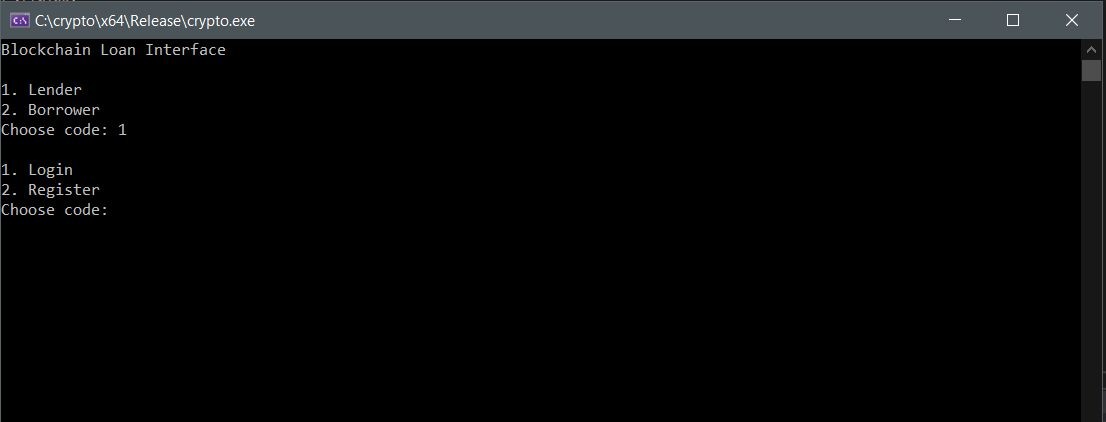
### LENDER OR BORROWER

Here, for now we proceed by choosing code ‘1’ i.e. as a lender to show functionalities of a lender.

### LOGIN OR REGISTER

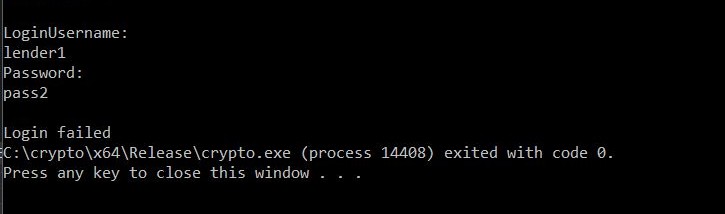
After choosing to login as a lender or borrower, the user is asked to **Login/Registe**r depending on if he is an existing customer or a new customer.

Users have to choose ‘1’ for login, ‘2’ for Register.



### LOGGED IN AS LENDER

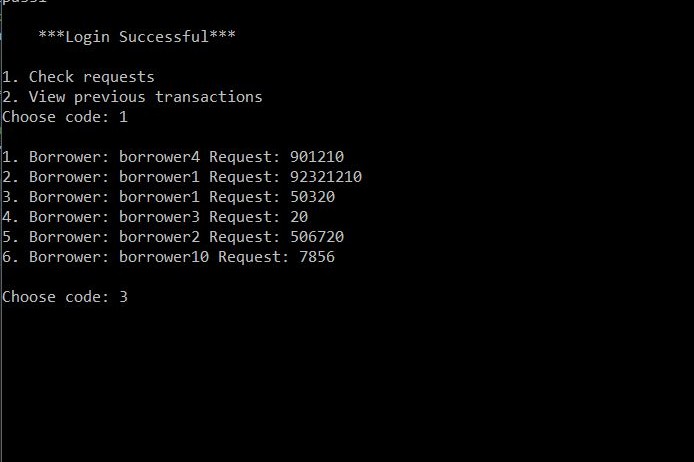
* + - If the entered credentials are incorrect then login failed will be displayed and the program will exit.



If the entered credentials are correct then the lender is shown to choose code ‘1’ or ‘2’ depending on if the user wants to **check the requests** or wants to **view previous transactions** respectively.

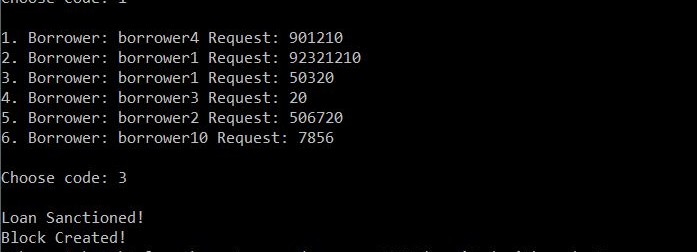
### Code ‘1’ :

Lender is shown the requests of the borrowers with **Borrower username** and the **loan amount requested** and he can select on his own the request which lender wants to grant loan (lend money) by entering the corresponding request no.



When the lender has selected the corresponding request no. to grant loan amount **createBlock()**

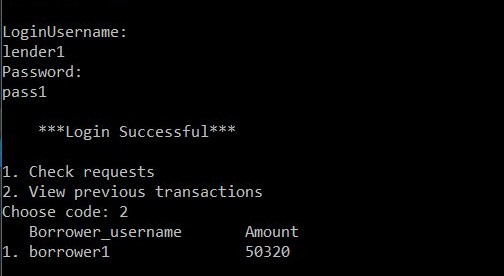
function is called to create a new block in the blockchain with the transaction details.



### Code ‘2’:

Lender is shown his previous transactions with detailed description of each transaction showing the borrower username to whom the loan was granted and the corresponding loan amount.

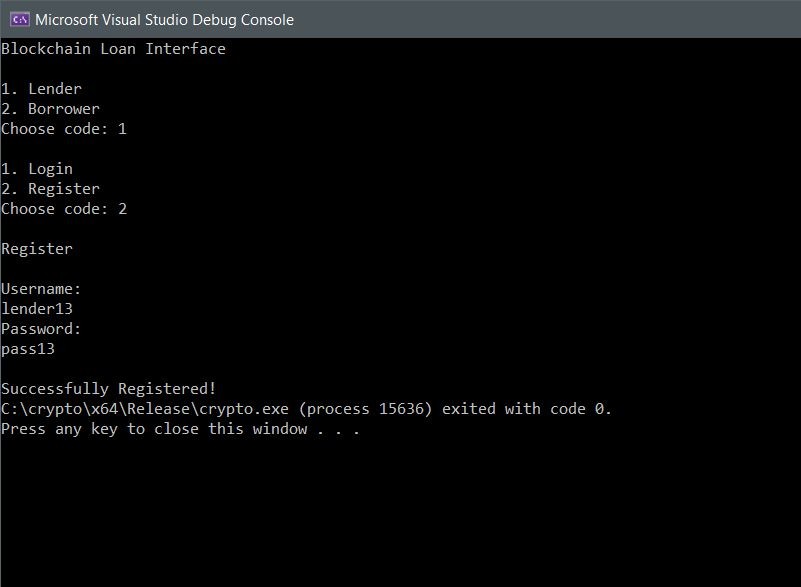
**viewUser()** Function is used to complete this task.



### LENDER - REGISTRATION

The lender chooses code ‘2’ if he wants to register into the interface for the first time. Once registration is done successfully, the lender’s details are added to the lender table.

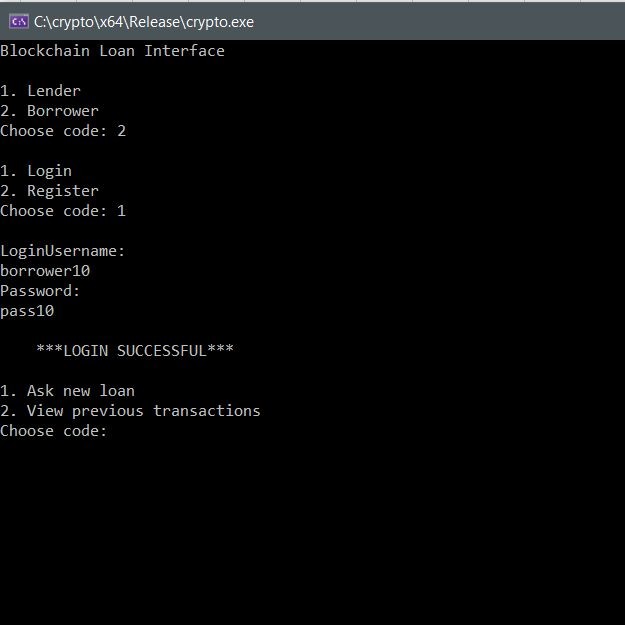
The screenshot of this registration process:



### LOGGED IN AS BORROWER

If in Blockchain Interface, code ‘2’ is chosen, then the user chooses to operate the interface as the borrower.

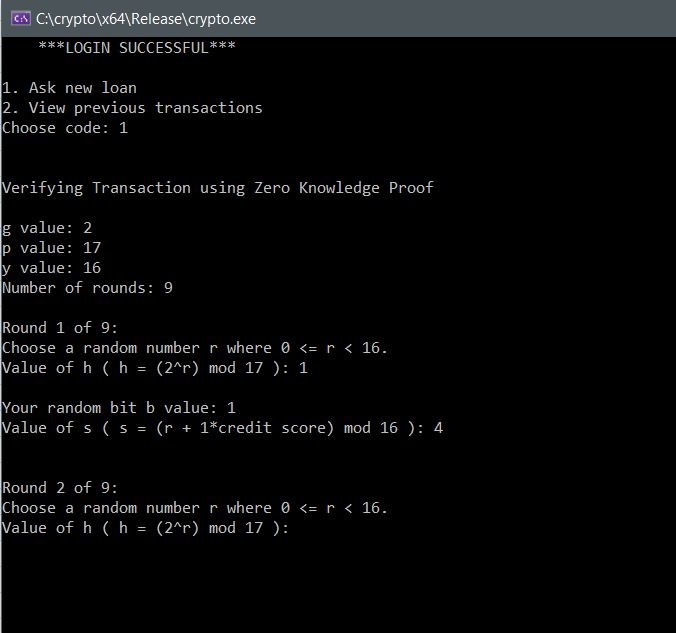
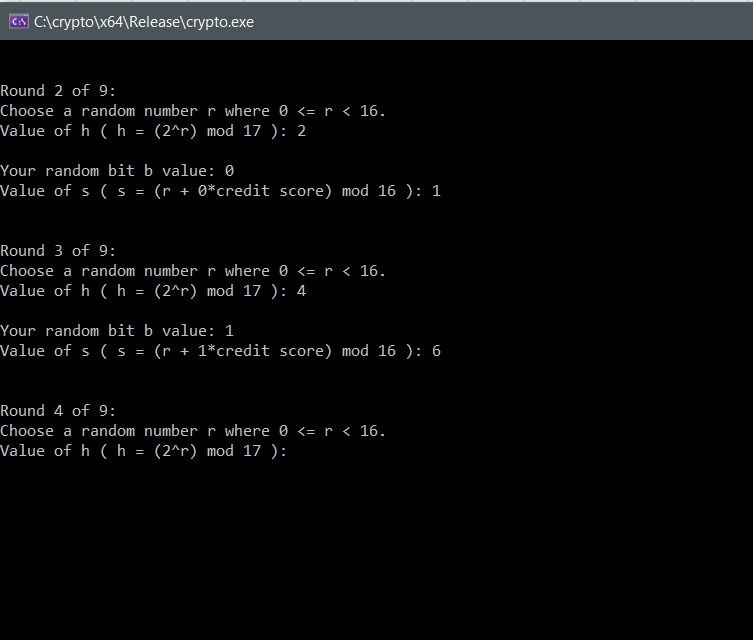
After that, if the user chooses code ‘1’ for login and if it is a success, the user gets 2 options to choose from: code ‘1’ to **ask for a new loan**, code ‘2’ to **check previous transactions.**

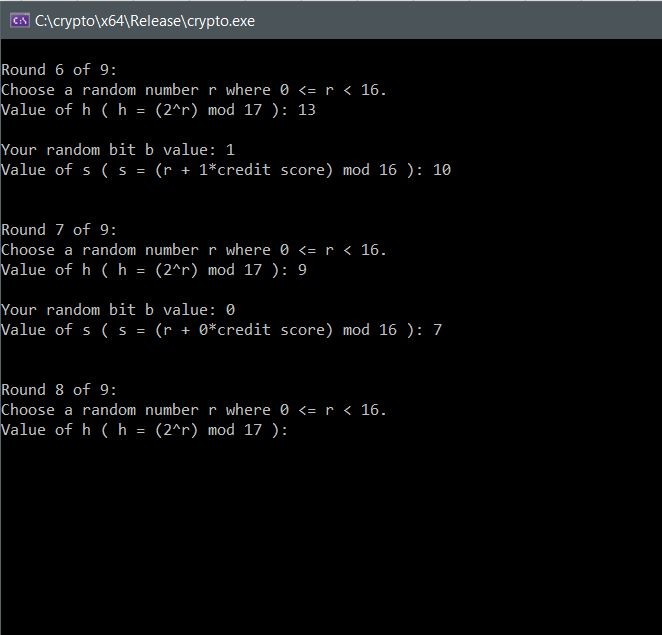


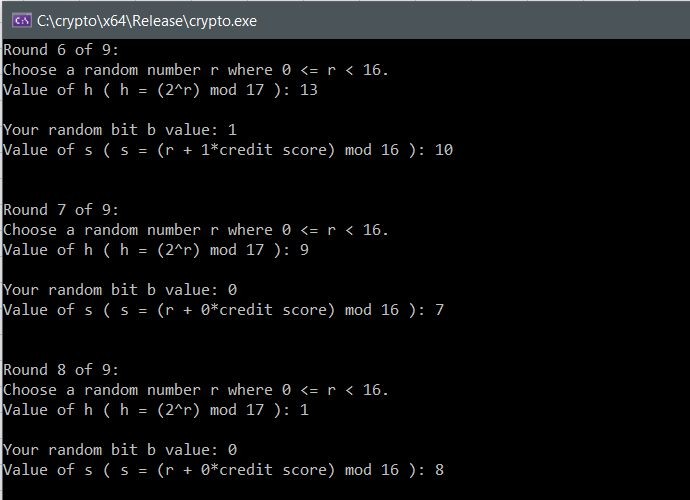
### Code 1: Ask new loan

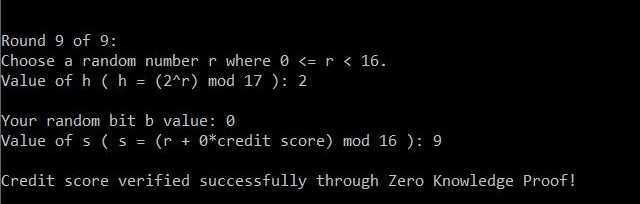
Whenever the borrower asks for a new loan, he should get the **y value of his credit score verified through zero-knowledge proof** first. The blockchain interface acts as the verifier and the borrower acts as the prover. The interface does the zero-knowledge proof for a random number of 6 to 10 rounds to ensure that **the borrower is not fake.** This is carried out by **verifyTransaction()** function.

The following are the screenshots of the zero-knowledge proof carried out for a random number of 9 rounds:





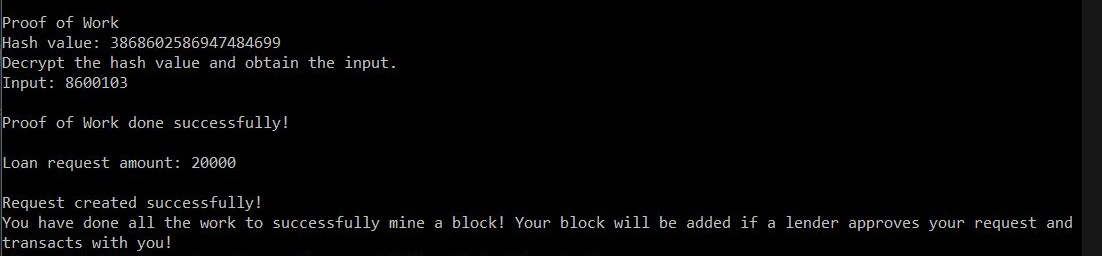




After the zero-knowledge proof is verified successfully, the borrower must **solve proof of work** (decrypt the hash value, in this case) successfully to become **eligible to mine a block.** This is carried out by the **mineBlock()** function. Once the proof of work is done successfully, the borrower becomes eligible to mine a block. Then, he can enter the loan amount that he wants and a new request is created with his

name and his requested loan amount. This request is placed in the request table.

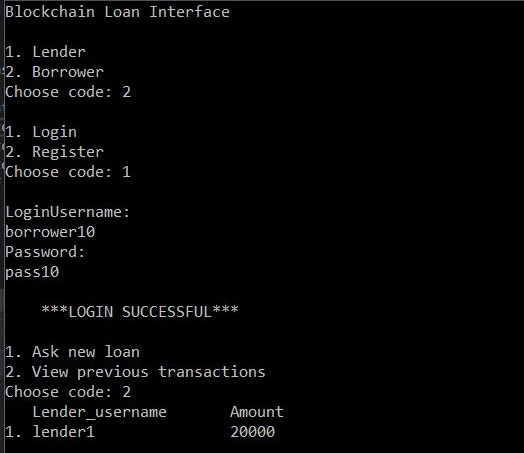
The screenshot of this process:



### Code 2: View Previous Transactions

The borrower can view his previous transactions. This is carried out by the **viewUser()**

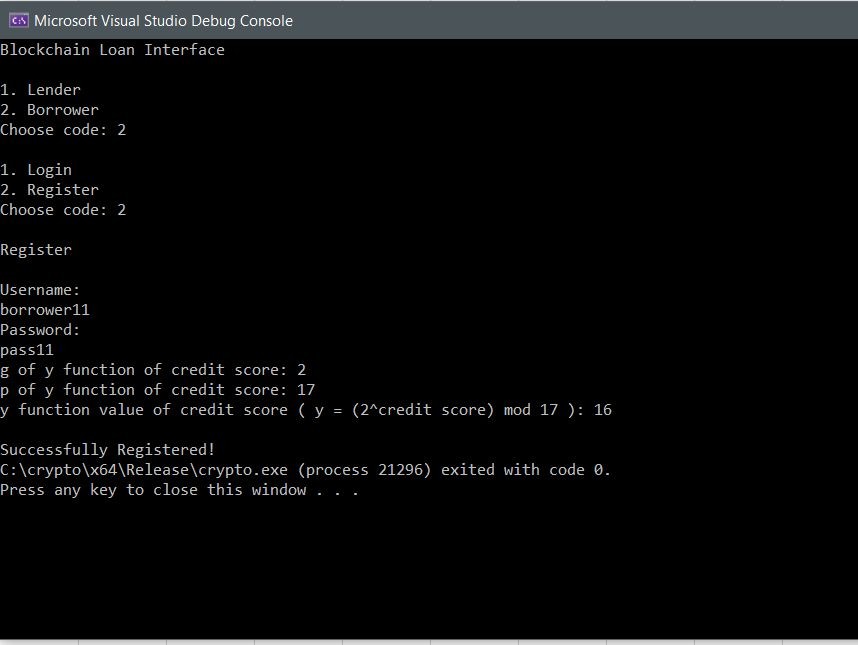
function.



### BORROWER - REGISTRATION

The borrower chooses code ‘2’ if he wants to register into the interface for the first time. While doing the registration, the borrower enters his g value, p value and the y value of his credit score. Once registration is done successfully, all of the borrower’s details are added to the borrower table.

The screenshot of this registration process:



## CONCLUSION:

* This project has been instrumental in understanding and learning the concept of blockchain, meanings of various new terms like zero-knowledge proof, proof of work, etc.
* We also learnt how to successfully implement blockchain and use it to solve real-life problems like security of credit scores and decentralisation of loan transactions,microtransactions.