```
pragma solidity ^0.4.11;

import './IERC20.sol';
import './SafeMath.sol';

contract KPRToken is IERC20 {

    using SafeMath for uint256;



    //public variables
    string public constant symbol="KPR";
    string public constant name="KPR Coin";
    uint8 public constant decimals=18;

    //1 ETH = 2,500 KPR
    uint56 public  RATE = 2500;

    //totalsupplyoftoken
    uint public totalSupply = 100000000 * 10 ** uint(decimals);

    uint public buyabletoken = 90000000 * 10 ** uint(decimals);
    //where the ETH goes
    address public owner;

    //map the addresses
    mapping(address => uint256) balances;
    mapping(address => mapping(address => uint256)) allowed;
    // 1514764800 : Jan 1 2018
    uint phase1starttime = 1517443200; // Phase 1 Start Date Feb 1 2018
    uint phase1endtime = 1519257600;  // Phase 1 End Date Feb 22 2018
    uint phase2starttime = 1519862400;  // Phase 2 Start Date March 1 2018
    uint phase2endtime = 1521676800; // Phase 2 End Date March 22 2018
    uint phase3starttime = 1522540800;  // Phase 3 Start Date May 1 2018
    uint phase3endtime = 1524355200; // Phase 3 End Date May 22 2018


    //create token function = check

    function() payable {
        buyTokens();
    }

    function KPRToken() {
        owner = msg.sender;
        balances[owner] = totalSupply;
    }
```

```
function buyTokens() payable {

    require(msg.value > 0);
    require(now > phase1starttime && now < phase3endtime);
    uint256 tokens;

    if (now > phase1starttime && now < phase1endtime){

        RATE = 3000;
        setPrice(msg.sender, msg.value);
    } else if(now > phase2starttime && now < phase2endtime){
        RATE = 2000;
        setPrice(msg.sender, msg.value);
        // tokens = msg.value.mul(RATE);
        // require(tokens < buyabletoken);
        // balances[msg.sender]=balances[msg.sender].add(tokens);
        // balances[owner] = balances[owner].sub(tokens);
        // buyabletoken = buyabletoken.sub(tokens);
        // owner.transfer(msg.value);

    } else if(now > phase3starttime && now < phase3endtime){

        RATE = 1000;
        setPrice(msg.sender, msg.value);
    }
}

function setPrice(address receipt, uint256 value){
    uint256 tokens;
    tokens = value.mul(RATE);
    require(tokens < buyabletoken);
    balances[receipt]=balances[receipt].add(tokens);
    balances[owner] = balances[owner].sub(tokens);
    buyabletoken = buyabletoken.sub(tokens);
    owner.transfer(value);
}

function balanceOf(address _owner) constant returns(uint256 balance) {

    return balances[_owner];

}

function transfer(address _to, uint256 _value) returns(bool success) {

    //require is the same as an if statement = checks
    require(balances[msg.sender] >= _value && _value > 0 );

    balances[msg.sender] = balances[msg.sender].sub(_value);
```

```
        balances[_to] = balances[_to].add(_value);

        Transfer(msg.sender, _to, _value);
        return true;
    }

    function transferFrom(address _from, address _to, uint256 _value) returns (bool
success) {

        //checking if the spender has permission to spend and how much
        require( allowed[_from][msg.sender] >= _value && balances[_from] >= _value &&
_value > 0);

        //updating the spenders balance
        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        Transfer(_from, _to, _value);
        return true;
    }

    function approve(address _spender, uint256 _value) returns(bool success) {

        //if above require is true,approve the spending
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
        return true;
    }

    function allowance(address _owner, address _spender) constant returns(uint256
remaining) {

        return allowed[_owner][_spender];

    }

    event Transfer(address indexed_from, address indexed_to, uint256 _value);
    event Approval(address indexed_owner, address indexed_spender, uint256 _value);


}
```