Practical session for blockchain

(8) Token Crowdsale

Jungyoon Song[1] Giyeong Lee[1]

[1]Financial Risk Engineering Lab.
Seoul National University

May 16, 2022

1 Structure and Procedures of Token Crowdsale Code

2 Token Crowdsale Contract

3 Execution: Token Crowdsale

Structure and Procedures of Token Crowdsale Code

- **Previous lecture**
  1. Use **inheritance** to increase the re-usability of the existing code
  2. **Interworking** between smart contracts
- **In this lecture,**
  1. Learn how cryptocurrency is offered, including ICO & ITO, IEO, STO
  2. Based on the token contract learned so far, implement token crowdsale contract

Token Crowdsale : ICO / ITO

- Initial Coin Offering(ICO), Initial Token Offering(ITO)
  - Launch an ICO as a way to raise funds in each country's currency or cryptocurrency
  - An initial coin offering (ICO) and an initial token offering (ITO) is the cryptocurrency industry's equivalent to an initial public offering (IPO)
- Advantages
  - Sales procedures are simple and cost less than others
  - Provides fast liquidity in short periods of time
  - Lack of government regulation
- Disadvantages
  - There is a lot of fraud because of lack of regulation
  - * ICO Fraud Cases (2018): Pincoin & iFan($660M), Plexcoin($15M), Bitcard($5M), Opair & Ebitz ($2.9M), . . .

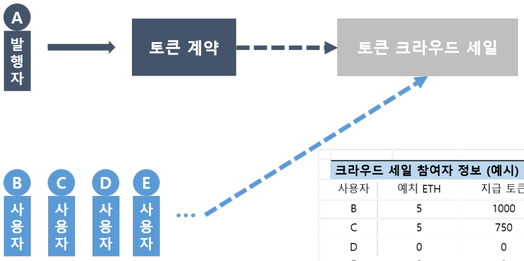
Token Crowdsale : IEO

- Initial Exchange Offering(IEO)
  - The act of receiving a token issued by an ICO project on a cryptocurrency exchange and selling it to a limited investor (member of the exchange)
  - The exchange has an incentive to formalize the initial disclosure process and evaluate the soundness of the project to reduce risk
- Advantages
  - Provide minimal project reliability to investors
  - The exchange can expect to attract new customers and earn commission fees, and it can also expect to increase reliability of the exchange if it is listed in the future
- Disadvantages
  - More expensive than ICOs, less initial liquidity
  - If there is a problem with the exchange itself, it is more likely to mislead investors than typical ICOs

Token Crowdsale : STO

- Security token & Utility Token
  - **Security token** represents ownership shares in a company that does business using blockchain technology
  - **Utility token** is promotional tools that grant holders special access or promotions for future product or service launches
- Advantages
  - Most reliable under the influence of authorities because it deals with real assets
- Disadvantages
  - Offering procedures are extremely complex, costly, and subject to strict regulations

Token Crowdsale : Summary



크라우드 세일 정보 (예시)	
판매 토큰	MYTOKEN
판매 기간	15분
판매 가격	100 토큰 / ETH
최대 판매 수량	10000
목표 금액	10 ether

크라우드 세일 참여자 정보 (예시)

사용자	예치 ETH	지급 토큰
B	5	1000
C	5	750
D	0	0
E	0	0

$\}$ 다른 이유?

1 Structure and Procedures of Token Crowdsale Code

2 Token Crowdsale Contract

3 Execution: Token Crowdsale

Token Crowdsale Contract

[Ex. 1] Crowdsale Contract

```
pragma solidity > 0.7.0 < 0.8.0;

contract Crowdsale is Owned {
    uint256 public fundingGoal;
    uint256 public startTime;
    uint256 public deadline;
    uint256 public price;
    uint256 public transferableToken;
    uint256 public soldToken;
    MyToken public tokenReward;
    bool public fundingGoalReached;
    bool public isOpened;
    mapping (address => Property) public fundersProperty;

    struct Property {
        uint256 paymentEther;
        uint256 reservedToken;
        bool withdrawed;
    }
```

Crowdsale Contract

[Ex. 1] Crowdsale Contract

```
event CrowdsaleStart(uint fundingGoal, uint deadline, uint transferableToken, address beneficiary);
event ReservedToken(address backer, uint amount, uint token);
event CheckGoalReached(address beneficiary, uint fundingGoal, uint amountRaised,
                       bool reached, uint raisedToken);
event WithdrawalToken(address addr, uint amount, bool result);
event WithdrawalEther(address addr, uint amount, bool result);

modifier afterDeadline() { require(block.timestamp >= deadline); _; }

constructor(uint _fundingGoalInEthers, uint _transferableToken, uint _amountOfTokenPerEther,
            MyToken _tokenAddr) {
    fundingGoal = _fundingGoalInEthers * 1 ether;
    price = 1 ether / _amountOfTokenPerEther;
    transferableToken = _transferableToken;
    tokenReward = MyToken(_tokenAddr);
}
```

Crowdsale Contract

[Ex. 1] Crowdsale Contract

```
function currentSwapRate() public view returns(uint) {
    if (startTime + 3 minutes > block.timestamp) { return 100; }
    else if (startTime + 5 minutes > block.timestamp) { return 50; }
    else if (startTime + 10 minutes > block.timestamp) { return 20; }
    else { return 0; }
}

function reserve() public payable {
    require(isOpened && block.timestamp < deadline);

    uint amount = msg.value;
    uint token = amount * (100 + currentSwapRate()) / price / 100;

    require(token > 0 && soldToken + token <= transferableToken);

    fundersProperty[msg.sender].paymentEther += amount;
    fundersProperty[msg.sender].reservedToken += token;
    soldToken += token;
    emit ReservedToken(msg.sender, amount, token);
}
```

Crowdsale Contract

[Ex. 1] Crowdsale Contract

```
function start(uint _durationInMinutes) public onlyOwner {
    require(fundingGoal > 0 && price > 0);
    require(address(tokenReward) > address(0) && transferableToken > 0
            && tokenReward.balanceOf(address(this)) >= transferableToken);
    require(_durationInMinutes > 0 && startTime == 0);

    startTime = block.timestamp;
    deadline = block.timestamp + _durationInMinutes * 1 minutes;
    isOpened = true;
    emit CrowdsaleStart(fundingGoal, deadline, transferableToken, owner);
}

function getRemainingTimeEthToken() public view returns(uint min, int shortage, uint remainToken) {
    if (block.timestamp < deadline) {
        min = (deadline - block.timestamp) / (1 minutes);
    }
    shortage = int256(fundingGoal - address(this).balance) / (1 ether);
    remainToken = transferableToken - soldToken;
}
```

Crowdsale Contract

[Ex. 1] Crowdsale Contract

```
function checkGoalReached() public afterDeadline {
    require(isOpened);

    if (address(this).balance >= fundingGoal) {
        fundingGoalReached = true;
    }

    isOpened = false;
    emit CheckGoalReached(owner, fundingGoal, address(this).balance, fundingGoalReached, soldToken);
}
```

Crowdsale Contract

[Ex. 1] Crowdsale Contract

```
function withdrawalOwner() public onlyOwner {
    require(!isOpened);

    if (fundingGoalReached) {
        uint amount = address(this).balance;
        if (amount > 0) {
            bool result = payable(msg.sender).send(amount);
            emit WithdrawalEther(msg.sender, amount, result);
        }

        uint val = transferableToken - soldToken;
        if (val > 0) {
            tokenReward.transfer(msg.sender, transferableToken - soldToken);
            emit WithdrawalToken(msg.sender, val, true);
        }
    } else {
        uint val = tokenReward.balanceOf(address(this));
        tokenReward.transfer(msg.sender, val);
        emit WithdrawalToken(msg.sender, val, true);
    }
}
```

Crowdsale Contract

[Ex. 1] Crowdsale Contract

```
    function withdrawal() public {
        require(!isOpened);
        require(!fundersProperty[msg.sender].withdrawed);
        if (fundingGoalReached) {
            if (fundersProperty[msg.sender].reservedToken > 0) {
                tokenReward.transfer(msg.sender, fundersProperty[msg.sender].reservedToken);
                fundersProperty[msg.sender].withdrawed = true;
                emit WithdrawalToken(msg.sender, fundersProperty[msg.sender].reservedToken,
                                     fundersProperty[msg.sender].withdrawed);
            }
        } else {
            if (fundersProperty[msg.sender].paymentEther > 0) {
                if (payable(msg.sender).send(fundersProperty[msg.sender].paymentEther)) {
                    fundersProperty[msg.sender].withdrawed = true;
                }
                emit WithdrawalEther(msg.sender, fundersProperty[msg.sender].reservedToken,
                                     fundersProperty[msg.sender].withdrawed);
            }
        }
    }
}
```

1 Structure and Procedures of Token Crowdsale Code

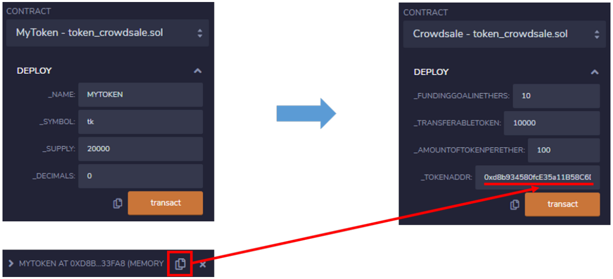
2 Token Crowdsale Contract

3 Execution: Token Crowdsale

Execution: Crowdsale

Owner(Issuer) A :

- Deploy a crowdsale contract after deploying a token contract



CONTRACT
MyToken - token_crowdsale.sol
DEPLOY
_NAME: MYTOKEN
_SYMBOL: tk
_SUPPLY: 20000
_DECIMALS: 0
transact

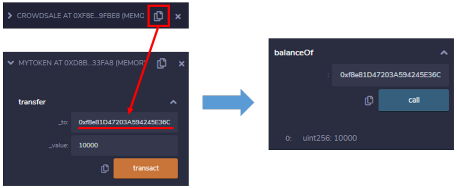
MYTOKEN AT 0XD8B...33FA8 (MEMORY)

CONTRACT
Crowdsale - token_crowdsale.sol
DEPLOY
_FUNDINGGOALINETHERS: 10
_TRANSFERABLETOKEN: 10000
_AMOUNTOFTOKENPERETHER: 100
_TOKENADDR: 0xd8b934580fcE35a11B58C6
transact

Execution: Crowdsale

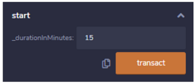
Owner(Issuer) A :

- Transfer tokens from token contract to crowdsale contract (left),
- Start after setting a period in a crowdsale contract (right)



```
> CROWDSALE AT 0XF8E...9FBE8 (MEMO
v MYTOKEN AT 0XD8B...33FA8 (MEMORY
transfer
_to: 0xf8e81D47203A594245E36C
_value: 10000
transact
```

```
balanceOf
: 0xf8e81D47203A594245E36C
call
0: uint256: 10000
```

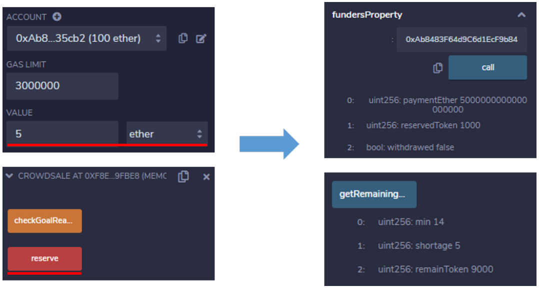


```
start
_durationInMinutes: 15
transact
```

Execution: Crowdsale

User(Buyer) B :

- Deposit a certain amount of ether and check for changes



```
ACCOUNT
0xAb8...35cb2 (100 ether)
GAS LIMIT
3000000
VALUE
5 ether
```

```
CROWDSALE AT 0XF8E...9FBE8 (MEMC
checkGoalRea...
reserve
```

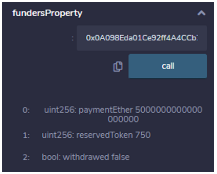
```
fundersProperty
: 0xAb8483F64d9C6d1EcF9b84
call
0: uint256: paymentEther 5000000000000000000
1: uint256: reservedToken 1000
2: bool: withdrawed false
```

```
getRemaining...
0: uint256: min 14
1: uint256: shortage 5
2: uint256: remainToken 9000
```

Execution: Crowdsale

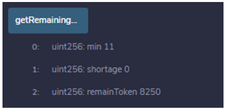
User(Buyer) C, D, ...

- Likewise, deposit a certain amount of ether and achieve the goal by making shortage less than zero
  - In the example below, an additional token rate of 50% is applied after 3 minutes, and 750 tokens are allocated after 5 ETH deposits



```
0x0A098Eda01Ce92ff4A4CCb

0: uint256: paymentEther 5000000000000
   000000
1: uint256: reservedToken 750
2: bool: withdrawed false
```

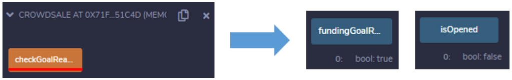


```
0: uint256: min 11
1: uint256: shortage 0
2: uint256: remainToken 8250
```

Execution: Crowdsale

Owner(Issuer) A

- After end of contract, Run the "checkGoalReached" function to check if the goal is achieved



- Issuer and buyer execute withdrawal respectively and check balance fluctuation
- Adjust the deposit ether to prevent the goal from being achieved to re-practice and check the difference in results

References

- https://docs.soliditylang.org/en/v0.8.13/