BTCOrigins

CSC2007 Mobile Application Development Spring 2022

Some background…

Ok none of the previous apps you built worked out for me… So, I am going back to first principles and will investigate the origins of Bitcoin, with your help of course.

We will attempt to understand how this futuristic money works by building a blockchain mobile app that will mine new blocks containing arbitrary transactional messages. We'll need to do it in C because, c'mon we're not intending to code the ultra-intensive Proof-of-Work (mining) computation in Kotlin right?

Fully understanding Bitcoin mining is not necessary for this quiz but if you somehow found loads of time left when you are done with the quiz, feel free to read Satoshi's original paper[1], or the endless articles when you google "how bitcoin works".

The essential concept is that the Bitcoin blockchain is very similar to a linked list of nodes (blocks) containing coin transaction data, with the "link" being a cryptographic hash of the previous block's header rather than a simple pointer.

1. When a new transaction occurs (in this quiz, the "transaction" data is simply a message), a header is constructed for the block of data (see `BlockHeader` struct in *blockchain.h*). The most obvious thing that the header needs to do is to have a record of the transaction data, stored as a **SHA256 hash** of the data itself, called **`dataHash`** in the `BlockHeader`.
2. To gain the right to add this block to the chain, the app (miner) needs to keep generating (mine) a **SHA256 hash** (see *sha-256.h*) of the entire header object until a *valid hash* appears that is smaller than a `targetHash` configured based on a network **difficulty** value provided. The higher the difficulty, the longer it takes.
3. A **`timestamp`** and **`nonce`** (the iteration count) is updated in the header as new hashes (of the header) are tried iteratively, so that the output hash is different each time in the search for a valid hash (see `mine` function in *blockchain.c*).
4. Once the valid hash appears, the last timestamp and nonce is recorded in the current header and the block is officially added to the chain by setting the **`previousHeaderHash`** in the current header, to the hash of the previous header.

The above steps are mostly already done for you in *blockchain.c*.

[1] https://bitcoin.org/bitcoin.pdf

What needs to be done

Fork the repo **csc2007-quiz03-2022** and inspect the code within the project. Examine the skeleton code before working on the lab. Refer to the screenshots on the following page for the following descriptions. The user story this time is…

As a promising future Bitcoin maximalist,
I want to have an app that allows me to add blocks containing arbitrary transaction messages to a simplified Bitcoin blockchain network based on a network difficulty that sets the Proof-of-Work intensity,
So that I can understand the core of what makes Bitcoin such a pristine asset and be confident to dump all my life savings in it[2].

Acceptance Criteria:

1. (1 mark) See a view that is very similar to that shown across the various Figures. It should contain AT LEAST
    a. an EditText (id **blocksEditText**)
    b. an EditText (id **difficultyEditText**)
    c. a multiline EditText (id **msgEditText**)
    d. a Button (id **genesisButton**)
    e. a Button (id **chainButton**)
    f. a TextView (id **dataHashTextView**)
    g. a TextView (id **logTextView**)
2. (1 mark) When the Buttons are clicked, the app will perform the following validation to ensure the right user inputs to the various UI elements above (Figure 1). In essence, **genesisButton** only requires **difficultyEditText** to be filled properly, whereas **chainButton** requires all input fields to be filled.
    a. Input only numbers in the EditText (id **difficultyEditText**). If this field is empty when **genesisButton** or **chainButton** is clicked, a message should be included in TextView (id **logTextView**) that reads **"difficulty cannot be empty…".** The number must be between 1 to 10. If the number is outside of these bounds when **genesisButton** or **chainButton** is clicked, a message should be included in TextView (id **logTextView**) that reads **"difficulty must be 1 to 10…"** (Figure 1)

[2] Nothing specified in this quiz constitutes financial advice.

b. Input up to 10 lines of text in EditText (id **msgEditText**). If this field is empty when **chainButton** is clicked, a message should be included in TextView (id **logTextView**) that reads **“describe your transaction in words…”** (Figure 1)

c. Input only numbers in the EditText (id **blocksEditText**). If this field is empty when **chainButton** is clicked, a message should be included in TextView (id **logTextView**) that reads **“blocks cannot be empty...”.** The number must be between 2 to 888. If the number is outside of these bounds when **chainButton** is clicked, a message should be included in TextView (id **logTextView**) that reads **“blocks must be 2 to 888...”** (Figure 1)

3. (1 mark) When I click on **chainButton**, verify that the native code received correct inputs by seeing logs of the input **difficulty** and transaction **message** values being printed in the Android console at the INFO level with the TAG set to “BTCONATIVE” (Figure 3). The logging should be performed in native code.

4. (1 mark) When I click on **genesisButton**, only one block is mined, the Genesis block. I should **see a log printed in the Android console that indicates that it is a GENESIS block** formatted alongside the hashes (Figure 2). The logging code has already been written in *blockchain.c* . Don’t change it; you just need to figure out where/how/when to call it. Basically you need to create and call a native function that calls `addBlockWithPrevPtr` *blockchain.c*, with the arguments:

   - `prevHeader:` empty object (i.e., the `nullptr`)
   - `data: "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"`
     (this is the actual message of the first Bitcoin block)
   - `length:` the number of chars of the `data` above including the terminating null byte (`'\0'`)
   - `difficulty:` number obtained from the EditText (id **difficultyEditText**)

   Note that all native libs are supplied in: ***app/src/main/cpp***

5. (1 Mark) When I click on **chainButton**, a blockchain is mined. The number of blocks provided in **blocksEditText** will be mined with the given text in **msgEditText** repeated for each block. I should see **logs printed in the Android console that indicates that there are BLOCKs**, formatted alongside the hashes (Figure 3).

   E.g., If **3** was input from **blocksEditText**, I should see 1 “addBlockWithPrevPtr:GENESIS:<dataHash>:<previousHeaderHash>” and 2 “addBlockWithPrevPtr:BLOCK:<dataHash>:<previousHeaderHash>” logs printed in the Android console, with different dataHash and previousHeaderHash values. (Figure 3). Again, the logging code has already been written so don’t change it.

   The pseudocode should be something like:

```
mine the Genesis block
set prevHeader = Genesis block's header
set msg = text as a C-style string from msgEditText
set numBlocks = number from blocksEditText
for i in 1 to numBlocks:
      mine ith block with prevHeader and msg
      set currHeader = ith block's header
      set prevHeader = currHeader
```

(This criteria may not be trivial, so skip first if you are stuck.)

6. (1 mark) See the correct **dataHash** in the TextView (id **dataHashTextView**) (some time) after the **genesisButton** or **chainButton** is clicked (Figures 2/3/4). This `dataHash` is the hash of the data (the transaction message) as recorded in the last block header.
    a. For the **chainButton**, the `dataHash` is the hash of the data provided in the EditText (id **msgEditText**).
    b. For the **genesisButton**, the `dataHash` is the hash of `"The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"`.
    c. This `dataHash` is already computed in the constructed `BlockHeader` after `addBlockWithPrevHash` completes. You just need to figure out how to get this into Kotlin to display in the view.
    d. The `dataHash` in the `BlockHeader` is of type `uint8_t` which each array element representing a byte (8-bit) of data. Before you pass it back to Kotlin, you may need to convert it into the common hex representation we humans normally see for hashes. There is a util function in *blockchain.c* called `makeCStringFromBytes`, and you can read the function documentation to decide whether you want to use it.
7. (1 mark) When **genesisButton** or **chainButton** is clicked, the network **difficulty** from the EditText (id **difficultyEditText**) will be relayed to `addBlockWithPrevPtr` and eventually used in `mine` in *blockchain.c* to determine the right `targetHash` for mining. This will result in different times taken to complete the mining task and receive the hashes in the TextViews. Large values will increase the time taken (Figures 2/3/4) as expected by the different target hashes set in the `mine` function.
8. (1 mark) The UI should not lag, freeze or ANR at any point. Note that difficulty values higher than 6 will likely be extremely (I mean really extremely) computationally intensive, unless of course you brought your Antminer S19 Pro in a box to class today.
9. (1 mark) After **every block** mined after clicking **genesisButton** or **chainButton**, the time taken (in milliseconds) of the mined block's header is shown in the TextView

(**logTextView**) (Figures 2/3/4). The time taken is from the time you click the Button, to when the hashes are shown in the TextViews.

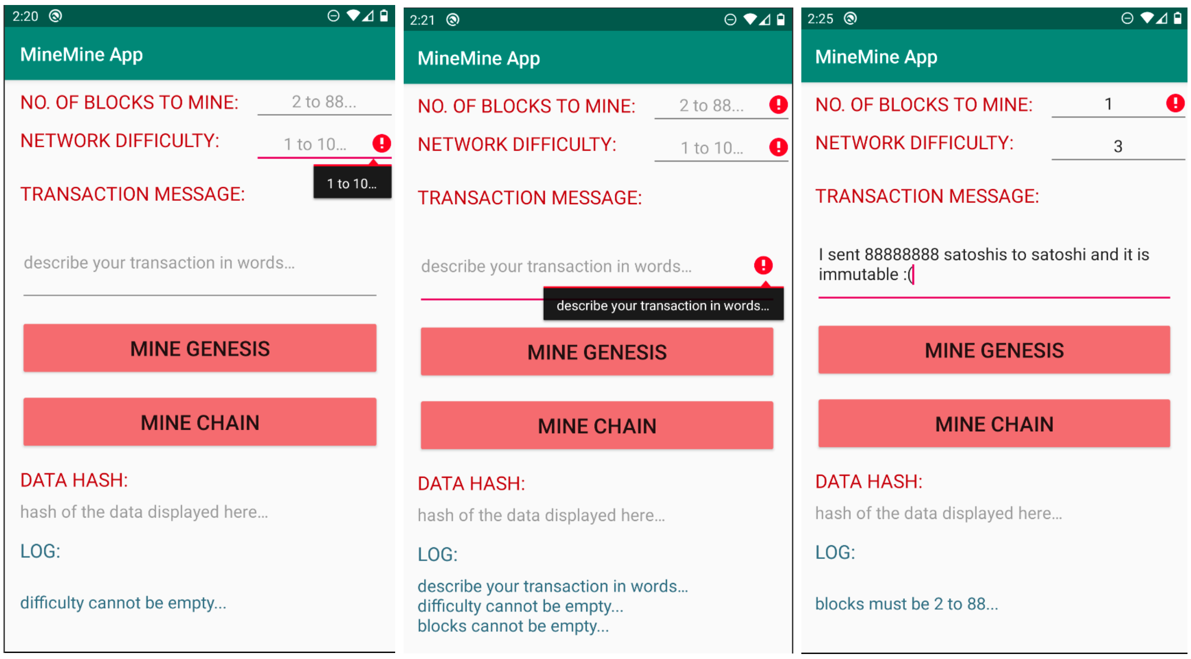
10. (1 mark) After **every block** mined after clicking **genesisButton** or **chainButton**, the nonce and timestamps should be printed in the Android console at the INFO level with the TAG set to "BTCONATIVE" (Figure 3). The logging should be performed in native code.
11. (1 mark) Well-commented code.
12. (1 mark) Good naming conventions for classes, methods and variables.

NEW Instruction: Please edit your README.md to describe which acceptance criteria you have completed. E.g., in your README.md:

```
## Fully completed
1,2,3,4,5,6
## Partially completed
7,8,9
## Not attempted
10,11,12
## Notes
I have added the following additional artifact in my app gradle to
aid dependency injection: `androidx.hilt:hilt-*:1.0.0`
```

(HINT: in case you are rusty with C syntax, you may want to refer to the `main(…)` *function in blockchain.c for ideas…)*

IMPORTANT:
Ensure that the ViewInstrumentedTest.kt test file compiles without errors (running and passing it will ensure even less errors). Do not edit any existing **package**, **class**, **variable**, **method** or **layout file** names. You may **add to, but do not edit existing strings in strings.xml**. Similarly, you may **add to, but do not edit existing dependencies** in, the gradle files.



2:20

MineMine App

NO. OF BLOCKS TO MINE: 2 to 88...

NETWORK DIFFICULTY: 1 to 10...

TRANSACTION MESSAGE: describe your transaction in words...

DATA HASH: hash of the data displayed here...

LOG:

difficulty cannot be empty...

2:21

MineMine App

NO. OF BLOCKS TO MINE: 2 to 88...

NETWORK DIFFICULTY: 1 to 10...

TRANSACTION MESSAGE: describe your transaction in words...

DATA HASH: hash of the data displayed here...

LOG:

describe your transaction in words...
difficulty cannot be empty...
blocks cannot be empty...

2:25

MineMine App

NO. OF BLOCKS TO MINE: 1

NETWORK DIFFICULTY: 3

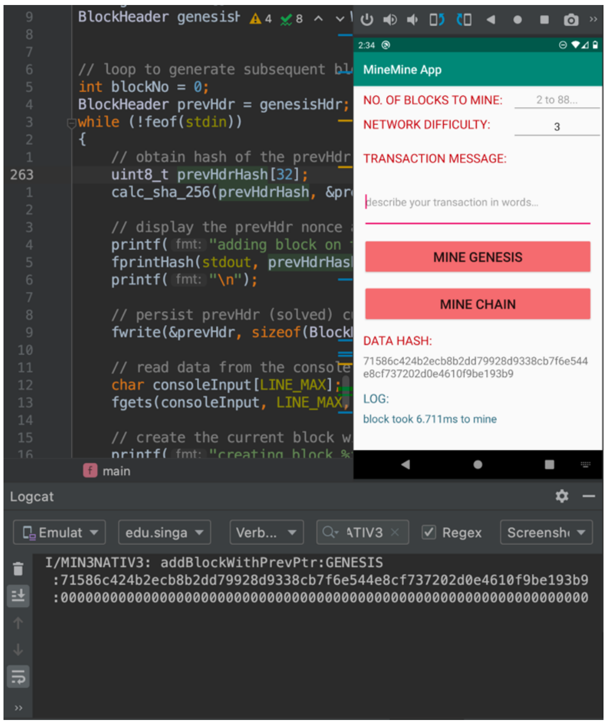
TRANSACTION MESSAGE: I sent 88888888 satoshis to satoshi and it is immutable :(

DATA HASH: hash of the data displayed here...

LOG:

blocks must be 2 to 88...

Figure 1: Various error messages in the LOG TextView – (left) clicking "MINE GENESIS" on empty fields, (middle) clicking "MINE CHAIN" on empty fields, and (right) clicking "MINE CHAIN" with invalid "NO. OF BLOCKS..." The red error prompts are optional.



```
BlockHeader genesisH

// loop to generate subsequent bl
int blockNo = 0;
BlockHeader prevHdr = genesisHdr;
while (!feof(stdin))
{
    // obtain hash of the prevHdr
    uint8_t prevHdrHash[32];
    calc_sha_256(prevHdrHash, &pr

    // display the prevHdr nonce
    printf( fmt: "adding block on
    fprintHash(stdout, prevHdrHas
    printf( fmt: "\n");

    // persist prevHdr (solved) c
    fwrite(&prevHdr, sizeof(Block

    // read data from the console
    char consoleInput[LINE_MAX];
    fgets(consoleInput, LINE_MAX,

    // create the current block w
    printf( fmt: "creating block %
```

2:34

MineMine App

NO. OF BLOCKS TO MINE: 2 to 88...

NETWORK DIFFICULTY: 3

TRANSACTION MESSAGE: describe your transaction in words...

DATA HASH: 71586c424b2ecb8b2dd79928d9338cb7f6e544e8cf737202d0e4610f9be193b9

LOG:

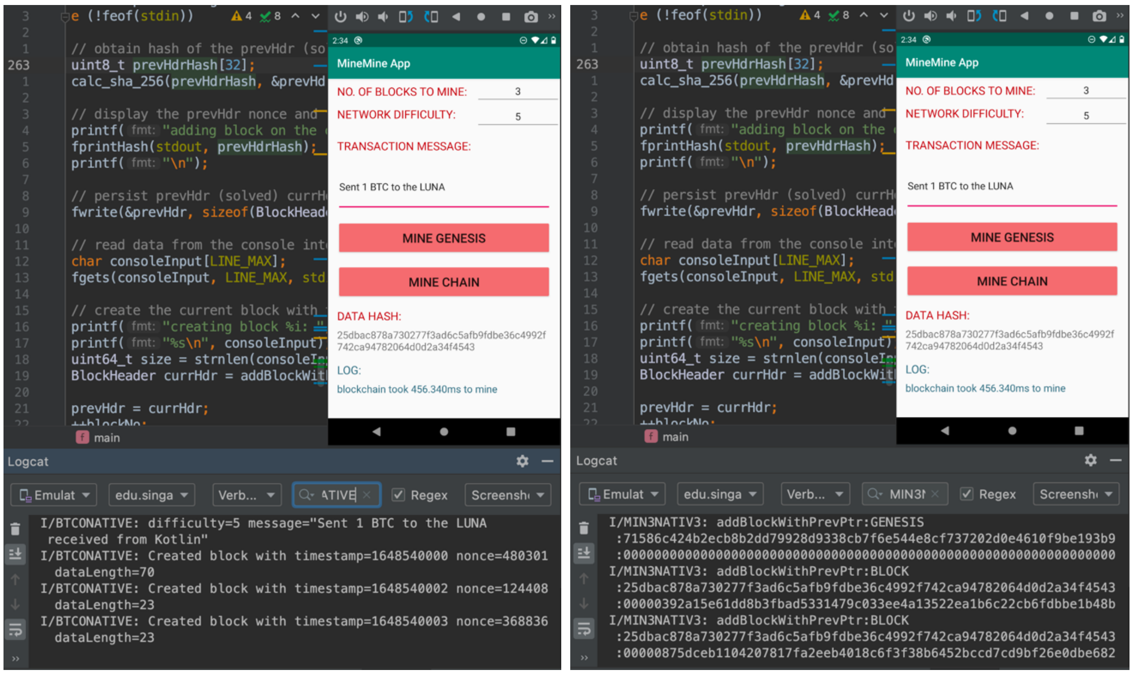
block took 6.711ms to mine

Logcat

Emulat | edu.singa | Verb... | ATIV3 | ✓ Regex | Screensh

```
I/MIN3NATIV3: addBlockWithPrevPtr:GENESIS
 :71586c424b2ecb8b2dd79928d9338cb7f6e544e8cf737202d0e4610f9be193b9
 :0000000000000000000000000000000000000000000000000000000000000000
```

Figure 2: After clicking on "MINE GENESIS" with network difficulty of "3". The logcat is filtered with "MIN3NATIV3".



```
e (!feof(stdin))

    // obtain hash of the prevHdr (so
    uint8_t prevHdrHash[32];
    calc_sha_256(prevHdrHash, &prevHd

    // display the prevHdr nonce and
    printf( fmt: "adding block on the
    fprintHash(stdout, prevHdrHash);
    printf( fmt: "\n");

    // persist prevHdr (solved) currH
    fwrite(&prevHdr, sizeof(BlockHead

    // read data from the console int
    char consoleInput[LINE_MAX];
    fgets(consoleInput, LINE_MAX, std

    // create the current block with
    printf( fmt: "creating block %i:
    printf( fmt: "%s\n", consoleInput)
    uint64_t size = strnlen(consoleIn
    BlockHeader currHdr = addBlockWit

    prevHdr = currHdr;
```

MineMine App

NO. OF BLOCKS TO MINE: 3

NETWORK DIFFICULTY: 5

TRANSACTION MESSAGE:

Sent 1 BTC to the LUNA

MINE GENESIS

MINE CHAIN

DATA HASH:

25dbac878a730277f3ad6c5afb9fdbe36c4992f742ca94782064d0d2a34f4543

LOG:

blockchain took 456.340ms to mine

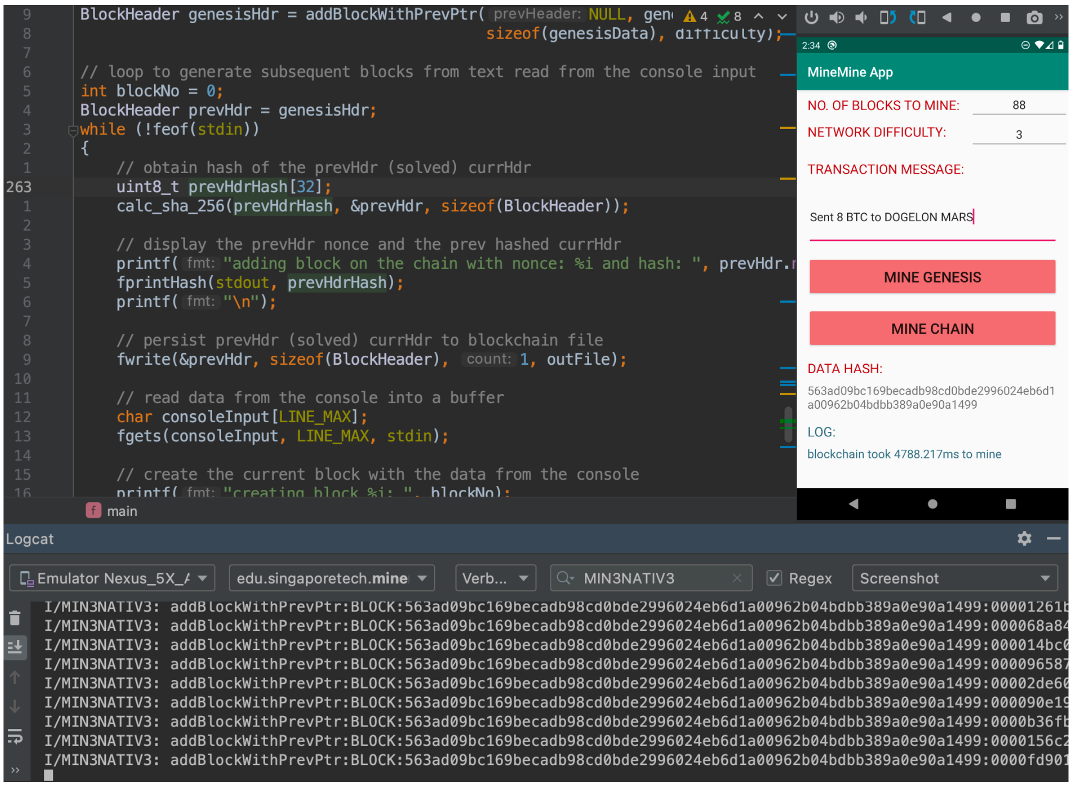
Logcat: Emulat · edu.singa · Verb... · ATIVE · ✓ Regex · Screenshot

```
I/BTCONATIVE: difficulty=5 message="Sent 1 BTC to the LUNA
 received from Kotlin"
I/BTCONATIVE: Created block with timestamp=1648540000 nonce=480301
  dataLength=70
I/BTCONATIVE: Created block with timestamp=1648540002 nonce=124408
  dataLength=23
I/BTCONATIVE: Created block with timestamp=1648540003 nonce=368836
  dataLength=23
```

Logcat: Emulat · edu.singa · Verb... · MIN3N · ✓ Regex · Screenshot

```
I/MIN3NATIV3: addBlockWithPrevPtr:GENESIS
 :71586c424b2ecb8b2dd79928d9338cb7f6e544e8cf737202d0e4610f9be193b9
 :0000000000000000000000000000000000000000000000000000000000000000
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK
 :25dbac878a730277f3ad6c5afb9fdbe36c4992f742ca94782064d0d2a34f4543
 :00000392a15e61dd8b3fbad5331479c033ee4a13522ea1b6c22cb6fdbbe1b48b
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK
 :25dbac878a730277f3ad6c5afb9fdbe36c4992f742ca94782064d0d2a34f4543
 :00000875dceb1104207817fa2eeb4018c6f3f38b6452bccd7cd9bf26e0dbe682
```

Figure 3: After clicking on “MINE CHAIN” with “3” blocks to mine and network difficulty of “5”. The logcat is filtered with “BTCONATIVE” (left) and “MIN3NATIV3” (right).



```
BlockHeader genesisHdr = addBlockWithPrevPtr( prevHeader: NULL, gen
                                              sizeof(genesisData), difficulty);

// loop to generate subsequent blocks from text read from the console input
int blockNo = 0;
BlockHeader prevHdr = genesisHdr;
while (!feof(stdin))
{
    // obtain hash of the prevHdr (solved) currHdr
    uint8_t prevHdrHash[32];
    calc_sha_256(prevHdrHash, &prevHdr, sizeof(BlockHeader));

    // display the prevHdr nonce and the prev hashed currHdr
    printf( fmt: "adding block on the chain with nonce: %i and hash: ", prevHdr.
    fprintHash(stdout, prevHdrHash);
    printf( fmt: "\n");

    // persist prevHdr (solved) currHdr to blockchain file
    fwrite(&prevHdr, sizeof(BlockHeader),  count: 1, outFile);

    // read data from the console into a buffer
    char consoleInput[LINE_MAX];
    fgets(consoleInput, LINE_MAX, stdin);

    // create the current block with the data from the console
    printf( fmt: "creating block %i: ", blockNo);
```

MineMine App

NO. OF BLOCKS TO MINE: 88

NETWORK DIFFICULTY: 3

TRANSACTION MESSAGE:

Sent 8 BTC to DOGELON MARS

MINE GENESIS

MINE CHAIN

DATA HASH:

563ad09bc169becadb98cd0bde2996024eb6d1a00962b04bdbb389a0e90a1499

LOG:

blockchain took 4788.217ms to mine

Logcat: Emulator Nexus_5X_A · edu.singaporetech.mine · Verb... · MIN3NATIV3 · ✓ Regex · Screenshot

```
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK:563ad09bc169becadb98cd0bde2996024eb6d1a00962b04bdbb389a0e90a1499:00001261b
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK:563ad09bc169becadb98cd0bde2996024eb6d1a00962b04bdbb389a0e90a1499:000068a84
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK:563ad09bc169becadb98cd0bde2996024eb6d1a00962b04bdbb389a0e90a1499:000014bc0
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK:563ad09bc169becadb98cd0bde2996024eb6d1a00962b04bdbb389a0e90a1499:00009658
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK:563ad09bc169becadb98cd0bde2996024eb6d1a00962b04bdbb389a0e90a1499:00002de6
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK:563ad09bc169becadb98cd0bde2996024eb6d1a00962b04bdbb389a0e90a1499:000090e1
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK:563ad09bc169becadb98cd0bde2996024eb6d1a00962b04bdbb389a0e90a1499:0000b36f
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK:563ad09bc169becadb98cd0bde2996024eb6d1a00962b04bdbb389a0e90a1499:0000156c
I/MIN3NATIV3: addBlockWithPrevPtr:BLOCK:563ad09bc169becadb98cd0bde2996024eb6d1a00962b04bdbb389a0e90a1499:0000fd90
```

Figure 4: After clicking on “MINE CHAIN” with “88” blocks to mine and network difficulty of “3”. The logcat is filtered with “MIN3NATIV3”.

Lab Quiz 3

1. Fork the repo **csc2007-quiz03-2022**, and then clone it. This code is incomplete.
2. Do everything you need to succeed in life, I mean here... Satisfy all the acceptance criteria if you can.
3. Commit and push all changes to your forked repository **csc2007-quiz03-2022.**

END OF DOCUMENT