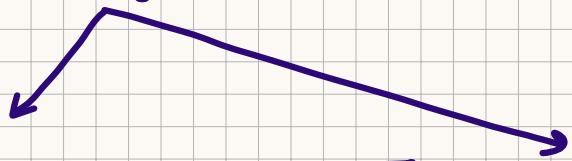
Project (1)

Simulating Crypto environment

Crypto



The Market

The basic idea of creating the market is to actual simulation of bitcoin/ any other crypto over an fixed horizon of time, it's price fluctuations done by various Agents Acting independently

The indivisual

Create an individual account
Assign him an wallet adress on which the transaction will takeplace
give him an PIN / generate him an Private key

Note before starting off this is an extremely rudimentary & doesn't really do all the complicated stuff all it actually does is to simulate the train & the daily bots's Price fluctuation

Some of the unknown areas have been generated using AI

The Actual Code breakdown

```
#include <bits/stdc++.h>
#include <random>
#include <thread>
#include <chrono>
using namespace std;

// Generate random hex string
string randomHex(int length) {
    static const char hexChars[] = "0123456789abcdef";
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dist(0, 15);

    string s;
    for (int i = 0; i < length; i++)
        s += hexChars[dist(gen)];
    return s;
}

int main() {
    random_device rd;
    mt19937 gen(rd());

    uniform_real_distribution<double> priceMove(-1000, 1000);
    uniform_int_distribution<int> buySell(0, 1);
    uniform_int_distribution<int> walletPick(0, 9);
    uniform_real_distribution<double> amountGen(0.01, 0.5);

    // User wallet
    string userAddress = "0x" + randomHex(40);
    string userPrivate = randomHex(64);
    double userBTC = 1000.0;

    // Demo contacts
    map<string, string> contacts = {
        {"1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa", "Alice"},
        {"3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy", "Dad"},
        {"1BoatSLRHtKNngkdXEeobR76b53LETtpyT", "Mom"},
        {"1Ez69SnzzmePmZX3WpEzMKTrcBF2gpNQ55", "Coffee Shop"},
        {"3QJmV3qfvL9SuYo34YihAf3sRCW3qSinyC", "General Store"}
    };

    // Market wallets
    vector<string> wallets(10);
```

This part explained nxt Page

Provides secure random number generation

for potential multithreading

for time related ops

initialize global Rand seed

Random price changes ±1000

0 = Buy, 1 = Sell; randomly

// Pick from any of 0-9 wallets created

Random trade Amts (0.01 - 0.5 BTC)

// Fake eth style Public adresses (80 hex chars total)

Starting BTC balance

There are hard coded demo wallets. These are actual legacy P2PKH adresses from BTC's early days

1

2

3

That Part explained here :-

Key Components Explained

`static const char hexChars[] = "0123456789abcdef";`

- Defines array of **16 valid hexadecimal characters** (0-9, a-f)
- `static` means created once, reused across calls (performance)

`random_device rd;`

- **Hardware RNG**: Gets true randomness from OS/hardware (not predictable)

`mt19937 gen(rd());`

- **Mersenne Twister engine**: High-quality pseudo-random generator
- Seeded by `rd()` for unpredictability

`uniform_int_distribution<> dist(0, 15);` ← **THIS IS THE "DISTRIBUTION"**

- Creates **uniform distribution** over integers 0-15 (equal probability 1/16 each)
- `<>` uses `int` type deduction
- Each call `dist(gen)` returns random int 0-15

`string s; for (int i = 0; i < length; i++) s += hexChars[dist(gen)];`

- Builds string by repeatedly:
  a. `dist(gen)` → random index 0-15
  b. `hexChars[index]` → random hex char ('a', '3', 'f', etc.)
  c. `s += char` → append to result

Example Execution (`randomHex(5)`):

```text
dist(gen) → 10 → hexChars[10] = 'a'
dist(gen) → 3  → hexChars[3]  = '3'
dist(gen) → 15 → hexChars[15] = 'f'
dist(gen) → 7  → hexChars[7]  = '7'
dist(gen) → 12 → hexChars[12] = 'c'
Result: "a3f7c"
```

Usage in simulator:

- `userAddress = "0x" + randomHex(40)` → "0x1a2b3c..."
- `Tx Hash: 0x` + randomHex(64) → Fake transaction IDs [1] [2] [3]

String randomHex (int length) {. } → Generates an random string of unspecified length (eg → for wallet adresses & private keys uses random_device (hardware entropy) to seed mt19937 (mersenne twister RNG), picks random hex digits (0-9, a-f) & builds the string called later to create fake "0x..." ethereum style adresses & wallet keys

① → sets up distribution for simulating market volatality. buy/sell desecisions & trade sizes

② user private = randomHex(64) => fake 256 bit Private key

③ vector < string> . . . . } => This code is used for creating randomHex(40), an vector of 10 fake eth style wallet adressess (each 40 hex chars prefixed with "0x") These simulate the "market makers" & the other traders, & is intended for automated market trades later

```
for (int i = 0; i < 10; i++)
    wallets[i] = "0x" + randomHex(40);
```
contd from Last Page of ③

```
// Initial BTC price
double price = 50000.0;
```
⇒ This is the opening Price

```
cout << "\n=== ARTIFICIAL BTC SIMULATOR ===\n";
cout << "Your Wallet Address: " << userAddress << "\n";
cout << "Your Private Key:    " << userPrivate << "\n";
cout << "Starting Balance:    " << userBTC << " BTC\n";
```
welcome banner

```
while (true) {
    cout << "\nChoose action (T = transaction, M = market, Q = quit): ";
    char choice;
    cin >> choice;
    cin.ignore(); // flush newline
```
∞ Interactive loop

Clears input buffer after reading char, discards leftover newline char to prevent I/P issue

```
    if (choice == 'Q' || choice == 'q') break;

    // ============================
    //       USER TRANSACTIONS
    // ============================
    if (choice == 'T' || choice == 't') {
        cout << "\nYour balance: " << userBTC << " BTC\n";
        cout << "Send to one of the following addresses:\n\n";
        for (auto &p : contacts)
            cout << p.second << ": " << p.first << "\n";
```
Prints "Alice": her BTC address"

```
        cout << "\nEnter address: ";
        string addr;
        cin >> addr;
        cin.ignore();

        if (!contacts.count(addr)) {
            cout << "❌ Unknown address!\n";
            continue;
        }
```
// map.count() checks if key exists

Skips to next loop iteration

```
        cout << "Enter amount: ";
        double amt;
        cin >> amt;
        cin.ignore();
```
Prompts user for the BTC amount, reads double & clears buffer input

```
        if (amt > userBTC) {
```

```
cout << "❌ Insufficient balance!\n";
        continue;
    }

    cout << "\nSending " << amt << " BTC to " << contacts[addr] <<
"...\n";

    for (int i = 1; i <= 3; i++) {
        cout << "Confirmations: " << i << "/3\n";
        this_thread::sleep_for(chrono::milliseconds(600));
    }

    userBTC -= amt;

    cout << "\n✅ Transaction Successful!\n";
    cout << "New Balance: " << userBTC << " BTC\n";
    cout << "Tx Hash: 0x" << randomHex(64) << "\n";
}

// ============================
//        MARKET MODE
// ============================
else if (choice == 'M' || choice == 'm') {
    cout << "\n=== MARKET LIVE MODE ===\n";
    cout << "Press ENTER to stop the market...\n\n";

    bool stopMarket = false;

    thread marketThread([&]() {
        while (!stopMarket) {
```

=> This prevents overdraft (continue skips back to the main menu if insufficient funds

=> This is fake blockchain confirmation loops 3x, Printing "confirmations: 1/3 : Pauses 600ms each x1 BS, in total

Mimics real BTC mining time

Deducts the amt shows Success & fake 64 hex transaction ID

false keeps the market running, true stops it

creates threads & captures all variable by reference (&) Price, gen, wallets, stopmarket etc Thread runs independently

```
int w = walletPick(gen);
        bool isBuy = buySell(gen);
        double amount = amountGen(gen);

        price += priceMove(gen);
        if (price < 1000) price = 1000;

        cout << "Price: $" << fixed << setprecision(2) << price << " | "
             << (isBuy ? "BUY " : "SELL ") << amount
             << " BTC by wallet " << (w + 1) << "\n";

        this_thread::sleep_for(chrono::milliseconds(700));
    }
});
```

→ generates BTC Amt but 0 or 1 to 0.5

→ Price floor. Price cannot drop below $1000 in this sim

→ Random wallet Selection → walletpick (Pre defined uniform_int_distribution<int>(0,9)) generates random integer 0-9 using gen (mt19937 engine) w picks which of 10 markets wallet is trading

→ Random trade direction → buySell (uniform_int_distribution<int>(0,1)) gives 0 or 1 Assigned to bool (false = 0 = Sell, true = 1 = buy) 50/50, buy/Sell odds

→ Price volatality Price move adds random change ±$1000 to current price simulate market swings

→ Console o/p formatting -

- fixed << setprecision(2) $50234.26 (2 decimal places)
- Ternary is buy? "BUY" "SELL". BUY if true & SELL if false
- w+1 ⇒ "wallet 7" (not "wallet 0")

this_thread::sleep_for(chrono::milliseconds(700))

Pacing delay pauses the thread 700ms before the next trade creates ~ 85 trades/min realistic pace

```
// Wait for ENTER to stop
        cin.get();
        stopMarket = true;
        marketThread.join();

        cout << "\nExited market mode.\n";
    }
  }

  cout << "Exiting simulator...\n";
  return 0;
}
```

⇒ Signing off.