Experiment No.6
To design a smart contract implementing the concepts of wallet and transaction using solidity
Date of Performance:
Date of Submission:

AIM: To design a smart contract implementing the concept of wallet and transaction using solidity

Objective: To develop a program using solidity to create a demonstration of wallet and perform transaction on the wallet

Theory: A blockchain wallet is a cryptocurrency wallet that allows users to manage different kinds of cryptocurrencies—for example, Bitcoin or Ethereum. A blockchain wallet helps someone exchange funds easily. Transactions are secure, as they are cryptographically signed. The wallet is accessible from web devices, including mobile ones, and the privacy and identity of the user are maintained. So a blockchain wallet provides all the features that are necessary for safe and secure transfers and exchanges of funds between different parties.

Helps in Exchange of funds
Privacy is maintained
Blockchain wallet
Transactions are secure
Accessible from web or mobile devices

Fig, 6.1. Illustrative view of Wallet

Blockchain Wallet Types

There are two types of blockchain wallets based on private keys: hot wallets and cold wallets. Hot wallets are like normal wallets that we carry for day-to-day transactions, and these wallets are user-friendly. Cold wallets are similar to a vault; they store cryptocurrencies with a high level of security.

Hot Wallets and Cold Wallets

Hot wallets are online wallets through which cryptocurrencies can be transferred quickly. They are available online. Examples are Coinbase and Blockchain.info. Cold wallets are digital offline wallets where the transactions are signed offline and then disclosed online. They are not maintained in the cloud on the internet; they are maintained offline to have high security.

Examples of cold wallets are Trezor and Ledger.With hot wallets, private keys are stored in the cloud for faster transfer. With cold wallets, private keys are stored in separate hardware that is not connected to the internet or the cloud, or they are stored on a paper document. Hot wallets are easy to access online 24/7 and can be accessed via a desktop or mobile device, but there is the risk of unrecoverable theft if hacked. With cold wallets, the method of the transaction helps in protecting the wallet from unauthorized access (hacking and other online vulnerabilities).

Process:

Step 1. Open Remix **IDE** by typing URL https://remix.ethereum.org/.

Step 2. In Remix IDE select 'Solidity' plugins

Step 3. Click on File Explorer

Step 4. In the default workspace click on 'create new file'

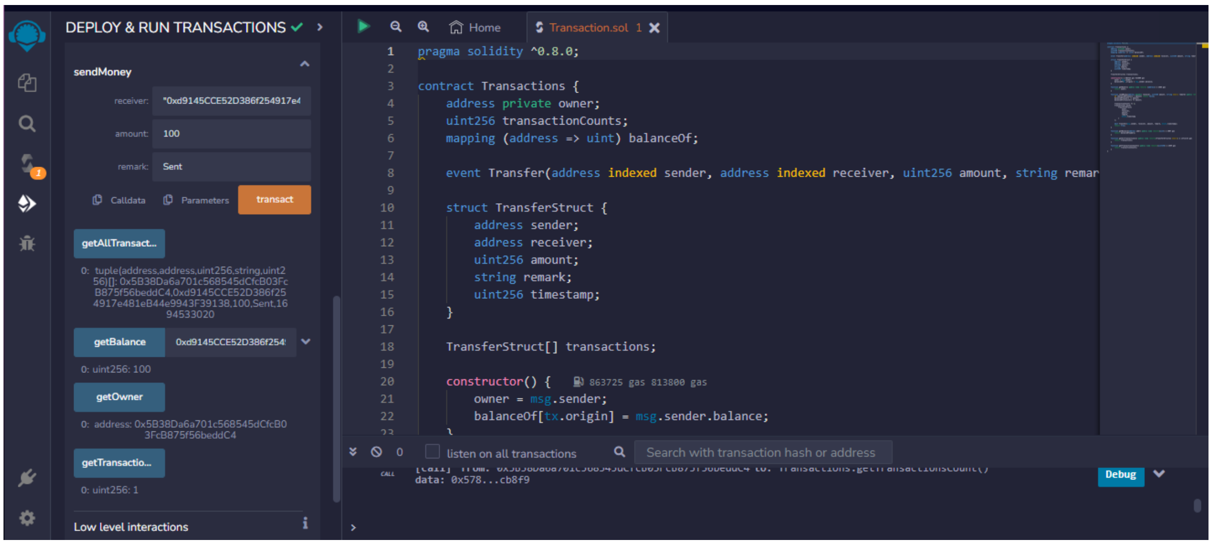
Step 5. Give suitable name to the *file* with extension .sol

Step 6. Type the (Wallet and transaction) code in the editor section of the Remix IDE for newly created file (.sol)

Step 7. After typing the code for the smart contract in the newly creates .sol file,click on the compiler option available and then compile the file

Step 8. If no error, then click on the 'Deploy and Run' to execute the smart contract.

Output:



```
sendMoney
receiver: "0xd9145CCE52D386f254917e4
amount: 100
remark: Sent

getAllTransact...
0: tuple(address,address,uint256,string,uint256)[]: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4,0xd9145CCE52D386f254917e481eB44e9943F39138,100,Sent,1694533020

getBalance 0xd9145CCE52D386f254
0: uint256: 100

getOwner
0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

getTransactio...
0: uint256: 1
```

```
pragma solidity ^0.8.0;

contract Transactions {
    address private owner;
    uint256 transactionCounts;
    mapping (address => uint) balanceOf;

    event Transfer(address indexed sender, address indexed receiver, uint256 amount, string remar

    struct TransferStruct {
        address sender;
        address receiver;
        uint256 amount;
        string remark;
        uint256 timestamp;
    }

    TransferStruct[] transactions;

    constructor() {    863725 gas 813800 gas
        owner = msg.sender;
        balanceOf[tx.origin] = msg.sender.balance;
```

```
data: 0x578...cb8f9
```

Conclusion: In Solidity, the execution of transactions within smart contracts is a fundamental aspect of decentralized application (DApp) development on the Ethereum blockchain. To facilitate secure and efficient transactions, Solidity provides a range of features, with the 'transfer' and 'send' functions standing out as essential tools in the developer's toolkit. The 'transfer' and 'send' functions are primarily used to transfer cryptocurrency (ether) or tokens between addresses within the Ethereum network. These functions are critical for various decentralized applications, including financial services, gaming platforms, supply chain management systems, and more. The 'transfer' function is a safer and more robust choice for conducting transactions within smart contracts. It is often recommended for transferring larger sums of ether or tokens. When using 'transfer,' if the destination address is a contract that does not have the required logic to receive funds (e.g., it lacks a payable function), the transaction will revert, ensuring that funds are not sent to contracts that cannot handle them. This safety feature prevents the loss of assets due to erroneous transactions. On the other hand, the 'send' function is less strict and can be used for less critical or smaller transactions. It returns a boolean value indicating the success or failure of the transaction but does not revert if the transaction fails. This means that developers need to manually handle failed 'send' transactions to prevent unintended consequences. Both 'transfer' and 'send' functions are easy to implement in Solidity. Developers can simply call these functions on the address they want to send funds to, followed by the amount of ether or tokens to transfer. Solidity takes care of the low-level details of the transaction, making it a convenient choice for developers.