BDL CW3 REPORT

Part 1

I used the following state variables:

- *owner* (public)
- *_totalSupply* – defaults at 0. internal with getter *totalSupply()*.
- *name* – internal with getter *getName()*.
- *symbol* – internal with getter *getSymbol()*.
- *price* – internal with getter *getPrice()*.
- *balances* – internal with getter for individual balances *balanceOf()*.

Minting tokens:

Minting (creating) tokens and giving them to a certain account can be done only by the owner through the *mint* function. I used an *onlyOwner* modifier to ensure that only the owner can call this function.

The process of minting is simple:

- Add the tokens into the token balance of the receiving account (both specified by owner)
- Update the totalSupply variable to realise this addition of tokens in circulation.

Transferring tokens:

Transferring tokens from account to account can be done by anyone as long as the caller has enough tokens in their balance, that is, their balance must be more than or equal to the amount they wish to transfer. This is checked through a *require* statement.

Here is the process of transferring tokens (The important steps of the *transfer* function):

- Check token balance of msg.sender satisfies with requirements outlined above.
- Take tokens out of msg.sender's token balance.
- Add tokens into the token balance of the recipient specified by msg.sender.

Selling tokens:

A specific number of tokens can be sold by any token holder on two conditions: they must have at least as many tokens as they specified to sell in their token balance, and the contracts balance must be at least the amount of wei due to the seller (i.e., (contracts balance ≥ price * #OfTokensToSell) must hold).

Here is the process of Selling tokens (the important steps of the *Sell* function):

- Check first requirement explained above. **(checks)**
- Check second requirement explained above. **(checks)**
- Take the tokens out of msg.sender's token balance. **(effects)**
- Update the totalSupply variable to realise this reduction of tokens in circulation. **(effects)**
- Make the ether transaction (wei sent from contract to seller) **(interaction)**

Users accessing information:

Users are able to access a token balance via the *balanceOf* function, with the argument of the address of the account they wish to view the token balance of.

Users are able to access the total supply, name, symbol, and price of the token via the getters *totalSupply(), getName(), getSymbol(),* and *getPrice()* respectively.

Users are also able to view the owners address through a call to the variable since it is a public state variable.

Hazard - Re-entrancy:

In the *sell* function, there is a *call* statement that is used to transfer wei into the sellers account. This opens up a potential re-entrancy attack where this transfer triggers another contracts fallback function, which could again call a function in the contract. The consequences could be devastating; it could be that the contracts entire balance is emptied unfairly.

Solution - checks-effects-interaction pattern:

Using the checks effects interaction pattern, attackers mounting a re-entrancy attack would have no unfair gain. This is because the internal state (the attackers token balance) is changed before any wei is transferred out of the contracts account, and so any looped call to the *sell* function would first check that the seller has enough tokens, then it would take their tokens away, then it would make the transfer. Now, an attempt of re-entrancy where the attacker calls the *sell* function repeatedly would just end up making the attacker sell all (or most of) their tokens (fairly). Using *transfer* instead of *call* to send wei would be another fix to re-entrancy, because *transfer* only allows 2300 gas to be used whereas there is no limit on *call*. The reason I don't use this is if the price of ether increases and 2300 gas becomes insufficient to complete some required actions done by another contract, then they would never be able to *sell* their tokens until the price drops again so that 2300 gas is enough to complete actions.

Things to note – cost affective techniques:

- My functionality allows for users to transfer tokens with themselves. **Since this has no security implications**, I decided not to constrain this with checks, **to allow for fewer lines of code and ultimately fewer gas costs at deployment and usage**.
- Similarly, users can transfer tokens to accounts that do not exist yet (i.e., addresses that isn't in the *balances* mapping yet). For the same reason as above, I allow this. What happens is this new address simply gets added to the mapping.
- Similarly, the only restriction on the number of tokens that one can transfer is that the transferrer must have the number of tokens they wish to transfer. Notice how a user can transfer 0 tokens. I choose to allow this for the same reason as mentioned above.
- Similarly, I allow for 0 tokens to be sold, causing a 0 value transfer, which is known to be harmless, and only causes the seller to pay the transaction costs. (Note, this makes the usage of the *customSend* function in the provided library fail on the value check if 0 tokens is being sold).

I am not 100% sure on my statement above that it "has no security implications". Maybe it can be that a user can spam transfers of 0 tokens, or transfers to themselves, potentially causing a DOS due to overwhelming the contract with lots of function calls. I am not sure if this scenario is possible, i.e., I am not sure how possible it is to overwhelm a contract in this way. One can argue that the attacker in this scenario would be paying the transaction fees anyway, so it doesn't really matter. However, if such an attack is possible, an easy fix would be to add the following:

- Require the *to* argument in the *transfer* function to be different from *msg.sender.*
- Require the *value* argument in the *transfer* function to be greater than 0.
- Require the *value* argument in the *sell* function to be greater than 0.

Cost of deployment:

Transaction fee : 0.002837967676677375 Ether (gas price * gas used)
At gas price : 0.000000002505433955 Ether
Amount of gas used : 1,132,725
My address : 0x0c15891cc34414d422028e7cab662aceb3961a3a
Contract address : 0xbf1b568fb2a30765f8d99ac03f1b77ea7da5d1c2

Cost of sending ether 0.2 ether to contract (via *receive* function; no calldata):

Transaction fee : 0.000052659925575225 Ether
Gas price : 0.000000002501065095 Ether
Gas used : 21,055

Cost of minting tokens (via the *mint* function):

Argument *to* : 0x0c15891cc34414d422028e7cab662aceb3961a3a
Argument *value* : 10
Transaction fee: 0.00017736770258042 Ether
Gas price : 0.000000002500249543 Ether
Gas used : 70,940

Cost of transferring tokens (via the *transfer* function):

Argument *to* : 0x95A1Eb250375D36834Da1d66017938Cc091CCb7f
Argument *value* : 5
Transaction fee: 0.00013087639800675 Ether
Gas price : 0.000000002500026705 Ether
Gas used : 52,350

Cost of Selling tokens (via the *sell* function):

Argument *value* : 2
Transaction fee : 0.000110113480309565 Ether
Gas price : 0.000000002500022257 Ether
Gas used : 44,045

Transaction history of deployment and interaction:

Refer to appendix 1.

Code of my contract:

Refer to appendix 2.

Owner assumptions:

In order for this implementation to be secure, we must assume that the owner is trustworthy. Users who decide to invest in these tokens should be able to sell them at any point. This requires the contract to have enough ether in it such that it can handle all the sell requests. So, we require the owner to top up the contracts balance to deal with this. Otherwise, the owner would be able to make all tokens unsellable through insufficient contract balance and therefore users would lose their tokens worth. Note, there can be a check in the mint function that makes sure the contract has enough ether to be able to handle every sell request at the time of the minting, but this doesn't stop the owner from then spending wei from the contracts balance later on, making the check pointless, so I left it out to save gas.

It is also key to note that the owner could call the *close* function
at any point in time causing all the users to lose their tokens completely (with no compensation). So, we need to assume that the owner acts in a reasonable manner for this contract to be secure. We could have made sure to pay all token holders as part of the *close* function however this would've required some checks with the contracts balance as well as multiple (potentially a lot) of transfers (each with their own checks). To reduce any security risks as well as to save gas (for the owner), I have decided to keep the *close* function simple (paying out nobody), and putting all the trust in the owner. The owner is also allowed to create tokens (for free) and give them to anyone (including themselves). The owner must act faithfully.

Part 2

To link the deployed library to my contract here is what I had to do:

1. Add the line "import "./customLib.sol"" to tell remix that we are using a library.
2. Add the line "using customLib for uint" making it so that we can use the customSend function as a function for uint, in this case, the value parameter.
3. Call the function like so: "(value * price).customSend(msg.sender)"

Once we have done this, we need to tell remix that the library customLib is already deployed at the address 0x9DA4c8B1918BA29eBA145Ee3616BCDFcFAA2FC51. We do this by editing the metadata (json) file of my token contract. There are placeholders for addresses for libraries that are imported into the file, we can simply add the library there under the goerli network. This will tell remix to use the library which is deployed at that address. I also had to update the autoDeployLib to false so that remix does not deploy the library (which is already deployed).

I avoided the use of delegateCall because if used, the address associated with the owner variable of my contract would get changed because it is also assigned in the library; and all the code in the library is run in my contacts context.

Cost of Selling tokens (via the *sell* function):

Argument *value* : 2
Transaction fee : 0.000120805001159728 Ether
Gas price : 0.000000002500000024 Ether
Gas used : 58,138
Transaction Hash: 0x1b20b686581251c93238f9cb924fcbf745833c4c7a72fbf098cc23e75a1f0517

Note, this costs more gas than selling from part 1.

For the code of the contract, refer to appendix 3.

Part 3

The owner of the contract will have a public key *pk*.
This key will be publicly known by anyone.
This is the key that recipients of tokens will use to encrypt their identification document.

The owner of the contract will also have a private key *vk*.
This is the only key that can decrypt the (encrypted) identification document. This *pk* will only be known to the owner of the contract, therefore only the owner will be able to gain information about the identification documents, so long as this *pk* is kept safe and **off-chain**.
This key should be kept off chain because, as we know, everything on the chain is public and viewable (even when declared as a private variable).
For this reason, users should also keep their identification documents off-chain, and only put their encrypted document on-chain. So users will need to do some offline pre-processing.

Because the *mint* function is executed by the owner, we would need an additional function (called *sendID*) that allows users to provide encrypted identification documents, which will be stored in a mapping.

General steps of the process:
(Recipient of new minted tokens = **R**)
(Public key = **pk**)
(Private key = **vk**)

- **R** encrypts their ID using **pk**. Resulting in encrypted ID **eid** (done offline/off-chain).
- **R** calls *sendID* function, passing in **eid** as an argument.
- Owner checks manually that ID is valid (reasoning explained below).
- Owner can call *mint* function as usual.

eids – new state variable:
We require a new mapping state variable called *eids*. This will map addresses to encrypted identification document (string) of the corresponding account holder of the address.

sendID function:
This function will take in a string as an argument. This string should be the encrypted identification of the sender.
This function will add an entry into the *eids* state variable, with the address being the caller of the function and the string being the argument passed in.
Of course, it can be that users send something unacceptable, that isn't their id, which I talk about in the next paragraph.

Mint function:
This can be identical.
We could add a check that there should exist an entry in the *eids* mapping containing the address of the recipient of the minted tokens.
But because users can provide fake id's and make it seem like there is a real id associated with their address, I would require the owner of the contract to manually check that the ID exists and to check (with the private key) that this ID is valid.

Appendix 1 - Transaction history of deployment and interaction:


```
{
  "accounts": {
    "account{0}": "0x0c15891cC34414d422028e7CAb662aCeb3961a3A"
  },
  "linkReferences": {},
  "transactions": [
    {
      "timestamp": 1670527262369,
      "record": {
        "value": "0",
        "inputs": "()",
        "parameters": [],
        "name": "",
        "type": "constructor",
        "abi": "0x89a7b192b9711b140b522ee78bf5edf67017589ce34b7e1c826e910c4731065b",
        "contractName": "B160966_part1",
        "bytecode":
"60806040523480156200001157600080fd5b50336000806101000a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff160217905550604051806040016040528060088152602001 7f435733546f6b656e0000000000000000000000000000000000000000000000008152506002908051906020019062000009f9291906200012f565b5060405180604001604052806004815260 20017f435733540000000000000000000000000000000000000000000000000000000081525060039080519060200190620000ed9291906200012f565b5061025860046006101000a8154816fffffffffffffffffffffffffffffffff02191690836fffffffffffffffffffffffffffffffff160217905550620002445565b8280546200013d90620001df565b906000526020600209060 1f0160209004810192826200016157600085556200001ad565b82601f106200017c57805160ff1916838001178555620001ad565b828001600101855582156200001ad579182015b828111156200001ac578251825591602001919060010190620001f565b5b5090506200001bc9190620001c0565b5090565b5b808211156200001db576000816000905550600101620001c1565b5090565b60006002820490506001821680620001f857607f821691505b602082108114156200020f576200020e62000215565b5b50919050565b7f4e487b7100000000000000000000000000000000000000000000000000000000600052602260045260246000fd5b6111d78062000254600039600 0f3fe608060405260043610610095576000 3560e01c806370a082311161005957806370a08231146101735780638da5cb5b146101b057806398d5fdca146101db578063a9059cbb14610206578063e4849b32146102435761009c565b806315070401146100e57806317d7de7c146100c957806318160ddd146100f457806340c10f191461011f57806343d726d61461015c5761009c565b3661009c57005b005b3480156100aa57600080fd5b506100b3610280565b6040516100c09190610d05565b60405180910390f35b3480156100d557600080fd5b506100de610312565b6040516100eb9190610d05565b60405180910390f35b34801561010057600080fd5b506101096103a4565b6040516101169190610de2565b60405180910390f35b34801561012b57600080fd5b506101466004803603810190610141919061 0b06565b6103ae565b60405161015391906 10cea565b60405180910390f35b34801561016857600080fd5b5061017610506565b005b34801561017f57600080fd5b5061019a600480360381019061019591906 10ad9565b6105cd565b6040516101a79190610de2565b60405180910390f35b3480156101bc57600080fd5b506101c5610616565b6040516101d29190610ccf565b60405180910390f35b3480156101e757600080fd5b506101f061061063a565b6040516101fd9190610dc7565b60405180910390f35b34801561021257600080fd5b5061022d6004803603810190610228919061 0b06565b610660565b60405161023a9190610ce a565b60405180910390f35b34801561024f57600080fd5b5061026a600480360381019061026591906
```

10b46565b6107ff565b6040516102779190610cea565b60405180910390f35b606060038054610280f9
0610fb1565b80601f0160208091040260200160405190810160405280929190818152602001828054
6102bb90610fb1565b8015610308578060 1f106102dd576101008083540402835291602001916103 0
8565b8201919060005260206000020905b8154815290600101906020018083116102eb57829003601f
168201915b5050505050905090565b606060028054610321906 10fb1565b80601f016020809104026
0200160405190810160405280929190818152602001828054610 34d90610fb1565b801561039a5780
601f1061036f576101008083540402835291602001916103 9a565b8201919060005260206000020905
b8154815290600101906020018083116103 7d57829003601f168201915b5050505050905090565b60
0060015490509056 5b6000806000905490610100 0a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffff
ffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff161461043f576040517f08c379a00000
0000000000000000000000000000000000000000000000000000000000008152600401610436 90610d87565b
60405180910390fd5b81600560008573ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffff
fffff16815260020019081526020016000206000828254610 48e9190610e24565b92505081905550816
0016000828254610 4a79190610e24565b92505081905550827 3ffffffffffffffffffffffffffffffffffffffff167f0f
6798a560793a54c3bcfe86a93cde1e73087d944c0ea20544137d4121396885836040516104f4919061
0de2565b60405180910390a2600190509291505056 5b6000805490610100 0a900473ffffffffffffffffffff
ffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff16146105945
76040517f08c379a0000000000000000000000000000000000000000000000000000000000008152600 4
0161058b90610d87565b60405180910390fd5b6000805490610100 0a900473ffffffffffffffffffffffffffffffff
ffffffff1673ffffffffffffffffffffffffffffffffffffffff16ff5b6000600560008373ffffffffffffffffffffffffffffffffffffffff16
73ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054905091905056 5b60008054
906101000a900473ffffffffffffffffffffffffffffffffffffffff1681565b6000600460009054906101000a90046fff
ffffffffffffffffffffffffff16905090565b6000816005600033 73ffffffffffffffffffffffffffffffffffffffff1673ffffffff
ffffffffffffffffffffffffff16815260200190815260200160002054101561088357 6040517f08c379a00
0000000000000000000000000000000000000000000000000000000000008152600 4016106db90610d675
65b60405180910390fd5b81600560003373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffff
fffffffff16815260200190815260200160002060008282546107339190610ed4565b92505081905550
81600560008573ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190
8152602001600020600082825461078991906 10e24565b92505081905550827 3ffffffffffffffffffffffffffffffff
ffffffffff163373ffffffffffffffffffffffffffffffffffffffff167fddf252ad1be2c89b69c2b068fc378daa952ba7f16
3c4a11628f55a4df523b3ef846040516107ed9190610de2565b60405180910390a3600190509291505
0565b6000816005600033 73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815
260200190815260200160002054101561088357 6040517f08c379a0000000000000000000000000000
0000000000000000000000000000000008152600 40161087a90610da7565b60405180910390fd5b60046
0009054906101000a90046ffffffffffffffffffffffffffffffffffffffff166ffffffffffffffffffffffffffffffff16826108bf919061
0e7a565b4710156109015 76040517f08c379a0000000000000000000000000000000000000000000000000000
0000000000008152600 4016108f890610d47565b60405180910390fd5b81600560003373ffffffffffffffff
ffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060008282
2546109509190610ed4565b92505081905550816001600082825461096 99190610ed4565b9250508
190555060003373ffffffffffffffffffffffffffffffffffffffff166004600090549061010 0a90046ffffffffffffffffffff
ffffffffff166ffffffffffffffffffffffffffffffff16846109c59190610e7a565b6040516109d190610cba565b600
6040518083038185875af1925050503d8060008114610a0e57604051915060 1f19603f3d011682016
040523d82523d6000602084013e610a13565b606091505b5050905080610a57576040517f08c379a0
0000000000000000000000000000000000000000000000000000000000008152600401610a4e90610d27
565b60405180910390fd5b3373ffffffffffffffffffffffffffffffffffffffff167f5e5e995ce3133561afceaa51a9a1
54d5db228cd7525d34df5185582c18d3df0984604051610a9d9190610de2565b60405180910390a26
0019150509190505 65b600081359050610abe81611173565b92915050565b600081359050610ad38
161118a565b92915050565b6000602082840312156 10aef57610aee611041565b5b6000610afd8482

8501610aaf565b9150509291505056 5b6000806040838503121561 0b1d57610b1c611041565b5b600
0610b2b85828601610aaf565b9250506020610b3c85828601610ac4565b9150509250929050565b60
0060208284031215610b5c57610b5b611041565b5b6000610b6a84828501610ac4565b91505092915
050565b610b7c81610f1a565b82525050565b610b8b81610f2c565b82525050565b6000610b9c8261
0dfd565b610ba68185610e13565b9350610bb6818560208601610f7e565b610bbf81611046565b840
191505092915050565b6000610bd7602683610e13565b9150610be282611057565b6040820190509
19050565b6000610bfa601f83610e13565b9150610c05826110a6565b602082019050919050565b60
00610c1d601283610e13565b9150610c28826110cf565b602082019050919050565b6000610c40601
583610e13565b9150610c4b826110f8565b602082019050919050565b6000610c63603083610e1356
5b9150610c6e82611121565b604082019050919050565b6000610c86600083610e08565b9150610c9
182611170565b600082019050919050565b610ca581610f38565b82525050565b610cb481610f7456
5b82525050565b6000610cc582610c79565b9150819050919050565b600060208201905061 0ce4600
0830184610b73565b92915050565b600060208201905061 0cff600830184610b82565b9291505056
5b600060208201905081810360008301526 10d1f8184610b91565b905092915050565b60006020820
190508181036000830152610d4081610bca565b9050919050565b600060208201905081810360008 3
0152610d6081610bed565b9050919050565b600060208201905081810360008301526 10d8081610c
10565b9050919050565b600060208201905081810360008301526 10da081610c33565b90509190505
65b600060208201905081810360008301526 10dc081610c56565b9050919050565b60006020820190
50610ddc6000830184610c9c565b92915050565b600060208201905061 0df76000830184610cab565
b92915050565b600081519050919050565b600081905092915050565b6000828252602082019050 9
2915050565b6000610e2f82610f74565b9150610e3a83610f74565b9250827fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff03821115610e6f57610e6e610fe3565b5b828201905092915050565b6000610e8582610f74565b9150610e9083610f74565b9250817fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff048311821515161561 0ec957610ec8610fe3565b5b828202905092915050565b6000610edf8261 0f74565b9150610eea83610f74565b925082821015610efd57610efc610fe3565b5b828203905092915 050565b6000610f1382610f54565b9050919050565b6000610f2582610f54565b9050919050565b600 0811515905091905056 5b60006fffffffffffffffffffffffffffffffff8216905091905056 5b600073ffffffffffffffffffffffffffffffffffffffff8216905091905056 5b600081905091905056 5b60005b83811015610f9c5780820151 818401526020810190506 10f81565b83811115610fab576000848401525b50505050565b600060028 204905060001821680610fc957607f821691505b60208210811415610fdd57610fdc611012565b5b509 19050565b7f4e487b7100000000000000000000000000000000000000000000000000000000600052 601160045260246000fd5b7f4e487b7100000000000000000000000000000000000000000000000000000000 0000000600052602260045260246000fd5b600080fd5b6000601f19601f8301169050919050565b7f4 661696c656420746f2073656e642065746865722069 6e746f20796f7572206160008201527f63636f75 6e7400000000000000000000000000000000000000000000000000000000602082015250565b7f436f6e7 472616374206861732069 6e737566669636 9656e742066756e647300600082015250565b7f496e73 756666696369656e742046756e647300000000000000000000000000600082015250565b7f596f7 520617265206e6f7420746865206f776e657200000000000000000000600082015250565b7f596f7 520646f206e6f74206861766520746861742 06d616e7920746f6b656e736000820152 7f20696e20796f757220626 16c616e6365000000000000000000000000000000602082015250565b50565b61117c 81610f08565b81146111875760008 0fd5b50565b6111938 1610f74565b811461119e57600080fd5b50 56fea2646970667358221220 1ae5ae51e64e448da363426a4ac0a1a485e7baab80ccc41aa979edae5f8 ead6064736f6c63430008070033",
      "linkReferences": {},
      "from": "account{0}"
    }
  },
  {

```
  "timestamp": 1670528122976,
  "record": {
    "value": "200000000000000000",
    "inputs": "()",
    "parameters": [],
    "type": "receive",
    "to": "created{1670527262369}",
    "abi": "0x89a7b192b9711b140b522ee78bf5edf67017589ce34b7e1c826e910c4731065b",
    "from": "account{0}"
  }
 },
 {
  "timestamp": 1670528824451,
  "record": {
    "value": "0",
    "inputs": "(address,uint256)",
    "parameters": [
      "0x0c15891cc34414d422028e7cab662aceb3961a3a",
      "10"
    ],
    "name": "mint",
    "type": "function",
    "to": "created{1670527262369}",
    "abi": "0x89a7b192b9711b140b522ee78bf5edf67017589ce34b7e1c826e910c4731065b",
    "from": "account{0}"
  }
 },
 {
  "timestamp": 1670530319228,
  "record": {
    "value": "0",
    "inputs": "(address,uint256)",
    "parameters": [
      "0x95A1Eb250375D36834Da1d66017938Cc091CCb7f",
      "5"
    ],
    "name": "transfer",
    "type": "function",
    "to": "created{1670527262369}",
    "abi": "0x89a7b192b9711b140b522ee78bf5edf67017589ce34b7e1c826e910c4731065b",
    "from": "account{0}"
  }
 },
 {
  "timestamp": 1670530493809,
  "record": {
    "value": "0",
    "inputs": "(uint256)",
```

```
      "parameters": [
        "2"
      ],
      "name": "sell",
      "type": "function",
      "to": "created{1670527262369}",
      "abi": "0x89a7b192b9711b140b522ee78bf5edf67017589ce34b7e1c826e910c4731065b",
      "from": "account{0}"
    }
  }
],
"abis": {
  "0x89a7b192b9711b140b522ee78bf5edf67017589ce34b7e1c826e910c4731065b": [
    {
      "inputs": [],
      "name": "close",
      "outputs": [],
      "stateMutability": "nonpayable",
      "type": "function"
    },
    {
      "inputs": [
        {
          "internalType": "address",
          "name": "to",
          "type": "address"
        },
        {
          "internalType": "uint256",
          "name": "value",
          "type": "uint256"
        }
      ],
      "name": "mint",
      "outputs": [
        {
          "internalType": "bool",
          "name": "",
          "type": "bool"
        }
      ],
      "stateMutability": "nonpayable",
      "type": "function"
    },
    {
      "inputs": [],
      "stateMutability": "nonpayable",
      "type": "constructor"
```

```
},
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "internalType": "address",
      "name": "to",
      "type": "address"
    },
    {
      "indexed": false,
      "internalType": "uint256",
      "name": "value",
      "type": "uint256"
    }
  ],
  "name": "Mint",
  "type": "event"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "value",
      "type": "uint256"
    }
  ],
  "name": "sell",
  "outputs": [
    {
      "internalType": "bool",
      "name": "",
      "type": "bool"
    }
  ],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "internalType": "address",
      "name": "from",
      "type": "address"
    },
```

```
    {
      "indexed": false,
      "internalType": "uint256",
      "name": "value",
      "type": "uint256"
    }
  ],
  "name": "Sell",
  "type": "event"
},
{
  "inputs": [
    {
      "internalType": "address",
      "name": "to",
      "type": "address"
    },
    {
      "internalType": "uint256",
      "name": "value",
      "type": "uint256"
    }
  ],
  "name": "transfer",
  "outputs": [
    {
      "internalType": "bool",
      "name": "success",
      "type": "bool"
    }
  ],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "internalType": "address",
      "name": "from",
      "type": "address"
    },
    {
      "indexed": true,
      "internalType": "address",
      "name": "to",
      "type": "address"
```

```
    },
    {
      "indexed": false,
      "internalType": "uint256",
      "name": "value",
      "type": "uint256"
    }
  ],
  "name": "Transfer",
  "type": "event"
},
{
  "stateMutability": "payable",
  "type": "fallback"
},
{
  "stateMutability": "payable",
  "type": "receive"
},
{
  "inputs": [
    {
      "internalType": "address",
      "name": "addr",
      "type": "address"
    }
  ],
  "name": "balanceOf",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "getName",
  "outputs": [
    {
      "internalType": "string",
      "name": "",
      "type": "string"
    }
  ],
```

```
  "stateMutability": "view",
  "type": "function"
 },
 {
  "inputs": [],
  "name": "getPrice",
  "outputs": [
   {
    "internalType": "uint128",
    "name": "",
    "type": "uint128"
   }
  ],
  "stateMutability": "view",
  "type": "function"
 },
 {
  "inputs": [],
  "name": "getSymbol",
  "outputs": [
   {
    "internalType": "string",
    "name": "",
    "type": "string"
   }
  ],
  "stateMutability": "view",
  "type": "function"
 },
 {
  "inputs": [],
  "name": "owner",
  "outputs": [
   {
    "internalType": "address payable",
    "name": "",
    "type": "address"
   }
  ],
  "stateMutability": "view",
  "type": "function"
 },
 {
  "inputs": [],
  "name": "totalSupply",
  "outputs": [
   {
    "internalType": "uint256",
```

```
          "name": "",
          "type": "uint256"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    }
  ]
 }
}
```

Appendix 2 – Code of my contract:

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >= 0.7.0 <0.9.0;

contract B160966_part1{

    address payable public owner;
    uint256 internal _totalSupply; //Defaults to 0
    string internal name;
    string internal symbol;
    uint128 internal price;
    mapping(address => uint256) internal balances;

    constructor() {
        owner = payable(msg.sender);
        name = "CW3Token";
        symbol = "CW3T";
        price = 600 wei;
    }

    event Transfer(
        address indexed from,
        address indexed to,
        uint256 value
    );

    event Mint(
        address indexed to,
        uint256 value
    );

    event Sell(
        address indexed from,
        uint256 value
    );

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address addr) public view returns (uint256) {
        return balances[addr];
    }

    function getName() public view returns (string memory){
        return name;
```

```
}

function getSymbol() public view returns (string memory){
    return symbol;
}

function getPrice() public view returns (uint128){
    return price;
}

function transfer(address to, uint256 value) public returns (bool success) {
    require(balances[msg.sender] >= value, "Insufficient Funds");

    balances[msg.sender] -= value;
    balances[to] += value;

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

    return true;
}

modifier onlyOwner(){
    require (msg.sender == owner, "You are not the owner");
    _;
}

function mint(address to, uint256 value) public onlyOwner returns (bool){
    balances[to] += value;
    _totalSupply += value;

    emit Mint(to, value);

    return true;
}

function sell(uint256 value) public returns (bool) {
    require (balances[msg.sender] >= value, "You do not have that many tokens in your balance");
    require (address(this).balance >= value * price, "Contract has insufficient funds");

    balances[msg.sender] -= value;
    _totalSupply -= value;

    (bool sent, ) = msg.sender.call{value: (value * price)}("");
    require(sent, "Failed to send ether into your account");

    emit Sell(msg.sender, value);

    return true;
```

```
    }

    function close() public onlyOwner{
        selfdestruct(owner); //This will also transfer the contracts balance to the owner.
    }

    fallback() external payable {}

    receive() external payable {}
}
```

Appendix 3 – Code of my contract:

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >= 0.7.0 <0.9.0;

import "./customLib.sol";

contract Token{
    using customLib for uint;

    address payable public owner;
    uint256 internal _totalSupply; //Defaults to 0
    string internal name;
    string internal symbol;
    uint128 internal price;
    mapping(address => uint256) internal balances;

    constructor() {
        owner = payable(msg.sender);
        name = "CW3Token";
        symbol = "CW3T";
        price = 600 wei;
    }

    event Transfer(
        address indexed from,
        address indexed to,
        uint256 value
    );

    event Mint(
        address indexed to,
        uint256 value
    );

    event Sell(
        address indexed from,
        uint256 value
    );

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address addr) public view returns (uint256) {
        return balances[addr];
    }
```

```
function getName() public view returns (string memory){
    return name;
}

function getSymbol() public view returns (string memory){
    return symbol;
}

function getPrice() public view returns (uint128){
    return price;
}

function transfer(address to, uint256 value) public returns (bool success) {
    require(balances[msg.sender] >= value, "Insufficient Funds");

    balances[msg.sender] -= value;
    balances[to] += value;

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

    return true;
}

modifier onlyOwner(){
    require (msg.sender == owner, "You are not the owner");
    _;
}

function mint(address to, uint256 value) public onlyOwner returns (bool){
    balances[to] += value;
    _totalSupply += value;

    emit Mint(to, value);

    return true;
}

function sell(uint256 value) public returns (bool) {
    require (balances[msg.sender] >= value, "You do not have that many tokens in your balance");
    require (address(this).balance >= value * price, "Contract has insufficient funds");

    balances[msg.sender] -= value;
    _totalSupply -= value;

    (bool sent) = (value * price).customSend(msg.sender);
    require(sent, "Failed to send ether into your account");
```

```
        emit Sell(msg.sender, value);

        return true;
    }

    function close() public onlyOwner{
        selfdestruct(owner);
        //This will transfer the contracts balance to the owner too.
    }

    fallback() external payable {}

    receive() external payable {}
}
```