Storage Variables in Solidity: Student Notes

Introduction
Storage variables in Solidity are variables whose values are permanently stored on the blockchain. They are a critical aspect of smart contract development, as they maintain the state of a contract between function calls and transactions.

Key Concepts

1. **Persistent Storage:**
   - Storage variables persist for the lifetime of the contract.
   - Modifying a storage variable updates the blockchain state, incurring gas costs.

2. **Location:**
   - Storage variables are stored in a specific location in the Ethereum Virtual Machine (EVM) storage.

3. **Scope:**
   - Defined at the contract level, outside of any functions.

Declaring Storage Variables

Storage variables are declared inside a contract but outside of any function. They can be of any data type supported by Solidity.

Examples:
```solidity
pragma solidity ^0.8.0;

contract StorageExample {
    uint256 public count;  // Unsigned integer storage variable
    address public owner;  // Address storage variable
    bool public isActive;  // Boolean storage variable
    bytes32 public dataHash;  // Fixed-size byte array storage variable
}
```

Modifying Storage Variables

Modifying storage variables changes the contract's state on the blockchain, which requires a transaction and costs gas.

Example:
```solidity
pragma solidity ^0.8.0;

contract StorageExample {
    uint256 public count;
    address public owner;
    bool public isActive;
```

```
    bytes32 public dataHash;

    function setCount(uint256 _count) public {
        count = _count;  // This will cost gas as it modifies blockchain state
    }

    function setOwner(address _owner) public {
        owner = _owner;
    }

    function setActive(bool _isActive) public {
        isActive = _isActive;
    }

    function setDataHash(bytes32 _dataHash) public {
        dataHash = _dataHash;
    }
}
```

Accessing Storage Variables

Storage variables can be read without any gas cost when accessed externally via view or pure functions.

Example:
```solidity
pragma solidity ^0.8.0;

contract StorageExample {
    uint256 public count;
    address public owner;
    bool public isActive;
    bytes32 public dataHash;

    function getCount() public view returns (uint256) {
        return count;  // Reading storage variable is free
    }

    function getOwner() public view returns (address) {
        return owner;
    }

    function getIsActive() public view returns (bool) {
        return isActive;
    }

    function getDataHash() public view returns (bytes32) {
        return dataHash;
    }
```

}
```

### Gas Costs

- **Writing to Storage:** Costs significant gas, as it changes the blockchain state.
- **Reading from Storage:** Free when accessed via external calls using view or pure functions.

### Complex Data Types

Storage variables can also be complex data types like arrays, mappings, and structs.

#### Arrays
```solidity
pragma solidity ^0.8.0;

contract StorageExample {
    uint256[] public numbers;

    function addNumber(uint256 _number) public {
        numbers.push(_number);  // Modify the storage array
    }

    function getNumber(uint256 _index) public view returns (uint256) {
        return numbers[_index];  // Access storage array
    }
}
```

Mappings
```solidity
pragma solidity ^0.8.0;

contract StorageExample {
    mapping(address => uint256) public balances;

    function setBalance(address _account, uint256 _balance) public {
        balances[_account] = _balance;  // Set value in the mapping
    }

    function getBalance(address _account) public view returns (uint256) {
        return balances[_account];  // Get value from the mapping
    }
}
```

Structs
```solidity
pragma solidity ^0.8.0;

```
contract StorageExample {
    struct User {
        string name;
        uint256 age;
    }

    User public user;

    function setUser(string memory _name, uint256 _age) public {
        user = User(_name, _age);  // Set values in the struct
    }

    function getUser() public view returns (string memory, uint256) {
        return (user.name, user.age);  // Get values from the struct
    }
}
```

Conclusion

Storage variables are essential for maintaining state in Solidity smart contracts. They are stored permanently on the blockchain, incurring gas costs when modified. Understanding how to declare, access, and modify storage variables is fundamental for efficient and effective smart contract development.