BCT - Viva Questions

1.Installation of Metamask and study Spending Ether per Transaction.

MetaMask Basics
What is MetaMask?
MetaMask is a crypto wallet and gateway to blockchain applications, especially Ethereum-based apps.
It's available as a browser extension and mobile app.
It stores Ether (ETH) and other ERC-20 tokens and lets users interact with decentralized apps (dApps) securely.

Why is MetaMask important in Blockchain?
MetaMask allows easy access to blockchain technology without needing deep technical knowledge.
It's essential for interacting with Ethereum's blockchain ecosystem, enabling functions like sending/receiving Ether, storing tokens, and accessing dApps directly from the browser.

Setting Up MetaMask
Steps to Install MetaMask:
1. Google Search: Search "MetaMask" on Google.
2. Download: Go to metamask.io and click on "Add to Chrome" to install the extension.
3. Start Setup: Open the extension and click "Get Started."
4.Agree to Terms: Click on "I Agree" to continue.
5. Create a Wallet: Select "Create a Wallet" to start setting up.
6. Create Password: Enter a strong password to secure your account. This password will encrypt your private key, which is essential for security.
7. Backup Phrase: After creating the password, you'll receive a 12-word Secret Backup Phrase. This phrase is critical for account recovery.
8. Complete Setup: After these steps, your MetaMask wallet is ready to use.

Important Terms in MetaMask
Private Key:
A unique code that allows access to your wallet and funds. Keep this secure because anyone with the private key can access your assets.

Secret Backup Phrase:
A 12-word phrase used for recovering your account. Write this down and store it securely; it's the only way to recover your wallet if you lose access.

Spending Ether on Transactions

How Does Spending Ether Work in MetaMask?

When you send Ether or interact with dApps, a small transaction fee (gas fee) is charged.
Gas Fee: This is paid in Ether and compensates the network for processing your transaction.
MetaMask shows options to adjust gas fees (faster transactions cost more).

Common Viva Questions

1. What is MetaMask, and what does it do?
   MetaMask is a crypto wallet and browser extension for managing Ethereum-based assets and connecting to blockchain applications.

2. How does MetaMask help with Ethereum transactions?
   MetaMask securely stores Ether and tokens, allows sending and receiving transactions, and connects users with Ethereum dApps.

3. What is a Private Key, and why is it important?
   A private key is a unique code that grants access to your assets in the wallet. Keeping it secure is crucial, as anyone with access to your private key can control your funds.

4. What is a Secret Backup Phrase, and why is it used?
   A Secret Backup Phrase is a 12-word phrase used to back up or restore a MetaMask wallet. It's the only way to recover the wallet if access is lost.

5. What is "Gas" in the context of Ethereum?
   Gas refers to the transaction fees needed to process and validate actions on the Ethereum network. The fee is paid in Ether, and users can adjust gas prices for faster or slower processing.

6. Why is a strong password important in MetaMask?
   A strong password is used to encrypt the private key within MetaMask, adding a layer of security to prevent unauthorized access.

7. What happens if you lose the Secret Backup Phrase?
   If you lose the backup phrase, you won't be able to recover your MetaMask wallet if you forget your password or lose access to your device.

Summary

MetaMask simplifies the management of digital assets and access to blockchain applications, making it a vital tool for users exploring blockchain technology. Through strong encryption, private key management, and gas fees, it ensures secure and effective interaction with the Ethereum network.

2. Write a smart contract on a test network, for bank account of a customer for following operations.

Bank Smart Contract for Ethereum Test Network

Objective:
This smart contract creates a bank account system for users, enabling basic operations like creating an account, depositing money, withdrawing money, and checking the account balance.

Solidity Code

```solidity
pragma solidity >= 0.7.0;

// Smart contract for a simple bank account with basic functionalities
contract Bank {
    mapping(address => uint) public user_account;        // Stores the balance for each user account
    mapping(address => bool) public user_exist;        // Tracks if a user account exists

    // Function to create an account with an initial deposit
    function create_account() public payable returns (string memory) {
        require(user_exist[msg.sender] == false, "Account already created!");
        user_account[msg.sender] = msg.value;           // Sets the initial balance to the amount sent with transaction
        user_exist[msg.sender] = true;                // Marks the user as having an account
        return "Account created";
    }

    // Function to deposit a specific amount into an account
    function deposit(uint amount) public payable returns (string memory) {
        require(user_exist[msg.sender] == true, "Account not created!");  // Checks if account exists
        require(amount > 0, "Amount should be greater than 0");
        user_account[msg.sender] += amount;              // Increases balance by the specified amount
        return "Amount deposited successfully";
    }

    // Function to withdraw a specific amount from the account
    function withdraw(uint amount) public payable returns (string memory) {
```

```
        require(user_exist[msg.sender] == true, "Account not created!");  // Checks if account exists
        require(amount > 0, "Amount should be greater than 0");
        require(user_account[msg.sender] >= amount, "Insufficient balance");
          user_account[msg.sender] -= amount;                // Decreases balance by the specified amount
        return "Amount withdrawn successfully";
    }

    // Function to check the current balance of the account
    function account_balance() public view returns (uint) {
        return user_account[msg.sender];
    }

    // Function to check if the user account exists
    function account_exists() public view returns (bool) {
        return user_exist[msg.sender];
    }
}
```

Explanation of Functions

1. create_account:
   Purpose: Allows a user to create an account with an initial deposit.
   How it works: Checks if an account already exists for the user. If not, it initializes the account balance with the Ether sent and marks the account as created.

2. deposit:
   Purpose: Allows users to deposit Ether into their account.
   How it works: Verifies that the account exists and the deposit amount is greater than zero, then adds the deposit to the user's balance.

3. withdraw:
   Purpose: Allows users to withdraw Ether from their account.
   How it works: Checks if the account exists, if the amount is greater than zero, and if there is enough balance. If these conditions are met, it deducts the specified amount from the balance.

4. account_balance:
   Purpose: Displays the current balance in the user's account.
   How it works: Returns the user's current balance without making any changes.

5. account_exists:
   Purpose: Checks if an account has been created for the user.
   How it works: Returns `true` if the account exists; otherwise, `false`.

Common Viva Questions

1. What is the purpose of the `create_account` function?
It allows users to create a new bank account with an initial deposit.

2. How does the `deposit` function work?
It checks if the account exists, verifies that the deposit amount is greater than zero, and then adds the specified amount to the user's account balance.

3. What happens if a user tries to withdraw more than their balance?
The contract will not allow the withdrawal and will throw an error message: "Insufficient balance."

4. What is `mapping`, and why is it used in this contract?
`mapping` is a data structure in Solidity used to store key-value pairs. Here, it stores user balances and tracks account existence.

5. Why do we use `require` statements?
require` statements are used to enforce conditions that must be true for the code to proceed. If a condition is not met, it reverts the transaction and returns an error message.

6. How does the `account_balance` function differ from `withdraw`?
`account_balance` is a view function that only reads the user's balance without changing it, whereas `withdraw` modifies the balance.

7. What would happen if a user called `create_account` twice?
- The contract would prevent this by checking `user_exist[msg.sender]`. If the account already exists, it returns an error: "Account already created!"

Summary
This smart contract implements a basic banking system, allowing users to create an account, deposit, withdraw funds, and check their balance. Using mappings, it stores individual user data securely, and `require` statements ensure only valid transactions are processed.

3.Write a program in solidity to create student data. Use the following constructs:

Solidity Code for Student Data Management

```solidity
pragma solidity >= 0.7.0;

contract Student_management {

   // Structure to define a student with ID, name, and department
   struct Student {
      int stud_id;
      string Name;
      string Department;
   }

   // Array to store students
   Student[] Students;

   // Function to add a student to the array
   function add_stud(int stud_id, string memory Name, string memory Department) public {
       Student memory stud = Student(stud_id, Name, Department);  // Create a new student struct
      Students.push(stud);                                  // Add student to the array
   }

   // Function to retrieve a student's information by their ID
   function getStudent(int stud_id) public view returns (string memory, string memory) {
      for (uint i = 0; i < Students.length; i++) {
         Student memory stud = Students[i];
         if (stud.stud_id == stud_id) {
             return (stud.Name, stud.Department);               // Return name and department if ID matches
         }
      }
       return ("Name Not Found", "Department Not Found");         // Return default message if not found
   }

   // Fallback function to handle unexpected Ether or function calls
   fallback() external payable {
       Students.push(Student(7, "XYZ", "Mechanical"));              // Adds a default student entry
   }
```

}
```

Explanation of Functions

1. add_stud:
Purpose: Adds a new student record to the `Students` array.
How it works: Takes `stud_id`, `Name`, and `Department` as inputs, creates a new `Student` struct, and appends it to the `Students` array.

2. getStudent:
Purpose: Retrieves a student's name and department based on their ID.
How it works: Iterates through the `Students` array to find the student with the matching `stud_id`. If found, it returns the student's name and department. If not found, it returns "Name Not Found" and "Department Not Found".

3. fallback:
Purpose: Handles unexpected Ether transfers or function calls.
How it works: Adds a default student with `stud_id` 7, name "XYZ", and department "Mechanical" whenever the fallback function is triggered.

Common Viva Questions

1. What is the purpose of the `struct` keyword in Solidity?
- `struct` is used to define custom data types. Here, it defines a `Student` with attributes like `stud_id`, `Name`, and `Department`.

2. How does the `add_stud` function work?
- It takes the student ID, name, and department as parameters, creates a new student, and adds it to the `Students` array.

3. How does the `getStudent` function find a student?
- It iterates through the `Students` array, checks each student's ID, and returns the name and department if the ID matches.

4. What is the fallback function, and why is it used here?
- The fallback function is executed if a contract receives Ether without a function call or an invalid function call. Here, it adds a default student entry when triggered.

5. What would happen if you call `getStudent` with an ID that doesn't exist?
- It returns "Name Not Found" and "Department Not Found".

6. What is the purpose of using `memory` in the function parameters?
- `memory` specifies a temporary storage location for data used only within the function, and it is cleared after function execution. Here, it's used for `Name` and `Department`.
```

Here are some theory questions related to the smart contract and Solidity concepts that could come up during a viva or examination:

Theory Questions

1. What is Solidity, and why is it commonly used for developing smart contracts?
   - Solidity is an object-oriented programming language for developing smart contracts on the Ethereum blockchain. It allows developers to create decentralized applications with features like token creation, identity verification, and asset management.

2. What is a smart contract? How does it differ from traditional contracts?
   - A smart contract is a self-executing contract with the terms of the agreement directly written into code. Unlike traditional contracts, smart contracts are automatically enforced without the need for intermediaries and are immutable once deployed on the blockchain.

3. What is the purpose of a `struct` in Solidity, and how does it help in data management?
   - `struct` in Solidity allows developers to create complex data types that can bundle multiple properties into a single type, like `Student` in this example. This makes it easier to manage and organize related data within the contract.

4. What are the different storage locations available in Solidity, and how do they differ?
   - Solidity has three main storage locations:
   - Storage: Permanent storage on the blockchain. Used for state variables.
   - Memory: Temporary storage for variables within functions; it's cleared after function execution.
   - Calldata: A non-modifiable temporary storage location used specifically for function parameters.

5. What is the significance of the `memory` keyword in function parameters?
   - The `memory` keyword specifies that variables are stored temporarily during function execution. It's used for data that doesn't need to persist after the function finishes, saving on storage costs.

6. How does the `fallback` function work in Solidity, and what are its common uses?
   - The `fallback` function is a special function triggered when a contract receives Ether without specific instructions or when an unknown function is called. It's used for handling unexpected calls or funds, as seen here where it adds a default student.

7. Explain the `require` function and its role in Solidity contracts.
   - `require` is a function that checks for conditions. If the condition fails, it halts the function execution and reverts any changes, helping ensure that the contract runs only under valid conditions.

8. What is the purpose of the `public` and `view` keywords in Solidity?

- public: This visibility modifier allows functions and variables to be accessed from outside the contract.

- view: This modifier indicates that the function doesn't modify the contract's state, allowing it to read data without changing it.

9. What is the difference between a transaction and a call in Ethereum?

- A transaction changes the blockchain's state, costs gas fees, and requires confirmation by miners. A call is used to read data from the blockchain without altering it, and it is free.

10. How does the `mapping` data structure work in Solidity, and why is it useful?

- A `mapping` is a key-value store in Solidity that associates a unique key with a specific value, allowing for efficient lookups. In this example, mappings could be used to store students' data by ID for quick access.

11. What are gas fees, and why are they important in Ethereum?

- Gas fees are charges required to execute transactions or operations on the Ethereum network. They prevent misuse by adding a cost to computations and incentivizing miners to validate transactions.

12. How does the Ethereum Virtual Machine (EVM) execute smart contracts?

- The EVM is a decentralized runtime environment that executes smart contract code. It runs on nodes within the Ethereum network, ensuring contract execution consistency across all nodes.

13. What are the security risks associated with Solidity smart contracts, and how can they be mitigated?

- Common risks include reentrancy attacks, integer overflows, and access control issues. Mitigation strategies include using secure coding practices, conducting audits, and utilizing security-focused tools like OpenZeppelin.

14. What is the purpose of using events in Solidity?

- Events in Solidity allow contracts to emit messages during execution, enabling external applications to listen for specific occurrences on the blockchain. They are commonly used for logging and notifying applications of changes.

15. How does immutability work in the context of smart contracts, and what are its advantages and limitations?

- Once deployed, a smart contract's code and state cannot be altered. This ensures trust and reliability but also means bugs or vulnerabilities cannot be fixed unless the contract includes upgradeable logic.

16. Why is Ethereum chosen for deploying smart contracts?

- Ethereum provides a robust blockchain with decentralized computation, supporting smart contracts and decentralized applications (DApps) through its EVM, widespread adoption, and development support.

These questions provide a comprehensive overview of the concepts relevant to smart contracts and Solidity. They cover both basic and intermediate aspects, ideal for a blockchain course viva.

This smart contract is designed to manage student data by allowing the addition of student records and retrieval of details by student ID. The fallback function adds a default student entry if unexpected interactions occur.