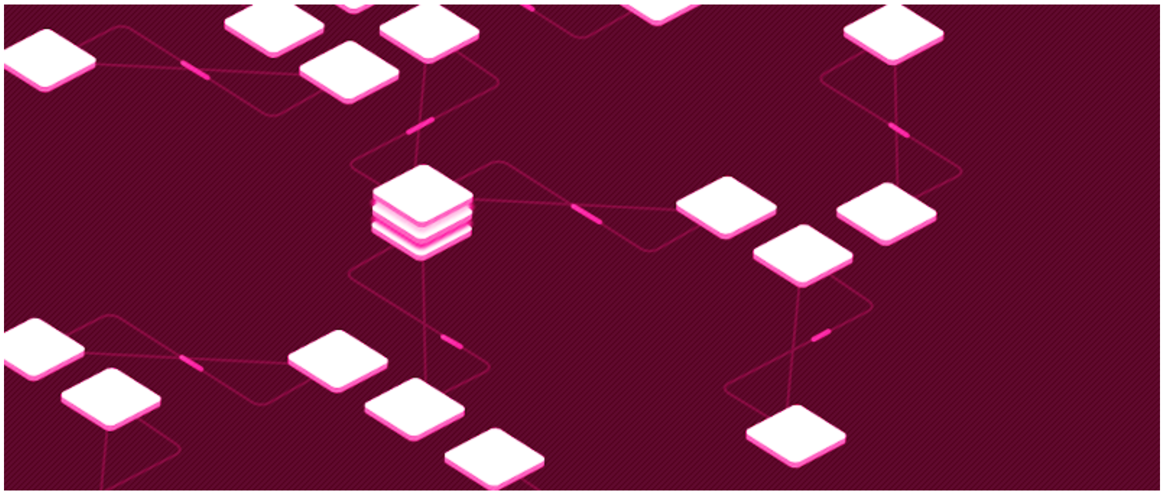
Dave Nash

DEVELOPER, ARCHITECT, IDEA FOUNDRY

BUILD A BLOCKCHAIN WITH C++

BLOCKCHAIN BLOG OCTOBER 20, 2017 63 COMMENTS



So, you might have heard a lot about something called a blockchain lately and wondered what all the fuss is about. A blockchain is a ledger which has been written in such a way that updating the data contained within it becomes very difficult, some say the blockchain is immutable and to all intents and purposes they're right but immutability suggests permanence and nothing on a hard drive could ever be considered permanent. Anyway, we'll leave the philosophical debate to the non-techies; you're looking for someone to show you how to write a blockchain in C++, so that's exactly what I'm going to do.

Before I go any further, I have a couple of disclaimers:

1. This tutorial is based on one written by Savjee using NodeJS; and
2. It's been a while since I wrote any C++ code, so I might be a little rusty.

The first thing you'll want to do is open a new C++ project, I'm using CLion from JetBrains but any C++ IDE or even a text editor will do. For the interests of this tutorial we'll call the project TestChain, so go ahead and create the project and we'll get started.

If you're using CLion you'll see that the main.cpp file will have already been created and opened for you; we'll leave this to one side for the time being.

Create a file called **Block.h** in the main project folder, you should be able to do this by right-clicking on the **TestChain** directory in the Project Tool Window and selecting: New > C/C++ Header File.

Inside the file, add the following code (if you're using CLion place all code between the lines that read `#define TESTCHAIN_BLOCK_H` and `#endif`):

```
#include <cstdint>
#include <iostream>
```

These lines above tell the compiler to include the `cstdint`, and `iostream` libraries.

Add this line below it:

```
using namespace std;
```

This essentially creates a shortcut to the `std` namespace, which means that we don't need to refer to declarations inside the `std` namespace by their full names e.g. `std::string`, but instead use their shortened names e.g. `string`.

So far, so good; let's start fleshing things out a little more.

A blockchain is made up of a series of blocks which contain data and each block contains a cryptographic representation of the previous block, which means that it becomes very hard to change the contents of any block without then needing to change every subsequent one; hence where the blockchain essentially gets its immutable properties.

So let’s create our block class, add the following lines to the **Block.h** header file:

```
class Block {
public:
    string sPrevHash;

    Block(uint32_t nIndexIn, const string &sDataIn);

    string GetHash();

    void MineBlock(uint32_t nDifficulty);

private:
    uint32_t _nIndex;
    int64_t _nNonce;
    string _sData;
    string _sHash;
    time_t _tTime;

    string _CalculateHash() const;
};
```

Unsurprisingly, we’re calling our class `Block` (line 1) followed by the **`public`** modifier (line 2) and public variable `sPrevHash` (remember each block is linked to the previous block) (line 3). The constructor signature (line 5) takes three parameters for `nIndexIn`, and `sDataIn`; note that the **`const`** keyword is used along with the reference modifier (**&**) so that the parameters are passed by reference but cannot be changed, this is done to improve efficiency and save memory. The GetHash method signature is specified next (line 7) followed by the `MineBlock` method signature (line 9), which takes a parameter `nDifficulty`. We specify the **`private`** modifier (line 11) followed by the private variables `_nIndex`, `_nNonce`, `_sData`, `_sHash`, and `_tTime` (lines 12–16). The signature for `_CalculateHash` (line 18) also has the **const** keyword, this is to ensure the method cannot change any of the variables in the block class which is very useful when dealing with a blockchain.

Now it’s time to create our **Blockchain.h** header file in the main project folder.

Let’s start by adding these lines (if you’re using CLion place all code between the lines that read `#define TESTCHAIN_BLOCKCHAIN_H` and `#endif`):

```
#include <cstdint>
#include <vector>
#include "Block.h"

using namespace std;
```

They tell the compiler to include the `cstdint`, and `vector` libraries, as well as the **Block.h** header file we have just created, and creates a shortcut to the `std` namespace.

Now let’s create our blockchain class, add the following lines to the **Blockchain.h** header file:

```
class Blockchain {
public:
    Blockchain();

    void AddBlock(Block bNew);

private:
    uint32_t _nDifficulty;
    vector<Block> _vChain;

    Block _GetLastBlock() const;
};
```

As with our block class, we’re keeping things simple and calling our blockchain class `Blockchain` (line 1) followed by the **`public`** modifier (line 2) and the constructor signature (line 3). The `AddBlock` signature (line 5) takes a parameter `bNew` which must be an object of the `Block` class we created earlier. We then specify the **`private`** modifier (line 7) followed by the private variables for `_nDifficulty`, and `_vChain` (lines 8–9) as well as the method signature for `_GetLastBlock` (line 11) which is also followed by the **`const`** keyword to denote that the output of the method cannot be changed.

Since blockchains use cryptography, now would be a good time to get some cryptographic functionality in our blockchain. We're going to be using the SHA256 hashing technique to create hashes of our blocks, we could write our own but really – in this day and age of open source software – why bother?

To make my life that much easier, I copied and pasted the text for the `sha256.h`, `sha256.cpp` and `LICENSE.txt` files shown on the C++ sha256 function from Zedwood and saved them in the project folder.

Right, let's keep going.

Create a source file for our block and save it as **Block.cpp** in the main project folder; you should be able to do this by right-clicking on the **TestChain** directory in the Project Tool Window and selecting: New > C/C++ Source File.

Start by adding these lines, which tell the compiler to include the **Block.h** and **sha256.h** files we added earlier.

```
#include "Block.h"
#include "sha256.h"
```

Follow these with the implementation of our block constructor:

```
Block::Block(uint32_t nIndexIn, const string &sDataIn) : _nIndex(nIndexIn), _
    _nNonce = -1;
    _tTime = time(nullptr);
}
```

The constructor starts off by repeating the signature we specified in the **Block.h** header file (line 1) but we also add code to copy the contents of the parameters into the the variables `_nIndex`, and `_sData`. The `_nNonce` variable is set to -1 (line 2) and the `_tTime` variable is set to the current time (line 3).

Let's add an accessor for the block's hash:

```
string Block::GetHash() {
    return _sHash;
}
```

We specify the signature for `GetHash` (line 1) and then add a return for the private variable `_sHash` (line 2).

As you might have read, blockchain technology was made popular when it was devised for the Bitcoin digital currency, as the ledger is both immutable and public; which means that, as one user transfers Bitcoin to another user, a transaction for the transfer is written into a block on the blockchain by nodes on the Bitcoin network. A node is another computer which is running the Bitcoin software and, since the network is peer-to-peer, it could be anyone around the world; this process is called 'mining' as the owner of the node is rewarded with Bitcoin each time they successfully create a valid block on the blockchain.

To successfully create a valid block, and therefore be rewarded, a miner must create a cryptographic hash of the block they want to add to the blockchain that matches the requirements for a valid hash at that time; this is achieved by counting the number of zeros at the beginning of the hash, if the number of zeros is equal to or greater than the difficulty level set by the network that block is valid. If the hash is not valid a variable called a nonce is incremented and the hash created again; this process, called Proof of Work (PoW), is repeated until a hash is produced that is valid.

So, with that being said, let's add the `MineBlock` method; here's where the magic happens!

```
void Block::MineBlock(uint32_t nDifficulty) {
    char cstr[nDifficulty + 1];
    for (uint32_t i = 0; i < nDifficulty; ++i) {
        cstr[i] = '0';
    }
    cstr[nDifficulty] = '\0';

    string str(cstr);

    do {
        _nNonce++;
        _sHash = _CalculateHash();
    } while (_sHash.substr(0, nDifficulty) != str);

    cout << "Block mined: " << _sHash << endl;
}
```

We start with the signature for the `MineBlock` method, which we specified in the **Block.h** header file (line 1), and create an array of characters with a length one greater that the value specified for `nDifficulty` (line 2). A **`for`** loop is used to fill the array with zeros, followed by the final array item being given the string terminator character (`\0`), (lines 3–6) then the character array or c-string is turned into a standard string (line 8). A **do…`while`** loop is then used (lines 10–13) to increment the `_nNonce` and `_sHash` is assigned with the output of `_CalculateHash`, the front portion of the hash is then compared the string of zeros we've just created; if no match is found the loop is repeated until a match is found. Once a match is found a message is sent to the output buffer to say that the block has been successfully mined (line 15).

We've seen it mentioned a few times before, so let's now add the `_CalculateHash` method:

```
inline string Block::_CalculateHash() const {
    stringstream ss;
    ss << _nIndex << _tTime << _sData << _nNonce << sPrevHash;

    return sha256(ss.str());
}
```

We kick off with the signature for the `_CalculateHash` method (line 1), which we specified in the **Block.h** header file, but we include the **`inline`** keyword which makes the code more efficient as the compiler places the method's instructions inline wherever the method is called; this cuts down on separate method calls. A string stream is then created (line 2), followed by appending the values for `_nIndex`, `_tTime`, `_sData`, `_nNonce`, and `sPrevHash` to the stream (line 3). We finish off by returning the output of the sha256 method (from the sha256 files we added earlier) using the string output from the string stream (line 5).

Right, let's finish off our blockchain implementation! Same as before, create a source file for our blockchain and save it as **Blockchain.cpp** in the main project folder.

Add these lines, which tell the compiler to include the **Blockchain.h** file we added earlier.

```
#include "Blockchain.h"
```

Follow these with the implementation of our blockchain constructor:

```
Blockchain::Blockchain() {
    _vChain.emplace_back(Block(0, "Genesis Block"));
    _nDifficulty = 6;
}
```

We start off with the signature for the blockchain constructor we specified in **Blockchain.h** (line 1). As a blocks are added to the blockchain they need to reference the previous block using its hash, but as the blockchain must start somewhere we have to create a block for the next block to reference, we call this a genesis block. A genesis block is created and placed onto the `_vChain` vector (line 2). We then set the `_nDifficulty` level (line 3) depending on how hard we want to make the PoW process.

Now it's time to add the code for adding a block to the blockchain, add the following lines:

```
void Blockchain::AddBlock(Block bNew) {
    bNew.sPrevHash = _GetLastBlock().GetHash();
    bNew.MineBlock(_nDifficulty);
    _vChain.push_back(bNew);
}
```

The signature we specified in **Blockchain.h** for `AddBlock` is added (line 1) followed by setting the `sPrevHash` variable for the new block from the hash of the last block on the blockchain which we get using `_GetLastBlock` and its `GetHash` method (line 2). The block is then mined using the `MineBlock` method (line 3) followed by the block being added to the `_vChain` vector (line 4), thus completing the process of adding a block to the blockchain.

Let's finish this file off by adding the last method:

```
Block Blockchain::_GetLastBlock() const {
    return _vChain.back();
}
```

We add the signature for `_GetLastBlock` from **Blockchain.h** (line 1) followed by returning the last block found in the `_vChain` vector using its **back** method (line 2).

Right, that's almost it, let's test it out!

Remember the **main.cpp** file? Now's the time to update it, open it up and replace the contents with the following lines:

```
#include "Blockchain.h"
```

This tells the compiler to include the Blockchain.h file we created earlier.

Then add the following lines:

```
int main() {
    Blockchain bChain = Blockchain();

    cout << "Mining block 1..." << endl;
    bChain.AddBlock(Block(1, "Block 1 Data"));

    cout << "Mining block 2..." << endl;
    bChain.AddBlock(Block(2, "Block 2 Data"));

    cout << "Mining block 3..." << endl;
    bChain.AddBlock(Block(3, "Block 3 Data"));

    return 0;
}
```

As with most C/C++ programs, everything is kicked off by calling the `main` method, this one creates a new blockchain (line 2) and informs the user that a block is being mined by printing to the output buffer (line 4) then creates a new block and adds it to the chain (line 5); the process for mining that block will then kick off until a valid hash is found. Once the block is mined the process is repeated for two more blocks.

Time to run it! If you are using CLion simply hit the 'Run TestChain' button in the top right hand corner of the window. If you're old skool, you can compile and run the program using the following commands from the command line:

```
gcc -lstdc++ \
    -o TestChain \
    -std=c++11 \
    -stdlib=libc++ \
    -x c++ \
    main.cpp Block.cpp Blockchain.cpp sha256.cpp
./TestChain
```

If all goes well you should see an output like this:

```
Mining block 1...
Block mined: 000000c1b50cb30fd8d9a0f2e16e38681cfcf9caead098cea726854925ab3772
Mining block 2...
```

```
Block mined: 0000005081063c8c854d11560cfea4fe734bde515a08565c26aa05448eea184e
Mining block 3...
Block mined: 000000ea61810fa85ff636440eb803263daf06b306c607aced9a1f996a421042
```

Congratulations, you have just written a blockchain from scratch in C++, in case you got lost I've put all the files into a Github repo. Since the original code for Bitcoin was also written in C++, why not take a look at its code and see what improvements you can make to what we've started off today?

If you have any questions or comments – as always – I'd love to hear them; please put them in the comments below.

(Visited 18,369 times, 1 visits today)

#Blockchain #Development

SHARE THIS POST

PREVIOUS POST

HOW TO USE JWT WITH SENECA, PASSPORT AND EXPRESS ON NODEJS

NEXT POST

HOW TO WRITE YOUR FIRST SMART PHONE APP IN XAMARIN FORMS

ABOUT THE AUTHOR

Dave Nash

YOU MAY ALSO LIKE

HOW TO WRITE YOUR FIRST SMART PHONE APP IN XAMARIN FORMS

HOW TO USE JWT WITH SENECA, PASSPORT AND EXPRESS ON NODEJS

63 COMMENTS

Magma13

JANUARY 23, 2018 AT 2:29 PM

My output stops at

Mining block 1…

What do you think can be the problem?

REPLY ↓

Dave Nash - SITE AUTHOR
APRIL 10, 2018 AT 9:24 AM

Blocks take time to be mined, if you want the output to be shown faster decrease the value for nDifficulty.

REPLY ↓

Gabi
FEBRUARY 18, 2018 AT 3:48 PM

Hello. How i can view all mined blocks ?

REPLY ↓

Dave Nash - SITE AUTHOR
APRIL 10, 2018 AT 9:24 AM

Hi Gabi, that's a little outside the scope of this tutorial; feel free to fork the repo and add the necessary code!

REPLY ↓

thomas
FEBRUARY 21, 2018 AT 9:50 AM

how can i store the blocks in my hard drive and also share the ledger to all the nodes in the network

REPLY ↓

Dave Nash - SITE AUTHOR
APRIL 10, 2018 AT 9:26 AM

Hi Thomas, over to you; feel free to fork the repo and add the code you need!

REPLY ↓

Sam
APRIL 26, 2018 AT 12:47 PM

Hi, Dave! I didn't understand your answer to Thomas, so I repeat the question again.
How are the blocks stored on the hard drive and then retrieved?
What kind of file should the blockchain be? Does every blockchain system use its own type which might be decoded by its own applications?
Should only newly created block or the whole ledger be shared on the network?
Thanks in advance.

REPLY ↓

Dave Nash - SITE AUTHOR
MAY 19, 2018 AT 1:21 PM

Hi Sam, this example keeps everything in memory; nothing is written to the file system. You might want to check the source code for Bitcoin to see how it handles file system and network access.

REPLY ↓

Stephanie
FEBRUARY 23, 2018 AT 12:30 AM

I really like what you have going on here, but I find it really difficult to follow because you don't explain what your variables mean or why you need them. Some are easy to guess at, but not all.

REPLY ↓

Dave Nash - SITE AUTHOR
APRIL 10, 2018 AT 9:28 AM

Hi Stephanie, which variables are you having trouble with?

REPLY ↓

Pedram
MARCH 4, 2018 AT 6:05 PM

where emplace_back is defined?

REPLY ↓

Dave Nash - SITE AUTHOR
APRIL 10, 2018 AT 9:29 AM

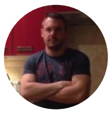
Hi Pedram, I hope this helps: http://en.cppreference.com/w/cpp/container/vector/emplace_back

REPLY ↓

sakshi
MARCH 21, 2018 AT 10:30 AM

header file such as ctime and sstream should be included for time(nullptr) and string stream respectively

REPLY ↓



Dave Nash - SITE AUTHOR
APRIL 10, 2018 AT 9:27 AM

Feel free to fork the repo and make the changes you think are missing!

REPLY ↓

□□□
APRIL 3, 2018 AT 4:31 PM

Thank you, I never find a Chinese example in C++, now I find the first one in English though, but really easy to understand

REPLY ↓

□□□
APRIL 4, 2018 AT 10:57 AM

should “char cstr[nDifficulty + 1];” be changed to “char * cstr= new char[nDifficulty + 1];”? I thought nDifficulty is not a constant

REPLY ↓

d00dz
MAY 4, 2018 AT 9:35 AM

I think is shorter use …. string str(nDifficulty, ‘0’);

REPLY ↓

Dave Nash - SITE AUTHOR
MAY 19, 2018 AT 1:23 PM

Feel free to fork the code on Github and add your own changes 🙂 https://github.com/teaandcode/TestChain

REPLY ↓

Yaduvendra Singh
APRIL 16, 2018 AT 9:27 PM

Thanks Dave ! Great example of using C++ for block chain.

REPLY ↓

alexis
APRIL 27, 2018 AT 4:49 AM

!Great it is working on linux debian 9.0¡

REPLY ↓

Roman
MARCH 16, 2020 AT 6:28 PM

Yea. Really good example of chainblock and c++. Tanks for your effort!

REPLY ↓

KS
MAY 3, 2018 AT 6:51 AM

Thx Dave! Easy to understand and expandable!

REPLY ↓

LordOfDarkess
MAY 16, 2018 AT 6:49 PM

I really like the idea that block mines itself 😉

REPLY ↓

nFactorial
MAY 19, 2018 AT 12:48 PM

Hi! You can to help me to replace vector with Simple linked list?

REPLY ↓

Dave Nash - SITE AUTHOR
MAY 19, 2018 AT 1:29 PM

I'm not an expert in C++, but generally lists aren't recommended (http://blog.davidecoppola.com/2014/05/cpp-benchmarks-vector-vs-list-vs-deque/); why do you want to change the vector to a list?

REPLY ↓

nFactorial
MAY 19, 2018 AT 2:15 PM

for a project at school, i need list

REPLY ↓

Stephen
AUGUST 2, 2018 AT 8:46 AM

Awesome article and code Dave! Although vector has great advantage over list with erase and insert, with blockchain's special property: the blocks cannot be deleted and always add new blocks to the end of the chain. Which means only push_back method is used in blockchain, so I think it's should be fine to use list versus vector right?
Anyways, great work I learned a lot!

REPLY ↓

nFactorial
MAY 21, 2018 AT 4:35 PM

_vChain.emplace_back(Block(0, "Genesis Block")); Here adding element 0 to next block?

REPLY ↓

Guillermo
MAY 31, 2018 AT 2:51 AM

I am getting an error at:
char cstr[nDifficulty + 1];

Also:
stringstream ss;
ss <<

I have them in the Block.cpp file.

REPLY ↓

Dave Nash - SITE AUTHOR
JUNE 2, 2018 AT 11:06 AM

What's the error? Also, what command are you running to compile the code?

REPLY ↓

Rajendra Maharjan
JUNE 8, 2018 AT 5:45 PM

I got the same error as mentioned by Guillermo above. Here is the error. I am running this project in CLion in windows. After using #include , the 2nd error mentioned above got resolved.
block.cpp
error C2131: expression did not evaluate to a constant
note: failure was caused by non-constant arguments or reference to a non-constant symbol
note: see usage of 'nDifficulty'
error C3863: array type 'char [nDifficulty+]' is not assignable
error C3863: array type 'char [nDifficulty+]' is not assignable
NMAKE : fatal error U1077: 'C:\PROGRA~2\MICROS~1.0\VC\bin\cl.exe' : return code '0x2'
Stop.
NMAKE : fatal error U1077: '"C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC
\BIN\nmake.exe"' : return code '0x2'
Stop.
NMAKE : fatal error U1077: '"C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC
\BIN\nmake.exe"' : return code '0x2'
Stop.
NMAKE : fatal error U1077: '"C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC
\BIN\nmake.exe"' : return code '0x2'
Stop.

REPLY ↓

Dave Nash - SITE AUTHOR
JULY 26, 2018 AT 6:07 PM

I haven't tried compiling the code on Windows, I recommend using a unix machine

REPLY ↓

Vel
DECEMBER 19, 2019 AT 6:45 PM

if you are using in windows replace
char cstr[nDifficulty + 1];
to
char *cstr = new char[nDifficulty + 1];

REPLY ↓

Joao
JUNE 13, 2018 AT 2:05 PM

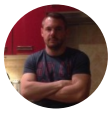
Great tutorial, Sr!

REPLY ↓

Ender0224
JUNE 18, 2018 AT 9:52 AM

hello Dave
thanks so much for your kind share.
will you go on update the rep for more details and function? or you just give a good example and stay here?

REPLY ↓



Dave Nash - SITE AUTHOR
JULY 26, 2018 AT 6:08 PM

I have no plans to update the code

REPLY ↓

shawn
JULY 9, 2018 AT 6:14 PM

Hello Dave,

I have the same issues as Guillermo.
I am relatively new to coding so this might seem relatively simple to you:
– I am using Microsoft Visual Studio.
– ndifficulty is underligned and it says that the expression must have a constant value.
– ss is underligned and it says “incomplete type not allowed”

Thank you in advance!

REPLY ↓

Dave Nash - SITE AUTHOR

JULY 26, 2018 AT 6:09 PM

I have not tried compiling the code in Visual Studio, I recommend using a unix machine

REPLY ↓

Leo
SEPTEMBER 7, 2018 AT 9:19 PM

"– ndifficulty is underligned and it says that the expression must have a constant value."
you should use dynamic allocation of memory. (new[], delete[] will help you)

"– ss is underligned and it says "incomplete type not allowed""
#include

REPLY ↓

javier
APRIL 12, 2019 AT 8:17 AM

I add `#include ` in Block.cpp, and then i can compile

REPLY ↓

Bob
SEPTEMBER 5, 2018 AT 9:58 AM

Hi, this explanation is awesome, and I can't thank you enough for posting it. Not only is your code clear, it makes the concept of block chains themselves so simple to understand. Thanks much for this code and for taking the time to make this entry!

REPLY ↓

StewPead
SEPTEMBER 17, 2018 AT 9:16 AM

I got errors:

C:\Users\RJay\AppData\Local\Temp\ccWgRhMX.o:main.cpp:(.text+0x1da): undefined reference to `Blockchain::AddBlock(Block)'
collect2.exe: error: ld returned 1 exit status

and
"_nIndex" is not a nonstatic data member or base class of class "Block"

Im using Visual Studio Code.

REPLY ↓

Dave Nash - SITE AUTHOR
NOVEMBER 18, 2018 AT 3:55 PM

I've not tried compiling the code on a Windows machine, did you try running the code from the Github repo too?

REPLY ↓

Unnati
SEPTEMBER 18, 2018 AT 9:21 AM

Hi
could u please tell me from here how to design a bitcoin, since its implemented using blockchain.
thanks.

REPLY ↓

Dave Nash - SITE AUTHOR
NOVEMBER 18, 2018 AT 3:41 PM

Thanks for your suggestion, but at present I have no plans to write a tutorial on how to design a coin. You might want to take what you've learned here in this tutorial and examine the Bitcoin code over on Github, that might help give you an idea on how to proceed.

REPLY ↓

udit
SEPTEMBER 30, 2018 AT 2:10 PM

once we get a block mined, at owners end how can we get our string back from, the blockchain ? owner must know what he is writing in ledger !

REPLY ↓

Dave Nash - SITE AUTHOR
NOVEMBER 18, 2018 AT 3:50 PM

If you take a look at the code in the tutorial, you will notice that I place the blocks inside a vector; so you should be able to get the string back to show to the user.

REPLY ↓

darludi58
OCTOBER 22, 2018 AT 6:59 PM

Result using codelite on ubuntu, what's wrong?

Command line is empty. Build aborted.Command line is empty. Build aborted.MESSAGE: Entering

```
directory `/home/map/Dev/BlockChain’
/bin/sh -c ‘/usr/bin/make -f/home/map/Dev/BlockChain/Makefile -j 32’
———-Building project:[ BlockChain – ]———-
make[2]: *** No rule to make target ‘src/Main.cpp’, needed by ‘CMakeFiles/BlockChain.dir
/src/Main.cpp.o’. Stop.
make[1]: *** [CMakeFiles/Makefile2:73: CMakeFiles/BlockChain.dir/all] Error 2
make: *** [/home/map/Dev/BlockChain/Makefile:84: all] Error 2
====0 errors, 0 warnings====
```

REPLY ↓

Dave Nash - SITE AUTHOR
NOVEMBER 18, 2018 AT 3:52 PM

Hey, I haven’t tried compiling the code on Ubuntu; are you using the compile command in the tutorial? Also, have you tried pulling down the code from Github and compiling that?

REPLY ↓

Jackson
OCTOBER 24, 2018 AT 12:30 PM

Can we convert completely or partially the above block chain code into parallel code using OpenMP or Multithreading? Any places you can suggest which can be parallelised? Thanks.

REPLY ↓

Dave Nash - SITE AUTHOR
NOVEMBER 18, 2018 AT 3:52 PM

That’s a little outside the scope of this tutorial, I’d be interested to hear how you get on though!

REPLY ↓

junior
OCTOBER 26, 2018 AT 5:45 PM

i have an error message when a compile the code:

unrecognized command line option ‘-stdlib=libc++’

REPLY ↓

Dave Nash - SITE AUTHOR
NOVEMBER 18, 2018 AT 3:54 PM

What are you entering on the command line to get that error?

REPLY ↓

KyawSwarLin
SEPTEMBER 21, 2019 AT 2:21 PM

Hi sir .how can code with C++ for POS mining algorithm.POW will consume power .So POS may be better.Please show some example of POS in above codes .Thanks you sir.
And also how about public and private key for this cryptocurrency.

REPLY ↓

Dave Nash - SITE AUTHOR
SEPTEMBER 28, 2019 AT 11:17 AM

Thanks for your question but I haven't done any work on Blockchain for well over a year so I cannot comment on PoW vs. PoS, I'm sure there are some examples of using PoS around the internet. This tutorial is just as an introduction of Blockchain written in C++, it's not written with any cryptocurrency in mind.

REPLY ↓

Anton Anderson
JANUARY 2, 2019 AT 3:32 PM

I have actually typed out this code as I was reading this fantastic blog article.
I was able to make it to run on a Windows 10 machine using Visual Studio 2010.

It works perfectly!

I recommend to change this line to Block::MineBlock():

```
char cstr[nDifficulty + 1];
```

To this:

```
char* cstr = new char[nDifficulty + 1];
```

And if one wants to see Proof of Work (the number of times before a valid hash is accepted then add this line of code:

```
cout << "Proof of Work (PoW): " << _nNonce << endl << endl;
```

And do not forget to add :

```
delete cstr;
```

at the end to avoid memory leaks!

Thank you Dave. This was EXACTLY what I was looking for to learn the basics behind blockchains.

Now I am ready to dig deeper into learning more about this. I have a posted a link to this blog post at my profile page at Codementor. And I plan to link it on my LinkedIn page as well.

Again you have my thanks!

REPLY ↓

Dave Nash - SITE AUTHOR
JANUARY 12, 2019 AT 3:58 PM

Hey, I’m glad you found the tutorial useful; I’m not a C++ developer so happy to hear feedback with ways to improve it.

REPLY ↓

Nikki
JANUARY 14, 2019 AT 7:48 AM

Will this work on macbook?

REPLY ↓

Dave Nash - SITE AUTHOR
JANUARY 21, 2019 AT 11:23 PM

It was written on an iMac, so I don’t see why a MacBook wouldn’t work

REPLY ↓

Chuck
JANUARY 31, 2019 AT 2:22 PM

Hi
Could you recommend me any method to calculate hash faster

REPLY ↓

Dave Nash - SITE AUTHOR
SEPTEMBER 9, 2019 AT 10:34 PM

You can try reducing the difficulty

REPLY ↓

Ayan Roy
FEBRUARY 21, 2019 AT 7:09 PM

Hi. I have a small question. I am mining block of the form “V001,1”. Now I would like to query the blockchain with input such as “V001” and want it to return the value 1. Do you know of anything how I can modify this code to do that ? Thanks in advance

REPLY ↓

Dave Nash - SITE AUTHOR
SEPTEMBER 9, 2019 AT 10:37 PM

You’d probably have to keep some kind of index to locate the block you’re looking for, I recommend forking the code on Github and giving it a try

REPLY ↓

LEAVE A COMMENT

COMMENT

NAME *

EMAIL *

WEBSITE

☐ NOTIFY ME OF FOLLOW-UP COMMENTS BY EMAIL.

☐ NOTIFY ME OF NEW POSTS BY EMAIL.

POST COMMENT

This site uses Akismet to reduce spam. Learn how your comment data is processed.

CATEGORIES

Select Category

TOP POSTS & PAGES

- Build a blockchain with C++
- How to write your first smart phone app in Xamarin Forms

KO-FI

Buy Me a Coffee

OTHER PLACES TO FIND ME

Dave Nash

DEVELOPER, ARCHITECT, IDEA FOUNDRY