ASSIGNMENT 10

1) **Why does Solidity has memory/storage/calldata data location?**
Ans:
storage : variable is a state variable (store on blockchain).
Unlike a memory variable, any variable defined as storage is written to the blockchain. Since these variables are written to the blockchain they are persistent.
A storage variable can be access from anywhere inside the smart contract and can possibly be also accessed outside the smart contract.
All global variables are by default storage variables.
Any variable defined as storage will incur gas fees since it is written to the blockchain.
memory : variable is in memory and it exists while a function is being called.
Variables stored in memory are not written to the blockchain. To be stored in memory a variable has to be defined inside a function. These variables are temporary and will be destroyed once the function has completed.
Memory variables are mainly used for assisting in performing some type of calculation or operation during the life of the function. Once the function is complete these variables are no longer needed so they are removed from memory.
Since these variables only exist during the execution of the function, they are not accessible from outside anywhere outside the function.
calldata : special data location that contains function arguments.
Calldata is only valid for parameters of external contract functions.
Calldata is a non-modifiable, non-persistent area where function arguments are stored. It behaves mostly like memory.
Any variable defined as calldata cannot be modifiable. In simple terms this means that you cannot change the value of the state of that variable. This is why this data type is assigned to parameters of functions. The data passed into the function is not modified but will be used within the function and a new variable will be returned.

2) **Understand array slice operator [:]**

Ans: Starting from version 0.6.0, Solidity supports array slices. Array slices are handy when you want to reference a contiguous portion of an array but do not want to perform a full copy of that portion. For now, array slices are only supported for calldata arrays.
The expression x[start:end] references a portion of the calldata array x starting at index start and ending just before index end.
Both start and end are optional. If not provided, start defaults to 0 and end defaults to the length of the respective array.
Please note that no copy from calldata to memory is performed.
You can use the slice syntax on variables, but also with msg.data.
Array slices can be especially useful in combination with abi.decode in the fallback function.

3) **Why memory array cannot be dynamic?**

Ans:
Since we are not allowed to change the contract state in a view function we will create this array in memory. In Solidity it is not possible to create dynamic arrays in memory, so we can now make use of the mapping containing the number of expected entries from part one, and use it as the length for our array.

4) **What is the exception for accessing non-existing index of an array in Solidity?**
Ans:
When calling a function via a message call but it does not finish properly (i.e., it runs out of gas, has no matching function, or throws an exception itself), except when a low level operation call, send, delegatecall, callcode or staticcall is used. The low level operations never throw exceptions but indicate failures by returning false more on low level function calls later.

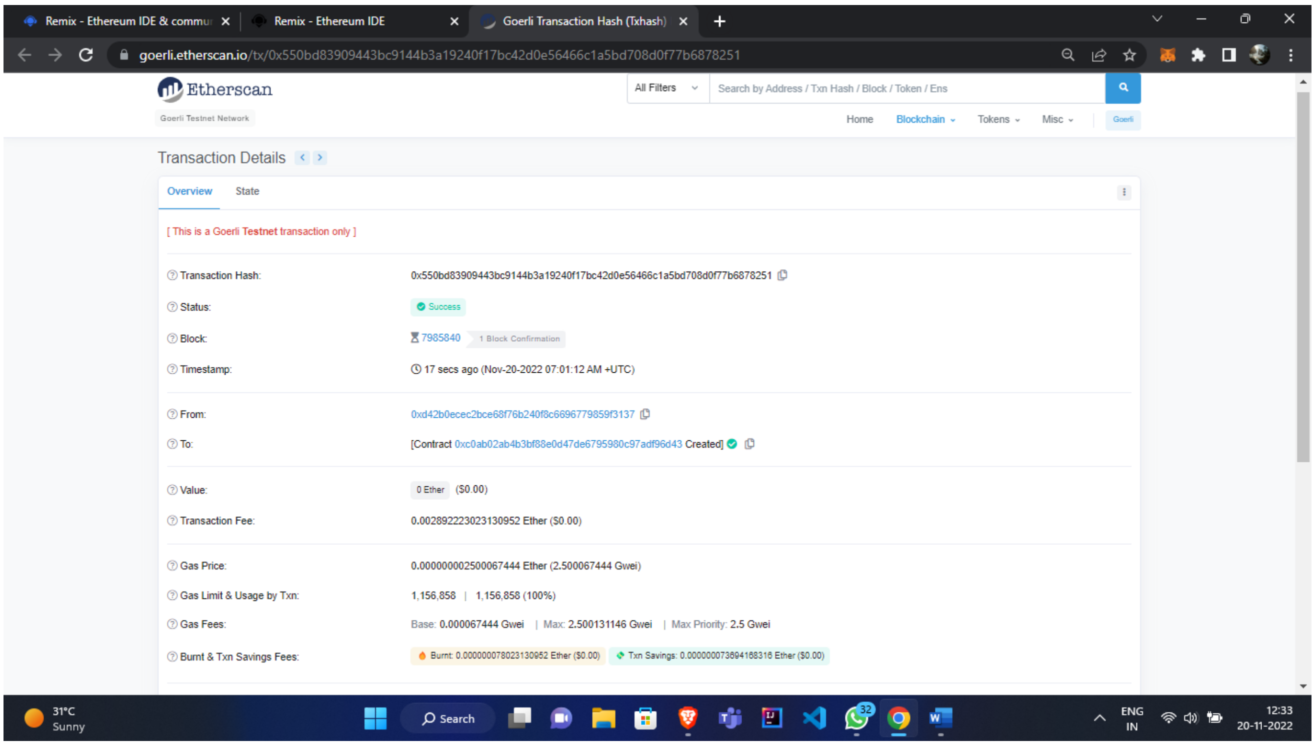
5) **Try to use array slices in either memory or storage arrays and report your findings.**

Ans: Array slices are a view on a contiguous portion of an array. They are written as x[start:end], where start and end

are expressions resulting in a uint256 type (or implicitly convertible to it). The first element of the slice is x[start]
and the last element is x[end - 1].
If start is greater than end or if end is greater than the length of the array, an exception is thrown.
Both start and end are optional: start defaults to 0 and end defaults to the length of the array.
Array slices do not have any members. They are implicitly convertible to arrays of their underlying type and support
index access. Index access is not absolute in the underlying array, but relative to the start of the slice.
Array slices do not have a type name which means no variable can have an array slices as type, they only exist in
intermediate expressions.

6) Deploy Crowdfunding contract on Goerli Testnet, Verify and Publish the contract, Perform Txs and add those to the PDF

Transaction Hash:
0x550bd83909443bc9144b3a19240f17bc42d0e56466c1a5bd708d0f77b6878251

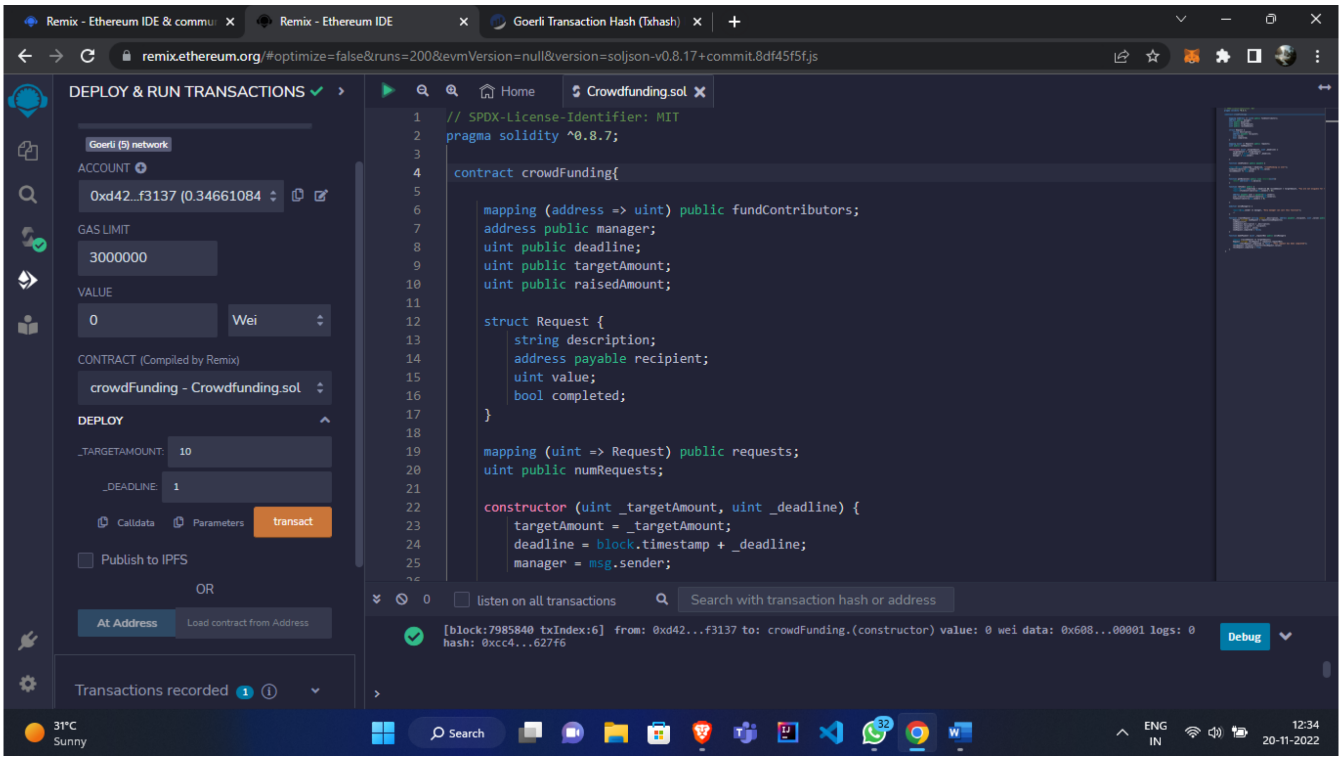


goerli.etherscan.io/tx/0x550bd83909443bc9144b3a19240f17bc42d0e56466c1a5bd708d0f77b6878251

Transaction Details

[This is a Goerli Testnet transaction only]

Transaction Hash:	0x550bd83909443bc9144b3a19240f17bc42d0e56466c1a5bd708d0f77b6878251
Status:	Success
Block:	7985840 1 Block Confirmation
Timestamp:	17 secs ago (Nov-20-2022 07:01:12 AM +UTC)
From:	0xd42b0ecec2bce68f76b240f8c6696779859f3137
To:	[Contract 0xc0ab02ab4b3bf88e0d47de6795980c97adf96d43 Created]
Value:	0 Ether ($0.00)
Transaction Fee:	0.002892223023130952 Ether ($0.00)
Gas Price:	0.000000002500067444 Ether (2.500067444 Gwei)
Gas Limit & Usage by Txn:	1,156,858 | 1,156,858 (100%)
Gas Fees:	Base: 0.000067444 Gwei | Max: 2.500131146 Gwei | Max Priority: 2.5 Gwei
Burnt & Txn Savings Fees:	Burnt: 0.000000078023130952 Ether ($0.00) Txn Savings: 0.000000073894168316 Ether ($0.00)



remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.17+commit.8df45f5f.js

DEPLOY & RUN TRANSACTIONS

Goerli (5) network
ACCOUNT: 0xd42...f3137 (0.34661084
GAS LIMIT: 3000000
VALUE: 0 Wei
CONTRACT (Compiled by Remix): crowdFunding - Crowdfunding.sol
DEPLOY
_TARGETAMOUNT: 10
_DEADLINE: 1

Crowdfunding.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

 contract crowdFunding{

     mapping (address => uint) public fundContributors;
     address public manager;
     uint public deadline;
     uint public targetAmount;
     uint public raisedAmount;

     struct Request {
         string description;
         address payable recipient;
         uint value;
         bool completed;
     }

     mapping (uint => Request) public requests;
     uint public numRequests;

     constructor (uint _targetAmount, uint _deadline) {
         targetAmount = _targetAmount;
         deadline = block.timestamp + _deadline;
         manager = msg.sender;
```

```
[block:7985840 txIndex:6] from: 0xd42...f3137 to: crowdFunding.(constructor) value: 0 wei data: 0x608...00001 logs: 0
hash: 0xcc4...627f6
```