Assignment 6 - Develop an employee vesting contract

300pts
Due Apr 11

Build a contract that records when an employee starts and how many tokens an employee has vested. The vesting schedule is a once-every-3-months schedule. When the employee starts create an account for the employee, then every 3 months calculate the amount of tokens that they receive and transfer these from an ERC 20 / ERC 777 / ERC1155 contract to the employee. After 2.5 years (10 quarters) the employee is fully vested.

Create an ERC-20/777/1155 contract with 1 billion (1,000,000,000) tokens as the company “tokens”. This is where you will transfer your tokens from. Each transfer is to the employee’s account in the ERC-20/777/1155 contract.

File: Vest90Days.sol

```
 1: // SPDX-License-Identifier: MIT
 2: pragma solidity >=0.8.0 <0.9.0;
 3:
 4: import "@openzeppelin/contracts/access/Ownable.sol";
 5:
 6: contract Vest90Days is Ownable {
 7:
 8:     address VvvTokenContractAddrss ;
 9:
10:     struct employeeData{
11:         string name;
12:         address owner;
13:         uint256 startTime;
14:         uint256 endTime;
15:         uint256 tokensVested;
16:         bool exists;
17:         bool hasExited;
18:     }
19:
20:     mapping(address => employeeData) perEmployeeData;
21:     address[] empList;
22:
23:     event NewEmployee(address indexed, uint256 startTime);
24:     event VestedTokens(address indexed, uint256 nTokens);
25:     event EmployeeExit(address indexed, uint256 endTime);
26:
27:     constructor( address _addr ) {
28:         VvvTokenContractAddrss = _addr;
29:     }
30:
31:     function startNewEmployee( address _employeeAddress, string memory _name ) public onlyOwner {
32:         employeeData memory newEmployee;
33:         newEmployee.name = _name;
34:         newEmployee.owner = _employeeAddress;
35:         newEmployee.startTime = block.timestamp;
36:         newEmployee.exists = true;
37:         perEmployeeData[_employeeAddress] = newEmployee;
38:         empList.push(_employeeAddress);
39:         emit NewEmployee(_employeeAddress, block.timestamp);
40:     }
41:
42:     function employeeExit( address _employeeAddress ) public onlyOwner {
43:         employeeData memory emp;
44:         emp = perEmployeeData[_employeeAddress];
```

```
45:         if ( emp.exists ) {
46:             emp.hasExited = true;
47:             emp.endTime = block.timestamp;
48:         }
49:         perEmployeeData[_employeeAddress] = emp;
50:         emit EmployeeExit(_employeeAddress, block.timestamp);
51:     }
52:
53:     function calculateVesting (address _anEmployee ) public onlyOwner {
54:         uint256 nTokens;
55:         // TODO calculate.
56:         // TODO use address of ERC777 to transfer tokens to _anEmployee
57:         emit VestedTokens(_anEmployee, nTokens);
58:     }
59:
60:     function vestedTokens ( address _anEmployee ) public view returns ( uint256 ) {
61:         employeeData memory emp;
62:         emp = perEmployeeData[_anEmployee];
63:         if ( emp.exists ) {
64:             return emp.tokensVested;
65:         }
66:         return 0;
67:     }
68:
69:
70:
71:     // List Employees by address
72:     function nEmployees() public view returns ( uint256 ) {
73:         return( empList.length );
74:     }
75:     function getName ( uint256 nth ) public view returns ( string memory ) {
76:         require(nth >= 0  && nth < empList.length, "nth out of range.");
77:         address a;
78:         a = empList[nth];
79:         employeeData memory emp;
80:         emp = perEmployeeData[a];
81:         if ( emp.exists ) {
82:             return emp.name;
83:         }
84:         return ( "" );
85:     }
86:     function getStartTime ( uint256 nth ) public view returns ( uint256 ) {
87:         require(nth >= 0  && nth < empList.length, "nth out of range.");
88:         address a;
89:         a = empList[nth];
90:         employeeData memory emp;
91:         emp = perEmployeeData[a];
92:         if ( emp.exists ) {
93:             return emp.startTime;
94:         }
95:         return ( 0 );
96:     }
97:
98:     // TODO - add other accessor functions for data.
99:
100: }
```

Develop the test of this contract (the TODO's) and a set of test for it.