Solidity Course Work for Blockchain and Cryptocurrency (SEng4306)

CHAPTER 1

Solidity Introduction

Developing the Smart Contract

Smart Contracts:

- Smart contracts are pieces of code deployed on the blockchain.
- They contain variables and functions that execute automatically when certain conditions are met.
- Deploying a Contract:

Compile Solidity code into bytecode.
- Send a transaction containing the bytecode to an Ethereum Virtual Machine (EVM) node.
- The node calculates an address for the new contract.
- Contract Deployment:

Contracts are compiled to creation bytecode.
- The data field in the transaction contains the creation bytecode.
- The to field is left blank to deploy a contract.
- The deployed contract has an address, balance, and runtime bytecode.

Transaction Lifecycle:

- Transactions originate from externally owned accounts (EOAs).
- Transactions occur sequentially on the blockchain.
- Each transaction sets a gas limit to control computational resources.
- Transactions send calldata, targeting specific contract methods.
- Smart contracts can call each other within the same transaction.

value types

Solidity offers various value types to suit different data storage and manipulation needs within smart contracts.

- **Unsigned Integer (uint)**: Represents non-negative whole numbers. The number after "uint" denotes the number of bits, which determines the range of values it can hold. For example, uint8 can hold values from 0 to 255.

- **Integer (int)**: Represents both positive and negative whole numbers. Similar to uint, the number after "int" denotes the number of bits, determining the range of values it can hold. For example, int8 can hold values from -128 to 127.
- **Boolean (bool)**: Represents a binary value that can be either true or false. Used for logical operations and conditional statements.
- **Enum**: Allows you to define a custom data type with a finite set of possible values. Enums are useful for representing a set of options or choices, like the cardinal directions in your example.

storage variables

storage variables declared within contract scope are stored in contiguous storage slots, each occupying 32 bytes.

- Storage Variables: Variables declared within a contract are storage variables and are stored in contiguous storage slots.
- Storage Slots: Solidity assigns each storage variable a unique storage slot, typically represented as hexadecimal values (e.g., 0x0, 0x1).
- Slot Allocation: Solidity automatically allocates storage slots for variables, except for constants. Variables are stored sequentially, with each variable occupying its own slot.
- Cost of Storage Operations: Reading from and writing to storage is relatively expensive compared to other opcodes in Ethereum smart contract execution.
- Packing Variables: Variables can be packed together to optimize storage usage, either automatically by Solidity or manually by the developer.

Functions

Functions in Solidity come with various keywords that define their visibility, accessibility, and behavior.

- Private Functions (private): These functions are accessible only within the contract that defines them. They cannot be called from outside the contract, including derived contracts.

- Pure Functions (pure): Pure functions are restricted from accessing or modifying the contract's state. They can only perform computations and return values based on their input parameters. These are useful for utility functions.

- Internal Functions (internal): Internal functions are accessible within the contract and any derived contracts. They cannot be called externally, meaning from outside the contract hierarchy.

- View Functions (view): View functions are read-only functions that can access, but not modify, the contract's state. They are commonly used for querying data from the blockchain.

- Public Functions (public): Public functions can be called both internally from within the contract and externally from outside the contract. They are accessible to anyone interacting with the contract.
-

Payable Functions (payable): Payable functions can receive Ether along with function calls. They are used to accept payments or transfer funds within a contract.

Regarding returning values, Solidity functions can specify a return type and use the returns keyword to indicate what they will return. In your example, the add function is a private pure function that returns the sum of two numbers. It's called in the constructor to initialize the sum variable with the result of the addition.