pragma solidity ^0.4.24;

pragma experimental ABIEncoderV2;

//用於進行安全的加減乘除

library SafeMath {

function add(uint a, uint b) internal pure returns (uint c) {

c = a + b;

require(c >= a);

}

function sub(uint a, uint b) internal pure returns (uint c) {

require(b <= a);

c = a - b;

}

function mul(uint a, uint b) internal pure returns (uint c) {

c = a \* b;

require(a == 0 || c / a == b);

}

function div(uint a, uint b) internal pure returns (uint c) {

require(b > 0);

c = a / b;

}

}

//ERC20界面的functions還有events的宣告

contract ERC20Interface {

//總共token的數量

function totalSupply() public constant returns (uint);

//回傳tokenOwner這個address的token數量

function balanceOf(address tokenOwner) public view returns (uint);

//從owner轉賬給to這個address GPA計算後個 token

function transfer(address to) public returns (bool success);

//從from轉到to tokens個token 的event

event Transfer(

address indexed from,

address indexed to,

uint tokens

);

}

//投標選課智能合約

contract FixedSupplyToken is ERC20Interface{

//使用safemath這個library

using SafeMath for uint;

//記錄合約所有者

address public owner;

//總發幣量

uint private \_totalSupply;

//記錄輪數

uint public turn;

//記錄某輪課程數

uint public coursecounter;

//balance用於記錄每個address的token的餘額

mapping (address => uint) private \_balances;

//bids用於記錄每個courseCode下某個地址的token的投標量

mapping (bytes32 => mapping(address => uint)) private \_bids;

//courses用於記錄課程代碼對應的課的名字

mapping (bytes32 => string) private \_courses;

//studentLimit用於記錄每輪 每個courseCode的課程對應的修課人數限制

mapping (uint => mapping (bytes32 => uint)) private \_studentLimit;

//haveBidBefore用於記錄某個地址在某個courseCode裡面是否是第一次投標

mapping (address => mapping(bytes32 => uint)) private \_haveBidBefore;

//addresses用於記錄每個courseCode某投標順位的地址

mapping (bytes32 => mapping(uint => address)) private \_addresses;

//biddercounter用於記錄每個courseCode的投標總人數

mapping (bytes32 => uint) private \_bidderCounter;

//winners用於記錄每個turn 每個courseCode下 , token投標數量大為順位的, 對應的贏家的address

mapping (uint => mapping(bytes32 => mapping(uint => address))) public \_winners;

//winnersBids用於記錄每個turn courseCode下 token投標數量大為順位的 對應的token投標量

mapping (uint => mapping (bytes32 => mapping(uint => uint))) public \_winnersBids;

//\_WinnersTotalNumber用於記錄每個courseCode下 投標贏家的數量

mapping (bytes32 => uint) private \_WinnersTotalNumber;

//\_turns用於輪次更替后的清空和保存 輪次 順位 課程代碼

mapping (uint => mapping(uint => bytes32)) private \_turns;

//\_ClassAvailable用於輪次記錄更替后的可用課程

mapping (uint => bytes32) public \_ClassAvailable;

//\_bidInformation turn 順序 課程 投標量

mapping (uint => mapping(address => mapping (uint => mapping (bytes32 => uint)))) private \_bidInformation;

//\_bidInformation turn 順序 地址 課程bytes32

mapping (uint => mapping (address => mapping (uint => bytes32))) private \_bidInformation2;

//\_bidInformation turn 順序 地址 課程coursecode

mapping (uint => mapping (address => mapping (uint => string))) private \_bidInformation4;

//\_bidInformation turn 地址 index

mapping (uint => mapping (address => uint)) private \_bidInformation3;

//\_bidInformation turn ADDRESS index 課程

mapping (uint => mapping (address => mapping (uint => winInfo))) private \_winList;

//turn index hashValue

mapping (uint => mapping (uint => hashInfo)) private \_hashValue;

//turn index address

mapping (uint => mapping (address => mapping(uint => uint))) private \_hashValueSortByAddress;

//turn index

mapping (uint => uint) private \_hashValueCounter;

mapping (uint => mapping (address => uint)) private \_hashValueSortByAddressCounter;

mapping (uint => mapping (address => mapping (uint => buildHash))) private \_HiddenBid;

mapping (uint => mapping (address => uint)) private \_HiddenBidCounter;

mapping (uint => mapping (address => mapping(uint => uint))) private \_verification;

mapping (uint => mapping (uint => address)) private \_allAddress;

mapping (uint => uint) private \_allAddressCounter;

mapping (uint => mapping (address => bool)) private \_allAddressChecker;

struct courseInfo

{

string courseCode;

bytes32 hexCode;

uint studentLimit;

string courseName;

}

struct bidInfo

{

string courseCode;

string courseName;

uint haveBidToken;

}

struct hashInfo

{

address bidder;

bytes32 hashValue;

}

struct buildHash

{

string courseCode;

uint bid;

string String;

}

struct winInfo

{

string courseCode;

uint bid;

}

//事件 用於公佈贏家 包括 輪數 課程代碼 排序 贏家地址 投標量

event Winning\_bidder(

uint turn,

bytes32 classCode,

string classname,

uint index,

address winner,

uint tokens

);

//事件 用於公佈剩餘課程 包括 輪數 課程代碼 名字 剩餘人數

event classes(

uint turn,

bytes32 classCode,

string classname,

uint studentlimit

);

//建構子，指定owner、totalsupply、owner的balance

constructor() public {

owner = msg.sender;

\_totalSupply = 10000000000;

\_balances[owner] = \_totalSupply;

turn = 0;

coursecounter = 0;

}

//function中附加onlyOwner使只有owner能操作此function

modifier onlyOwner {

require(msg.sender == owner);

\_;

}

//任何人 查詢totalsupply

function totalSupply() public constant returns (uint){

return \_totalSupply;

}

//將字串轉換成bytes32

function stringToBytes32(string source)internal pure returns (bytes32 result) {

bytes memory tempEmptyStringTest = bytes(source);

if (tempEmptyStringTest.length == 0) {

return 0x0;

}

assembly {

result := mload(add(source, 32))

}

return result;

}

//將uint轉換成string

function uint2str(uint i) internal pure returns (string){

if (i == 0) return "0";

uint j = i;

uint length;

while (j != 0){

length++;

j /= 10;

}

bytes memory bstr = new bytes(length);

uint k = length - 1;

while (i != 0){

bstr[k--] = byte(48 + i % 10);

i /= 10;

}

return string(bstr);

}

//將bytes32轉換成string

function bytes32ToString (bytes32 data)internal pure returns (string) {

bytes memory bytesString = new bytes(32);

for (uint j=0; j<32; j++) {

byte char = byte(bytes32(uint(data) \* 2 \*\* (8 \* j)));

if (char != 0) {

bytesString[j] = char;

}

}

return string(bytesString);

}

//owner 用於添加課程供查詢

function addCourses(string \_courseCode, string courseName, uint studentLimit) public onlyOwner returns (bool){

bytes32 courseCode = stringToBytes32(\_courseCode);

\_turns[turn][coursecounter] = courseCode;

coursecounter = coursecounter.add(1);

\_courses[courseCode] = courseName;

\_studentLimit[turn][courseCode] = studentLimit;

return true;

}

//任何人 用於查詢課程

function searchCourses(string \_courseCode) public view returns (courseInfo courseinfo){

courseinfo.hexCode = stringToBytes32(\_courseCode);

courseinfo.courseCode = \_courseCode;

courseinfo.courseName = \_courses[courseinfo.hexCode];

courseinfo.studentLimit = \_studentLimit[turn][courseinfo.hexCode];

return courseinfo;

}

//回傳選課資訊index從1開始

function bidInformation(uint turnLocal,uint index,address tokenOwner) public view returns (uint,string,string){

require(msg.sender == tokenOwner || msg.sender == owner);

return (\_bidInformation[turnLocal][tokenOwner][index][\_bidInformation2[turnLocal][tokenOwner][index]],\_courses[stringToBytes32(\_bidInformation4[turnLocal][tokenOwner][index])],\_bidInformation4[turnLocal][tokenOwner][index]);

}

//任何人 查詢 自己的token存量

function balanceOf(address tokenOwner) public view returns (uint){

require(msg.sender == tokenOwner || msg.sender == owner);

return \_balances[tokenOwner];

}

//計算msg.sender所有得標課程後存在\_winList

function calculatewinlist (uint turnLocal,address tokenOwner) internal returns(bool){

require(msg.sender == tokenOwner || msg.sender == owner);

uint p = 0;

for (uint i = 0; i < 100; i++){

if(\_bidInformation[turnLocal][tokenOwner][i][\_bidInformation2[turnLocal][tokenOwner][i]] != 0){

for (uint j = 0; j <= \_studentLimit[turnLocal][\_bidInformation2[turnLocal][tokenOwner][i]]; j++){

if(\_winners[turnLocal][\_bidInformation2[turnLocal][tokenOwner][i]][j] == tokenOwner){

\_winList[turnLocal][tokenOwner][p].courseCode = \_bidInformation4[turnLocal][tokenOwner][i] ;

\_winList[turnLocal][tokenOwner][p].bid = \_bidInformation[turnLocal][tokenOwner][i][\_bidInformation2[turnLocal][tokenOwner][i]];

p++;

break;

}

else if(\_winners[turnLocal][\_bidInformation2[turnLocal][tokenOwner][i]][j] == 0){

break;

}

}

}

else{

return true;

}

}

}

//owner 用於發幣給addresses，需要輸入 地址、課程數量、GPA、是否為新生

function transfer(address to) public onlyOwner returns (bool success){

\_balances[msg.sender] = \_balances[msg.sender].sub(38000);

\_balances[to] = \_balances[to].add(38000);

emit Transfer(msg.sender, to, 38000);

return true;

}

//此函數用於比較兩個字串是否相同 因為solidity的機制讓storage的和memory的不能直接比較 (僅僅用於防止學生投標不存在的課程)

function stringsEqual(string storage \_a, string memory \_b) view internal returns (bool) {

bytes storage a = bytes(\_a);

bytes memory b = bytes(\_b);

if (a.length != b.length)

return false;

// @todo unroll this loop

for (uint i = 0; i < a.length; i ++)

if (a[i] != b[i])

return false;

return true;

}

//學生 用於投標 需要輸入 課程代碼、token數量

function bid (string \_courseCode, uint tokens, address tokenowner) internal returns (bool){

bytes32 courseCode = stringToBytes32(\_courseCode);

//require(tokenowner != owner);

require(\_balances[tokenowner] >= tokens);

require(!(stringsEqual(\_courses[courseCode], '')));

\_balances[tokenowner] = \_balances[tokenowner].sub(tokens);

\_balances[owner] = \_balances[owner].add(tokens);

\_bids[courseCode][tokenowner] = \_bids[courseCode][tokenowner].add(tokens);

if (\_haveBidBefore[tokenowner][courseCode] != 1){

\_bidderCounter[courseCode] = \_bidderCounter[courseCode].add(1);

\_addresses[courseCode][\_bidderCounter[courseCode]] = tokenowner;

\_bidInformation2[turn][tokenowner][\_bidInformation3[turn][tokenowner]] = courseCode;

\_bidInformation4[turn][tokenowner][\_bidInformation3[turn][tokenowner]] = \_courseCode;

\_bidInformation[turn][tokenowner][\_bidInformation3[turn][tokenowner]][\_bidInformation2[turn][tokenowner][\_bidInformation3[turn][tokenowner]]] = tokens;

\_bidInformation3[turn][tokenowner] = \_bidInformation3[turn][tokenowner] + 1;

}

else{

for (uint i = 0; i < 100; i++){

if (\_bidInformation2[turn][tokenowner][i] == courseCode){

\_bidInformation[turn][tokenowner][i][\_bidInformation2[turn][tokenowner][i]] = \_bidInformation[turn][tokenowner][i][\_bidInformation2[turn][tokenowner][i]] + tokens;

break;

}

}

}

\_haveBidBefore[tokenowner][courseCode] = 1;

return true;

}

//學生 用於查詢自己在某節課投標的數量

function haveBid (string \_courseCode, address tokenOwner) internal view returns (uint){

bytes32 courseCode = stringToBytes32(\_courseCode);

require(msg.sender == tokenOwner || msg.sender == owner);

return \_bids[courseCode][tokenOwner];

}

//

function StoreHashValue (bytes32 hashvalue) public returns (bool){

hashInfo memory temp;

temp.bidder = msg.sender;

temp.hashValue = hashvalue;

\_hashValue[turn][\_hashValueCounter[turn]] = temp;

\_hashValueSortByAddress[turn][msg.sender][\_hashValueSortByAddressCounter[turn][msg.sender]] = \_hashValueCounter[turn];

\_hashValueSortByAddressCounter[turn][msg.sender] = \_hashValueSortByAddressCounter[turn][msg.sender].add(1);

\_hashValueCounter[turn] = \_hashValueCounter[turn].add(1);

if (\_allAddressChecker[turn][msg.sender] != true){

\_allAddress[turn][\_allAddressCounter[turn]] = msg.sender;

\_allAddressCounter[turn] = \_allAddressCounter[turn].add(1);

\_allAddressChecker[turn][msg.sender] = true;

}

return true;

}

function inputHiddenBid (string courseCode, uint bid1,string String,bytes32 hashvalue, string courseName, uint studentLimit,address student)returns(bool){

require(msg.sender == owner);

addCourses(courseCode,courseName, studentLimit);

buildHash memory temp;

temp.courseCode = courseCode;

temp.bid = bid1;

temp.String = String;

if (verify(courseCode,bid1,String,hashvalue)){

for (uint i = 0; i < \_hashValueSortByAddressCounter[turn][student]; i++){

if ((\_hashValue[turn][\_hashValueSortByAddress[turn][student][i]].hashValue == hashvalue) && (\_verification[turn][student][i] == 0)){

\_HiddenBid[turn][student][\_HiddenBidCounter[turn][student]] = temp;

\_HiddenBidCounter[turn][student] = \_HiddenBidCounter[turn][student].add(1);

\_verification[turn][student][i] = 1;

uint flag = 0;

for (uint j = 0; j < \_hashValueSortByAddressCounter[turn][student]; j++){

if (\_verification[turn][student][j] != 1){

flag = 1;

break;

}

}

if (flag == 0){

uint total = 0;

uint p;

for (p = 0; p < \_HiddenBidCounter[turn][student]; p++){

total = total.add(\_HiddenBid[turn][student][p].bid);

}

if (total <= 38000 && total >= 1){

for (p = 0; p < \_hashValueSortByAddressCounter[turn][student]; p++){

bid(\_HiddenBid[turn][student][p].courseCode,\_HiddenBid[turn][student][p].bid,student);

}

return true;

}else{

return false;

}

}else{

return true;

}

}

}

return false;

}else{

return false;

}

}

function verify (string courseCode, uint bid1, string String,bytes32 hashvalue) internal pure returns (bool){

if ( hashvalue == keccak256(courseCode,',',uint2str(bid1),',',String))

return true;

else

return false;

}

function winlist(uint turnLocal, address tokenOwner,uint index) public view returns (string, uint){

return (\_winList[turnLocal][tokenOwner][index].courseCode,\_winList[turnLocal][tokenOwner][index].bid);

}

//owner 結算函式，輸入courseCode即可對某課程進行結算

//根據投標總人數小於等於和大於總名額分成兩種情況，大於用bubble sort將投標人排序 ，小於就照單全收

function settlement (bytes32 courseCode) internal onlyOwner returns(uint){

uint i;

uint j;

uint max;

address temp;

if (\_bidderCounter[courseCode] > \_studentLimit[turn][courseCode]){

for (i = 0; i < \_studentLimit[turn][courseCode]; i++){

max = 0;

for (j = 1; j <= \_bidderCounter[courseCode]; j ++){

\_haveBidBefore[\_addresses[courseCode][j]][courseCode] = 0;

if (\_bids[courseCode][\_addresses[courseCode][j]] > max){

temp = \_addresses[courseCode][j];

max = \_bids[courseCode][\_addresses[courseCode][j]];

}

}

\_winners[turn][courseCode][i] = temp;

\_winnersBids[turn][courseCode][i] = \_bids[courseCode][temp];

\_bids[courseCode][temp] = 0;

emit Winning\_bidder (turn, courseCode, \_courses[courseCode],i, temp, \_winnersBids[turn][courseCode][i]);

}

\_WinnersTotalNumber[courseCode] = \_studentLimit[turn][courseCode];

for (j = 1; j <= \_bidderCounter[courseCode]; j ++){

\_balances[\_addresses[courseCode][j]] = \_balances[\_addresses[courseCode][j]].add(\_bids[courseCode][\_addresses[courseCode][j]]);

\_bids[courseCode][\_addresses[courseCode][j]] = 0;

}

\_bidderCounter[courseCode] = 0;

return 1;

}

else {

for (i = 0; i < \_bidderCounter[courseCode]; i++){

max = 0;

for (j = 1; j <= \_bidderCounter[courseCode]; j ++){

\_haveBidBefore[\_addresses[courseCode][j]][courseCode] = 0;

if (\_bids[courseCode][\_addresses[courseCode][j]] > max){

temp = \_addresses[courseCode][j];

max = \_bids[courseCode][\_addresses[courseCode][j]];

}

}

\_winners[turn][courseCode][i] = temp;

\_winnersBids[turn][courseCode][i] = \_bids[courseCode][temp];

\_bids[courseCode][temp] = 0;

emit Winning\_bidder (turn, courseCode, \_courses[courseCode],i, temp, \_winnersBids[turn][courseCode][i]);

}

\_WinnersTotalNumber[courseCode] = \_bidderCounter[courseCode];

\_bidderCounter[courseCode] = 0;

}

return 0;

}

//把放在turns裡面的課一節一節塞到settlement裡面，有空的課程就記錄下來，新的投標名額\_studentLimit用原本的\_studentLimit減去\_WinnersTotalNumber

//再用event公告，最後用coursecounter記錄下一輪課的總數 turn++ 開啟下一輪

function ChangeTurn () public onlyOwner returns(bool){

uint i;

uint j;

uint temp;

j = 0;

for (i = 0; i < coursecounter; i++){

if (settlement(\_turns[turn][i]) == 0){

\_ClassAvailable[j] = \_turns[turn][i];

\_turns[turn.add(1)][j] = \_turns[turn][i];

j++;

\_studentLimit[turn.add(1)][\_turns[turn][i]] = \_studentLimit[turn][\_turns[turn][i]].sub(\_WinnersTotalNumber[\_turns[turn][i]]);

temp = turn.add(1);

emit classes(turn, \_turns[turn][i], \_courses[\_turns[turn][i]], \_studentLimit[temp][\_turns[turn][i]]);

}

}

for (i = 0; i < \_allAddressCounter[turn]; i++){

calculatewinlist(turn,\_allAddress[turn][i]);

}

coursecounter = j;

turn++;

}

function GetCourseData(uint turn, address acount, uint index)public view returns (string){

return \_winList[turn][acount][index].courseCode;

}

}