CSIT6000Q (L1)
Fall 2023
Sample Question Paper
25/11/2023
Time Limit: 90 Minutes

Name: _____
Stu ID: _____
Email Address: _____

This exam contains 17 pages (including this cover page) and 5 questions.
Total of points is 119.

Rest of introduction. Rest of introduction. Rest of introduction. Rest of introduction. Rest of introduction. Rest of introduction. Rest of introduction. Rest of introduction.

Grade Table (for teacher use only)

Question	Points	Score
1	45	
2	32	
3	10	
4	18	
5	14	
Total:	119	

Question 1 [45 points]

(In each question, please select only ONE correct answer. If you think there are more than one answer, please select the most reasonable one.)

(a) (3 points) Which of the following is not part of block-chain technology?

A. Ledger.
B. Wallet.
C. Hash.
D. Certified Authority.

(a) **D**

(b) (3 points) what are the pillars of blockchain technology ?

A. Decentralization.
B. Scalability.
C. Immutability.
D. All of the above.

(b) **D**

(c) (3 points) _____ receive verify, gather and execute transactions.

A. Miner nodes.
B. Smart Contracts.
C. Light wallets.
D. Ethereum full node.

(c) **A**

(d) (3 points) Smart Contract characteristics do not include:

A. Alterable
B. Fast and cost-effective
C. A high degree of accuracy
D. Transparency

(d) **A**

(e) (3 points) A popular public-private key implementation known as Rivest-Shamir Adelman (RSA) algorithm is used for the Bitcoin and Ethereum Blockchain.

A. True
B. False

(e) **B**

(f) (3 points) What blockchain is considered Blockchain 1.0, the first blockchain?

A. Bitcoin Cash.
B. Ethereum.

C. Litecoin.
D. Bitcoin.
E. NEO.

(f) **D**

(g) (3 points) What powers the Ethereum Virtual Machine?

A. Gas.
B. Ether.
C. Block Reward.
D. Mining Reward.

(g) **A**

(h) (3 points) Is syntax pragma Correct?

```
pragma solidity >=0.4.20 <0.4.25;
```

A. Correct.
B. Incorrect.

(h) **A**

(i) (3 points) How much does the self-destruct function cost in gas?

A. 5000.
B. 2100.
C. 3000.
D. 2800.

(i) **A**

(j) (3 points) What is the result when the following code compiles and deploy?

```
1.pragma solidity >=0.8.2 <0.9.0;
2.contract A {
3.function getNumber () public returns (uint8){
4.    return 1;
5.    }
6. }
7.contract B is A {
8. function getNumber () public returns (uint8){
9.    return 1;
10.   }
11. }
```

A. Compilation succeeds .
B. An error at line 3 causes compilation to fail.

C. An error at line 8 causes compilation to fail.
D. Compilation succeeds but an exception is thrown at line 8.

(j) **B**

(k) (3 points) What is the result when the following code compiles and deploy?

```
1.pragma solidity >=0.8.2 <0.9.0;
2.
3.contract A {
4. function getNumber () public virtual returns (uint8){
5.    return 1;
6.    }
7. }
8.contract B is A {
9. function getNumber () public returns (uint8){
11.   return 1;
12.   }
13. }
```

A. Compilation succeeds.
B. An error at line 3 causes compilation to fail.
C. An error at line 8 causes compilation to fail.
D. Compilation succeeds but an exception is thrown at line 8.

(k) **C**

(l) (3 points) What is the result when the following code compiles and deploy?

```
1.pragma solidity >=0.8.2 <0.9.0;
2.
3.contract A {
4. function getNumber () public virtual returns (uint8){
5.    return 1;
6.    }
7. }
8.contract B is A {
9. function getNumber () public override returns (uint8){
11.   return 1;
12.   }
13. }
```

A. Compilation succeeds.
B. An error at line 3 causes compilation to fail.
C. An error at line 8 causes compilation to fail.
D. Compilation succeeds but an exception is thrown at line 8.

(l) **A**

(m) (3 points) Which of the following statements is true for contract A?

```
contract A {


}
```

A. public void A() is a valid constructor declaration.
B. A A() public is a valid method declaration.
C. constructor() internal is a valid constructor declaration..
D. None of the above.

(m) **D**

(n) (3 points) Which of the following statements is true for contract A?

```
1.# @version ^0.2.16
2.
3.greet: public(String[100])
4.
5.@external
6.def __init__():
7.    greet = "Hello World"
```

A. Compilation succeeds.
B. An error at line 1 causes compilation to fail.
C. An error at line 2 causes compilation to fail.
D. An error at line 7 causes compilation to fail.

(n) **D**

(o) (3 points) Which of the following are true about Solidity vs Vyper:
A. Vyper has less feature compared to Solidity.
B. Vyper is more secured .
C. Both support Structs.
D. All of the above.

(o) **D**

Question 2 [32 points]

(a) (10 points) Is the following smart contract safe? If not, what type of vulnerability associated it with.

```
1.contract BecTokenSimplified {
2. using SafeMath for uint256;
3.
4.  uint256 public totalSupply;
```

```
5.  mapping(address => uint256) balances;
6.
7.  constructor() {
8.    totalSupply = 7000000000 * (10**18);
9.    balances[msg.sender] = totalSupply;
10.  }
11.
12.  function batchTransfer(address[] receivers, uint256 value) returns
    (bool) {
13.    uint256 amount = receivers.length * value;
14.    require(value > 0 && balances[msg.sender] >= amount);
15.    balances[msg.sender] = balances[msg.sender].sub(amount);
16.    for (uint i = 0; i < receivers.length; i++) {
17.      balances[receivers[i]] = balances[receivers[i]].add(value);
18.    }
19.    return true;
20.  }
21.}
```

Solution:
Integer Overflow at 14 and 15.

(b) (6 points) Is the following smart contract safe? If not, what type of vulnerability associated it with.

```
1 contract MarketPlace {
2 uint public price ;
3 uint public stock ;
4 /.../
5 function updatePrice ( uint _price ){
6 if ( msg . sender == owner )
7 price = _price ;
8 }
9 function buy ( uint quant ) returns ( uint ){
10 if ( msg . value < quant * price || quant > stock )
11 throw ;
12 stock -= quant ;
13 /.../
14 }}
```

Solution: A contract that acts as a market place where users can buy/ sell some tokens. Due to TOD, some order may or may not go through.

(c) (5 points) Is the following smart contract safe? If not, what type of vulnerability

associated it with.

```
1 contract theRun {
2 uint private Last_Payout = 0;
3 uint256 salt = block . timestamp ;
4 function random returns ( uint256 result ){
5 uint256 y = salt * block . number /( salt %5);
6 uint256 seed = block . number /3 + ( salt %300)
7 + Last_Payout +y;
8 // h = the blockhash of the seed - th last block
9 uint256 h = uint256 ( block . blockhash ( seed ));
10 // random number between 1 and 100
11 return uint256 (h % 100) + 1;
12 }}
```

Solution:

contract which depends on block timestamp to send out money.

(d) (5 points) Is the following smart contract safe? If not, what type of vulnerability associated it with.

```
1 contract KingOfTheEtherThrone {
2 struct Monarch {
3 // address of the king .
4 address ethAddr ;
5 string name ;
6 // how much he pays to previous king
7 uint claimPrice ;
8 uint coronationTimestamp ;
9 }
10 Monarch public currentMonarch ;
11 // claim the throne
12 function claimThrone ( string name ) {
13 /.../
14 if ( currentMonarch . ethAddr != wizardAddress )
15 currentMonarch . ethAddr . send ( compensation );
16 /.../
17 // assign the new king
18 currentMonarch = Monarch (
19 msg . sender , name ,
20 valuePaid , block . timestamp );
21 }}
```

Solution: A code snippet of a real contract which does not check the return value after calling other contracts

(e) (6 points) Will the following code compile successful?

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.6 <0.9.0;
constructor and set the value
contract Test {
   // Declaring variable
   string str;

   // Defining a constructor
   constructor(string memory str_in){
      str = str_in;
   }
   // Defining a function to return value of variable 'str'
   function str_out() public view returns(string memory){
      return str;
   }
}
```

Solution:
Yes

Question 3 [10 points]

There is a table 'Ethereum.blocks' which store all the blocks of ethereum. The data column contains only one row of data. Please write an query to fetches the number of blocks for the last 90 days using attribute of the below data table.

Attribute	Data
time	2015-09-27 22:51
number	300248
gas_limit	3141592
gas_used	0
difficulty	7132374666666
total_difficulty	1403681940690798318
size	549
base_fee_per_gas	
hash	0x075bf83f88c422b32346e3fb47f5bfbab4ca0be38d64ffd5f05ecd6602770cf9
parent_hash	0x2691fef8655dd9ed8c9f6ec43da3cda4c69ba85b0688d0cd04ac1d939875b4f5
miner	0x22a0fbf89ad1362d74f626436d8c4fc6dc4f0679
nonce	0x9c63b62f9c0e6a07
date	2015-09-27

Solution:

```
SELECT
    DATE_TRUNC('day',time) AS dt
    , COUNT(*) AS num_blocks
FROM ethereum.blocks
WHERE
    time >= (DATE_TRUNC('day',CURRENT_TIMESTAMP) - '90 days'::INTERVAL)
    AND time <= DATE_TRUNC('day',CURRENT_TIMESTAMP)
GROUP BY 1
ORDER BY 1;
```

Question 4 [18 points]

Write a Contract named Hexagon that extends GeometricObject and implements the Comparable interfaces. Assume all six sides of the hexagon are of equal size. The Hexagon contract is defined as follows:

```
import "@openzeppelin/contracts/utils/math/Math.sol";
import { Clones } from "@openzeppelin/contracts/proxy/Clones.sol";

interface GeometricObject{

    function getPerimeter() external view returns(uint256);

    function getArea() external view returns(uint256);

}
interface Comparable{

    function compareTo(address obj) external returns(int256);

}

interface Cloneable{

    function clone(address objAddress) external returns(GeometricObject);

}
contract Hexagon is GeometricObject, Comparable, Cloneable{

    using Math for uint256;

    uint256 public side;
```

(a) (2 points) Construct a Hexagon with the specified side.

```
constructor(uint256 _side){

    side=_side;
}
```

(b) (2 points) Implement the abstract method getArea in GeometricObject. (Hint: $area = 3 * \sqrt{3} * side * side$)

```
function getArea() public view returns(uint256){

     return 3*Math.sqrt(3)*side*side;

}
```

(c) (2 points) Implement the abstract method getPerimeter in GeometricObject.

```
function getPerimeter() public view returns(uint256){

    return 6*side;

}
```

(d) (6 points) Implement the compareTo method in the Comparable interface. (Hint: compare two Hexagons based on their areas)

```
function compareTo(address obj) public view returns(int256){

     Hexagon hexagon = Hexagon(obj);
    if (side==hexagon.side()) {
        return 0;
    }
     else {
        return int256(hexagon.side() - side);
     }

}
```

(e) (6 points) Implement the clone method.So that X.Clone() will set a new hexagon with the same size as X.

```
function clone(address objAddress) public returns(GeometricObject) {

      Hexagon newHexagonAddress = Hexagon(Clones.clone(objAddress));

      return newHexagonAddress;

}
```

```
}
```

Question 5 [14 points]

(a) Write a following contract using solidity.

```
# Voting with delegation.

# Information about voters
struct Voter:
    # weight is accumulated by delegation
    weight: int128
    # if true, that person already voted (which includes voting by
        delegating)
    voted: bool
    # person delegated to
    delegate: address
    # index of the voted proposal, which is not meaningful unless ‘voted‘
        is True.
    vote: int128

# Users can create proposals
struct Proposal:
    # short name (up to 32 bytes)
    name: bytes32
    # number of accumulated votes
    voteCount: int128

voters: public(map(address, Voter))
proposals: public(map(int128, Proposal))
voterCount: public(int128)
chairperson: public(address)
int128Proposals: public(int128)


@public
@constant
def delegated(addr: address) -> bool:
    return self.voters[addr].delegate != ZERO_ADDRESS


@public
@constant
def directlyVoted(addr: address) -> bool:
    return self.voters[addr].voted and (self.voters[addr].delegate ==
        ZERO_ADDRESS)
```

```
# Setup global variables
@public
def __init__(_proposalNames: bytes32[2]):
    self.chairperson = msg.sender
    self.voterCount = 0
    for i in range(2):
        self.proposals[i] = Proposal({
            name: _proposalNames[i],
            voteCount: 0
        })
        self.int128Proposals += 1

# Give a ‘voter‘ the right to vote on this ballot.
# This may only be called by the ‘chairperson‘.
@public
def giveRightToVote(voter: address):
    # Throws if the sender is not the chairperson.
    assert msg.sender == self.chairperson
    # Throws if the voter has already voted.
    assert not self.voters[voter].voted
    # Throws if the voter's voting weight isn't 0.
    assert self.voters[voter].weight == 0
    self.voters[voter].weight = 1
    self.voterCount += 1

# Used by ‘delegate‘ below, and can be called by anyone.
@public
def forwardWeight(delegate_with_weight_to_forward: address):
    assert self.delegated(delegate_with_weight_to_forward)
    # Throw if there is nothing to do:
    assert self.voters[delegate_with_weight_to_forward].weight > 0

    target: address =
        self.voters[delegate_with_weight_to_forward].delegate
    for i in range(4):
        if self.delegated(target):
            target = self.voters[target].delegate
            # The following effectively detects cycles of length <= 5,
            # in which the delegation is given back to the delegator.
            # This could be done for any int128ber of loops,
            # or even infinitely with a while loop.
            # However, cycles aren't actually problematic for correctness;
            # they just result in spoiled votes.
            # So, in the production version, this should instead be
            # the responsibility of the contract's client, and this
            # check should be removed.
```

```
            assert target != delegate_with_weight_to_forward
        else:
            # Weight will be moved to someone who directly voted or
            # hasn't voted.
            break

    weight_to_forward: int128 =
        self.voters[delegate_with_weight_to_forward].weight
    self.voters[delegate_with_weight_to_forward].weight = 0
    self.voters[target].weight += weight_to_forward

    if self.directlyVoted(target):
        self.proposals[self.voters[target].vote].voteCount +=
            weight_to_forward
        self.voters[target].weight = 0

    # To reiterate: if target is also a delegate, this function will need
    # to be called again, similarly to as above.

# Delegate your vote to the voter `to`.
@public
def delegate(to: address):
    # Throws if the sender has already voted
    assert not self.voters[msg.sender].voted
    # Throws if the sender tries to delegate their vote to themselves or
        to
    # the default address value of
        0x0000000000000000000000000000000000000000
    # (the latter might not be problematic, but I don't want to think
        about it).
    assert to != msg.sender
    assert to != ZERO_ADDRESS

    self.voters[msg.sender].voted = True
    self.voters[msg.sender].delegate = to

    # This call will throw if and only if this delegation would cause a
        loop
        # of length <= 5 that ends up delegating back to the delegator.
    self.forwardWeight(msg.sender)

# Give your vote (including votes delegated to you)
# to proposal `proposals[proposal].name`.
@public
def vote(proposal: int128):
    # can't vote twice
    assert not self.voters[msg.sender].voted
```

```
    # can only vote on legitimate proposals
    assert proposal < self.int128Proposals

    self.voters[msg.sender].vote = proposal
    self.voters[msg.sender].voted = True

    # transfer msg.sender's weight to proposal
    self.proposals[proposal].voteCount += self.voters[msg.sender].weight
    self.voters[msg.sender].weight = 0

# Computes the winning proposal taking all
# previous votes into account.
@public
@constant
def winningProposal() -> int128:
    winning_vote_count: int128 = 0
    winning_proposal: int128 = 0
    for i in range(2):
        if self.proposals[i].voteCount > winning_vote_count:
            winning_vote_count = self.proposals[i].voteCount
            winning_proposal = i
    return winning_proposal

# Calls winningProposal() function to get the index
# of the winner contained in the proposals array and then
# returns the name of the winner
@public
@constant
def winnerName() -> bytes32:
    return self.proposals[self.winningProposal()].name
```

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
/// @title Voting with delegation.
contract Ballot {
    // This declares a new complex type which will
    // be used for variables later.
    // It will represent a single voter.
    struct Voter {
        uint weight; // weight is accumulated by delegation
        bool voted; // if true, that person already voted
        address delegate; // person delegated to
        uint vote;  // index of the voted proposal
    }

    // This is a type for a single proposal.
    struct Proposal {
```

```
    bytes32 name;  // short name (up to 32 bytes)
    uint voteCount; // number of accumulated votes
}

address public chairperson;

// This declares a state variable that
// stores a `Voter` struct for each possible address.
mapping(address => Voter) public voters;

// A dynamically-sized array of `Proposal` structs.
Proposal[] public proposals;

/// Create a new ballot to choose one of `proposalNames`.
constructor(bytes32[] memory proposalNames) {
    chairperson = msg.sender;
    voters[chairperson].weight = 1;

    // For each of the provided proposal names,
    // create a new proposal object and add it
    // to the end of the array.
    for (uint i = 0; i < proposalNames.length; i++) {
        // `Proposal({...})` creates a temporary
        // Proposal object and `proposals.push(...)`
        // appends it to the end of `proposals`.
        proposals.push(Proposal({
            name: proposalNames[i],
            voteCount: 0
        }));
    }
}

// Give `voter` the right to vote on this ballot.
// May only be called by `chairperson`.
function giveRightToVote(address voter) external {
    // If the first argument of `require` evaluates
    // to `false`, execution terminates and all
    // changes to the state and to Ether balances
    // are reverted.
    // This used to consume all gas in old EVM versions, but
    // not anymore.
    // It is often a good idea to use `require` to check if
    // functions are called correctly.
    // As a second argument, you can also provide an
    // explanation about what went wrong.
    require(
        msg.sender == chairperson,
```

```
            "Only chairperson can give right to vote."
        );
        require(
            !voters[voter].voted,
            "The voter already voted."
        );
        require(voters[voter].weight == 0);
        voters[voter].weight = 1;
    }

    /// Delegate your vote to the voter `to`.
    function delegate(address to) external {
        // assigns reference
        Voter storage sender = voters[msg.sender];
        require(sender.weight != 0, "You have no right to vote");
        require(!sender.voted, "You already voted.");

        require(to != msg.sender, "Self-delegation is disallowed.");

        // Forward the delegation as long as
        // `to` also delegated.
        // In general, such loops are very dangerous,
        // because if they run too long, they might
        // need more gas than is available in a block.
        // In this case, the delegation will not be executed,
        // but in other situations, such loops might
        // cause a contract to get "stuck" completely.
        while (voters[to].delegate != address(0)) {
            to = voters[to].delegate;

            // We found a loop in the delegation, not allowed.
            require(to != msg.sender, "Found loop in delegation.");
        }

        Voter storage delegate_ = voters[to];

        // Voters cannot delegate to accounts that cannot vote.
        require(delegate_.weight >= 1);

        // Since `sender` is a reference, this
        // modifies `voters[msg.sender]`.
        sender.voted = true;
        sender.delegate = to;

        if (delegate_.voted) {
            // If the delegate already voted,
            // directly add to the number of votes
```

```
            proposals[delegate_.vote].voteCount += sender.weight;
        } else {
            // If the delegate did not vote yet,
            // add to her weight.
            delegate_.weight += sender.weight;
        }
    }

    /// Give your vote (including votes delegated to you)
    /// to proposal `proposals[proposal].name`.
    function vote(uint proposal) external {
        Voter storage sender = voters[msg.sender];
        require(sender.weight != 0, "Has no right to vote");
        require(!sender.voted, "Already voted.");
        sender.voted = true;
        sender.vote = proposal;

        // If `proposal` is out of the range of the array,
        // this will throw automatically and revert all
        // changes.
        proposals[proposal].voteCount += sender.weight;
    }

    /// @dev Computes the winning proposal taking all
    /// previous votes into account.
    function winningProposal() public view
            returns (uint winningProposal_)
    {
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal_ = p;
            }
        }
    }

    // Calls winningProposal() function to get the index
    // of the winner contained in the proposals array and then
    // returns the name of the winner
    function winnerName() external view
            returns (bytes32 winnerName_)
    {
        winnerName_ = proposals[winningProposal()].name;
    }
}
```