

BDL Coursework 3 Report

Name : Madhav Benoi

student no : s1902743

Part 1 : Smart Contract Programming

High Level Design

The contract implement an Token API, for a user to implement their own Token on the Ethereum blockchain.

Constructor

In this particular contract, the constructor has the following parameters:

- `string memory _name`: The name of the token.
- `string memory _symbol`: The symbol of the token.

The constructor sets the contract owner to the address of the contract deployer, sets the name and symbol of the token to the provided parameters, sets the total supply to 0, and sets the token price to 600 wei.

Internal variables

The following are all the internal variables that are declared in the contract.

- `owner`: The address of the contract owner. This is set in the Constructor. `address` type annotation.
- `total_Supply`: The total token supply. `uint256` is used as type annotation.
- `name`: The name of the token. `string` is used as type.
- `symbol`: The symbol of the token. `string` is used as type.
- `token_price`: The price of the token in wei. This is set to 600 wei during the constructor as specified by the coursework spec. `uint126` is used as type annotation as it is `uint126` in one of the function calls.
- `balances`: A `hashmap` that maps addresses to `uint256`.

Extra things

The only extra thing that was added is the modifier `onlyOwner` to prevent certain users from calling functions in the chain. Everything follow closely to what the coursework spec wants.

Process of buying/selling tokens:

- Users can buy tokens by requesting the owner to mint a specified amount of tokens for their address, using an out-of-band communication mechanism such as email or messaging. The owner can then call the `mint` function to mint the agreed upon amount of tokens for the user's address.
- Users can sell tokens by calling the `sell` function and specifying the number of tokens they want to sell. The contract will send the appropriate amount of wei to the user's address.

Gas Evaluation

Everything is in `units` as gas prices change but the computation remain largely the same in solidity.

Contract Deployment

Contract creation

- Transaction hash :

`0xdaf0d7c84b3f0d342b782c88d6a9bea919270d6300e3158b744e8fda96d04b22`

- Deployment address : `0xD260950502A3EfC1f88F06c04C84667dBe8c6D44`

- Gas : `1222807 units`

The heaviest in terms of computation, this makes sense as the it requires setting up all the variables and data structures.

Minting :

- Transaction hash :

`0xf353229b285c38f6e83f6140cb5d1e0adb34b61fc7997e241a07cb0b8a37b3cb`

- Gas : 71688 units

Second heaviest in terms of gas usage, this is most likely due to the additional `require` check with the modifiers.

Fallback :

- Transaction hash :

`0x2526f69223d83b2f5de1a9b48385e96f21b1425864a588240748036569a232f6`

- Gas : 21088 units

transfer :

- Transaction hash :

```
0x9956f64534a47d205ff791b83b9b527940791a961d95ec3f170facc092c6e13b
```

- Gas : 30580 units

Sell :

- Transaction hash :

```
0x9e471945f9724a943a2771595c697e1ccc4b8afd0f8cd6073fec5ac5334ec9a5
```

- Gas : 44066 units

Keep this units in mind when comparing with the library version of the code.

Closing :

- Transaction hash :

```
0x8c6986674599f7f7643ffbf6ffc43e79c1738b581669eb2ae43e84ebe72ff32d
```

- Gas : 28508

other functions

Does not cost any gas as they're just getter functions, they index the contract bytecode in the blockchain.

Improvement techniques

One of the improvements I made was to ensure that all the variables are `internal` instead of `public`. This makes sense because there are getter functions available for viewing these variables if needed, making it redundant and inefficient to have them as public.

Perhaps another step for improvement is using `bytes32` rather than string memory for token name and symbol. I had considered this but it was mentioned in the piazza that this doesn't matter plus it sacrifices some design choices.

Potential Hazards

Here's a list of hazards I shall be talking about, and some other that will be mentioned.

Reintrancy

This should not happen as the token balance is cleared before any sort of transaction happens, but this is dangerous when the custom library is used.

Griefing and Front running

Both of these are avoided as, grieving the `send()` is avoided and in Front running might not happen.

Other hazards

I have made an additional check for overflow for one of the functions, this was unnecessary, but I am leaving it in as looking into if this check was necessary made me see that in solidity 0.8 onwards, you don't these checks or use the library `Safemath` as these checks are now made by the compiler and hence is unnecessary.

Hence, integer overflows and underflows are not possible in this code.

Another scenario is the trust placed on the owner to mint the coins for the user, if the user does not pay and the coin is minted. The user cannot sell their tokens as the contract is devoid of any money (assuming the contract has no money).

Part 2 : Using Libraries

Write a small description of how you linked your contract to the deployed library.

I had to do some googling and found this article that explained how to link a deployed library to a contract. The article :

<https://medium.com/remix-ide/deploying-with-libraries-on-remix-ide-24f5f7423b60>

Seeing how remix links the library internally, we look at contracts/artifacts/Token.json

```
{
  "deploy": {
    "VM:-": {
      "linkReferences": {},
      "autoDeployLib": true
    },
    "main:1": {
      "linkReferences": {},
      "autoDeployLib": true
    },
    "ropsten:3": {
      "linkReferences": {},
      "autoDeployLib": true
    },
    "rinkeby:4": {
      "linkReferences": {},
      "autoDeployLib": true
    },
    "kovan:42": {
      "linkReferences": {},
      "autoDeployLib": true
    }
  }
}
```

```

    },
    "goerli:5": {
      "linkReferences": {
        "contracts/customLib.sol": {
          "customLib":
"0x9DA4c8B1918BA29eBA145Ee3616BCDFcFAA2FC51"
        }
      },
      "autoDeployLib": false
    }
  }

```

As you can see I've already linked it, but internally this how abi is linked in remix. To mess with this first we need to enable a setting in remix in Settings which generates metadata.

autoDeployLib is set to false as otherwise, remix would deploy our own instance.

Now we can link the library locally and add the relevant lines that will be used. (See code for difference)

Gas difference

During contract creation, an increase in gas units from **1222807** to **1269651** indicates higher computational effort due to more complex logic or more blockchain calls. Similarly, an increase in gas units from **44066** to **57753** for the function **Sell** could be due to a library function call **customSend** in customLib which adds additional computational costs and can increase gas costs of function calls to the contract.

Part 3 : KYC Considerations and Token issuance

Know your customer is basically a way for organisation to prevent fraud like money laundering by verifying certain personal information. This is to ensure that legitimate parties are conducting the transactions.

There needs to be a 3rd party verification that needs to happen in-order to verify that the identity and legitimacy of the user. Once this is done the issuer can then mint the tokens for the user. In order, for the identity of the user to be anonymous we're gonna use a public key encryption.

Assuming, the 3rd party signs the hash of the information of the customer's documents with a private key. The contract issuer has the hash of the information as the id of each customer documents. The customer first does that and then sends the ciphertext to the verifier, which is the contract issuer. The contract issuer has the hash of the documents and uses their

public key to verify the ciphertext and compare it to hash they have.

All this process is done in `Mint` function.

One question is, cant another user just submit real user's ciphertext again. This can be prevented by changing the public key parameters, while this would be slow. This would completely remove the ability to forge a ciphertext or reuse old ones.

Most of these implementation has to be done by a 3rd party as it is not easy to do on-chain.

Appendix

Transaction History

All of this can be viewed in <https://goerli.etherscan.io>

Without custom library

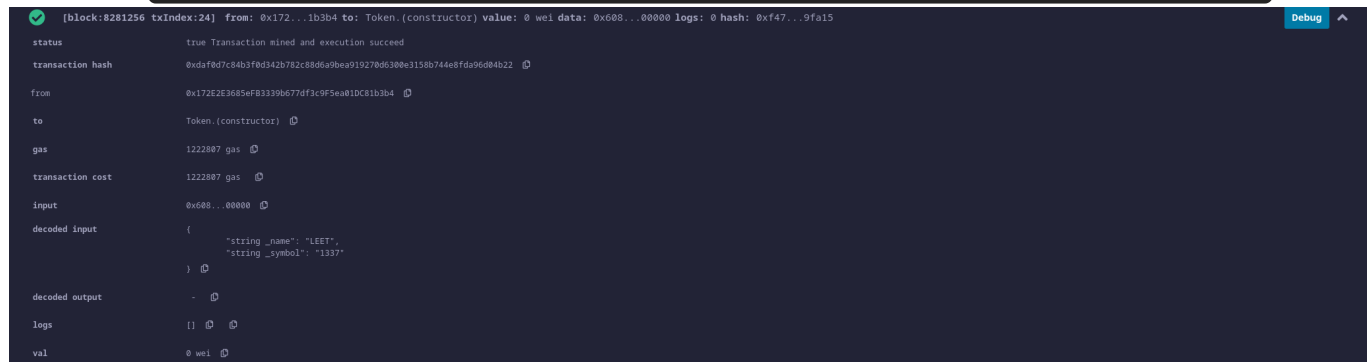
User address : `0xec432F4595EaD9ab0a374C5fB8E842dAD50BaAA6`

Owner address : `0x172E2E3685eFB3339b677df3c9F5ea01DC81b3b4`

Contract address : `0xD260950502A3EfC1f88F06c04C84667dBe8c6D44`

Contract Creation

Tx hash : `0xdaf0d7c84b3f0d342b782c88d6a9bea919270d6300e3158b744e8fda96d04b22`



The screenshot shows a transaction details page for a contract creation on the Goerli testnet. The transaction hash is `0xdaf0d7c84b3f0d342b782c88d6a9bea919270d6300e3158b744e8fda96d04b22`. The transaction was successful, with a status of 'true Transaction mined and execution succeed'. The 'from' field shows the user address `0x172E2E3685eFB3339b677df3c9F5ea01DC81b3b4`. The 'to' field is 'Token.(constructor)'. The 'gas' and 'transaction cost' are both 1222807. The 'input' is `0x608...00000`. The 'decoded input' is a JSON object: `{ "string_name": "LEET", "string_symbol": "1337" }`. The 'decoded output' is an empty array. The 'logs' and 'val' fields are also empty.

Owner calls `Mint` function

Tx Hash : **0xf353229b285c38f6e83f6140cb5d1e0adb34b61fc7997e241a07cb0b8a37b3cb**

```
[block:8281276 txIndex:23] from: 0x172...1b3b4 to: Token.mint(address,uint256) 0xD26...c6D44 value: 0 wei data: 0x40c...0000a logs: 1 hash: 0xef4...097f2
status true Transaction mined and execution succeed
transaction hash 0xf353229b285c38f6e83f6140cb5d1e0adb34b61fc7997e241a07cb0b8a37b3cb
from 0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4
to Token.mint(address,uint256) 0xD260950502A3Efc1f88F06c04C84667DBe8c6D44
gas 71688 gas
transaction cost 71688 gas
input 0x40c...0000a
decoded input {
  "address to": "0xec432f4595ea09ab0a374c5f88E842dAD508aAA6",
  "uint256 value": "18"
}
decoded output -
logs [
  {
    "from": "0xD260950502A3Efc1f88F06c04C84667DBe8c6D44",
    "topic": "0xf0f798a560793a54c3bcfe86a93cde1e73087d944c0ea28544137d4121396885",
    "event": "Mint",
    "args": {
      "to": "0xec432f4595ea09ab0a374c5f88E842dAD508aAA6",
      "i": "18",
      "to": "0xec432f4595ea09ab0a374c5f88E842dAD508aAA6",
      "value": "18"
    }
  }
]
val 0 wei
```

User pays for the tokens by calling fallback functions

Tx hash : **0x2526f69223d83b2f5de1a9b48385e96f21b1425864a588240748036569a232f6**

```
[block:8281296 txIndex:27] from: 0xec4...BAaA6 to: Token.(fallback) 0xD26...c6D44 value: 6000 wei data: 0x123...x1234 logs: 0 hash: 0xb1a...a8e95
status true Transaction mined and execution succeed
transaction hash 0x2526f69223d83b2f5de1a9b48385e96f21b1425864a588240748036569a232f6
from 0xec432f4595ea09ab0a374c5f88E842dAD508aAA6
to Token.(fallback) 0xD260950502A3Efc1f88F06c04C84667DBe8c6D44
gas 21088 gas
transaction cost 21088 gas
input 0x123...x1234
decoded input -
decoded output -
logs []
val 6000 wei
```

Owner mints coin for themself (to test transfer next step)

Tx hash : **0x4032d8aa55524dded3b5c0e49bb43e779071a7dc7b013e1347bc7778c6a4a6dd**

```
[block:8281328 txIndex:35] from: 0x172...1b3b4 to: Token.mint(address,uint256) 0xD26...c6D44 value: 0 wei data: 0x40c...0000a logs: 1 hash: 0xccb...0ec68
status true Transaction mined and execution succeed
transaction hash 0x4032d8aa55524dded3b5c0e49bb43e779071a7dc7b013e1347bc7778c6a4a6dd
from 0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4
to Token.mint(address,uint256) 0xD260950502A3Efc1f88F06c04C84667DBe8c6D44
gas 54588 gas
transaction cost 54588 gas
input 0x40c...0000a
decoded input {
  "address to": "0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4",
  "uint256 value": "18"
}
decoded output -
logs [
  {
    "from": "0xD260950502A3Efc1f88F06c04C84667DBe8c6D44",
    "topic": "0xf0f798a560793a54c3bcfe86a93cde1e73087d944c0ea28544137d4121396885",
    "event": "Mint",
    "args": {
      "to": "0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4",
      "i": "18",
      "to": "0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4",
      "value": "18"
    }
  }
]
val 0 wei
```

Owner transfers 10 tokens to the User

Tx hash: **0x9956f64534a47d205ff791b83b9b527940791a961d95ec3f170facc092c6e13b**

✓ [block:8281363 txIndex:43] from: 0x172...1b3b4 to: Token.transfer(address,uint256) 0xD26...c0d44 value: 0 wei data: 0xa90...0000a logs: 1 hash: 0x007...5b9ac

status

true Transaction mined and execution succeed

transaction hash

0x9956f64534a47d205ff791b83b9b527940791a961d95ec3f170facc092c6e13b

from

0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4

to

Token.transfer(address,uint256) 0xD260950502A3EFC1f88F06c04C84667DBe8c6D44

gas

38580 gas

transaction cost

27745 gas

input

0xa90...0000a

decoded input

{

"address to": "0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4",

"uint256 value": "10"

}

decoded output

-

logs

[

{

"from": "0xD260950502A3EFC1f88F06c04C84667DBe8c6D44",

"topic": "0x09f22ad1be2c89b69c2b068fc376d4a952ba7f163c4a1628f55a4df523b3ef",

"event": "Transfer",

"args": {

"0": "0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4",

"1": "0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4",

"2": "10",

"from": "0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4",

"to": "0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4",

"value": "10"

}

}

}

]

val

0 wei

Debug

User sells 10 token

Tx hash: **0x9e471945f9724a943a2771595c697e1ccc4b8afd0f8cd6073fec5ac5334ec9a5**

✓ [block:8281371 txIndex:27] from: 0xec4...BAA6 to: Token.sell(uint256) 0xD26...c0d44 value: 0 wei data: 0xe48...0000a logs: 1 hash: 0x50f...3d472

status

true Transaction mined and execution succeed

transaction hash

0x9e471945f9724a943a2771595c697e1ccc4b8afd0f8cd6073fec5ac5334ec9a5

from

0xec432f4595eaD9ab0a374C5f8BE842dAD508bAA6

to

Token.sell(uint256) 0xD260950502A3EFC1f88F06c04C84667DBe8c6D44

gas

44066 gas

transaction cost

38782 gas

input

0xe48...0000a

decoded input

{

"uint256 value": "10"

}

decoded output

-

logs

[

{

"from": "0xD260950502A3EFC1f88F06c04C84667DBe8c6D44",

"topic": "0x5e9e95ce313361afcea51a9a15405db228c07325d34df5185502c18d3df09",

"event": "Sell",

"args": {

"0": "0xec432f4595eaD9ab0a374C5f8BE842dAD508bAA6",

"1": "10",

"from": "0xec432f4595eaD9ab0a374C5f8BE842dAD508bAA6",

"value": "10"

}

}

}

]

val

0 wei

Debug

Owner closes

Tx hash: **0x8c6986674599f7f7643ffb6f6ffc43e79c1738b581669eb2ae43e84ebe72ff32d**

✓ [block:8281381 txIndex:12] from: 0x172...1b3b4 to: Token.close() 0xD26...c0d44 value: 0 wei data: 0x43d...726d6 logs: 0 hash: 0xa76...c7871

status

true Transaction mined and execution succeed

transaction hash

0x8c6986674599f7f7643ffb6f6ffc43e79c1738b581669eb2ae43e84ebe72ff32d

from

0x172E2E3685eF8339b677df3c9f5ea01DC81b3b4

to

Token.close() 0xD260950502A3EFC1f88F06c04C84667DBe8c6D44

gas

28588 gas

transaction cost

28588 gas

input

0x43d...726d6

decoded input

{}

decoded output

-

logs

[]

val

0 wei

Debug

With custom library

User address : **0xec432F4595EaD9ab0a374C5fB8E842dAD50BaAA6**

Owner address : **0x172E2E3685eFB3339b677df3c9F5ea01DC81b3b4**

Contract address : **0x67E7F359f623A73ebD984baE3B359f72896E2754**

Contract Creation

Tx hash : **0x3b62e4968bbb635bc75f2013eefe22fa95bcde3aaefee27f5d2614afa5c5fc82**

✔

[block:8291766 txIndex:13] from: 0x172...1b3b4 to: Token.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0x3ab...12d1b

Debug

^

status

true Transaction mined and execution succeed

transaction hash

0x3b62e4968bbb635bc75f2013eefe22fa95bcde3aaefee27f5d2614afa5c5fc82

from

0x172E2E3685eFB3339b677df3c9F5ea01DC81b3b4

to

Token.(constructor)

gas

1269651 gas

transaction cost

1269651 gas

input

0x608...00000

decoded input

{

"string_name": "LEET",

"string_symbol": "1337"

}

decoded output

-

logs

[]

val

0 wei

Mint

Tx hash : **0x248751e84e842087468d59eb1324426ab7b36c0deaa8b024a1f5b280953fbf1b**

✔

[block:8291805 txIndex:11] from: 0x172...1b3b4 to: Token.mint(address,uint256) 0x67E...E2754 value: 0 wei data: 0x40c...0000a logs: 1 hash: 0x912...cef5b

Debug

^

status

true Transaction mined and execution succeed

transaction hash

0x248751e84e842087468d59eb1324426ab7b36c0deaa8b024a1f5b280953fbf1b

from

0x172E2E3685eFB3339b677df3c9F5ea01DC81b3b4

to

Token.mint(address,uint256) 0x67E7F359f623A73ebD984baE3B359f72896E2754

gas

71688 gas

transaction cost

71688 gas

input

0x40c...0000a

decoded input

{

"address to": "0xec432F4595EaD9ab0a374C5fB8E842dAD50BaAA6",

"uint256 value": "18"

}

decoded output

-

logs

[

{

"from": "0x67E7F359f623A73ebD984baE3B359f72896E2754",

"topic": "0x8f079ba560793a54c3bcfe86a93cde1e73807d944c8ea28544137d4121396885",

"event": "Mint",

"args": {

"to": "0xec432F4595EaD9ab0a374C5fB8E842dAD50BaAA6",

"value": "18"

val

0 wei

fallback

Tx hash : **0xe97ed497e36dbf51c8b40d4f9fdbddb92f63f54c14a4c77f2b227a08d17bd03c**

✔

[block:8291809 txIndex:6] from: 0x172...1b3b4 to: Token.(fallback) 0x67E...E2754 value: 6000 wei data: 0x123...x1234 logs: 0 hash: 0x75a...4f64a

Debug

^

status

true Transaction mined and execution succeed

transaction hash

0xe97ed497e36dbf51c8b40d4f9fdbddb92f63f54c14a4c77f2b227a08d17bd03c

from

0x172E2E3685eFB3339b677df3c9F5ea01DC81b3b4

to

Token.(fallback) 0x67E7F359f623A73ebD984baE3B359f72896E2754

gas

21088 gas

transaction cost

21088 gas

input

0x123...x1234

decoded input

-

decoded output

-

logs

[]

val

6000 wei

Sell

Tx hash : **0x91f5d2ff7d29a05a3036edff5627d59c51955c45b2e826bbbbc8e77a770f6d2c**

✓

[block:8291813 txIndex:9] from: 0xec4...BaA6 to: Token.sell(uint256) 0x67E...E2754 value: 0 wei data: 0xe48...0000a logs: 1 hash: 0x5fb...7dc32

Debug

⬆

status

true Transaction mined and execution succeed

transaction hash

0x91f5d2ff7d29a05a3036edff5627d59c51955c45b2e826bbbbc8e77a770f6d2c

from

0xec432f4595e09ab0a374c5fb8e842d0508aaA6

to

Token.sell(uint256) 0x67E7F359f623A73eb0984baE3B359f72896E2754

gas

57753 gas

transaction cost

47922 gas

input

0xe48...0000a

decoded input

(
 "uint256 value": "10"
)

decoded output

-

logs

[
 {
 "from": "0x67E7F359f623A73eb0984baE3B359f72896E2754",
 "topic": "0xe5e995ce3133561afceaa51a9a154d5db228c07525034df5185582c18d3df09",
 "event": "Sell",
 "args": {
 "0": "0xec432f4595e09ab0a374c5fb8e842d0508aaA6",
 "1": "10",
 "from": "0xec432f4595e09ab0a374c5fb8e842d0508aaA6",
 "value": "10"
 }
 }
)
]

val

0 wei

Close

Tx hash : **0x6a22f63e7d1239b6f3402db5acefbdb1e9095a1e0c7d783c00d929b76b4b49bf**

✓

[block:8291824 txIndex:8] from: 0x172...1b3b4 to: Token.close() 0x67E...E2754 value: 0 wei data: 0x43d...726d6 logs: 0 hash: 0xbe1...a86b7

Debug

⬆

status

true Transaction mined and execution succeed

transaction hash

0x6a22f63e7d1239b6f3402db5acefbdb1e9095a1e0c7d783c00d929b76b4b49bf

from

0x172e2e3685efB3339b677d7f3c9f5ea81DC81b3b4

to

Token.close() 0x67E7F359f623A73eb0984baE3B359f72896E2754

gas

28588 gas

transaction cost

28589 gas

input

0x43d...726d6

decoded input

()

decoded output

-

logs

[]

val

0 wei

Code

Without library

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract Token {
    // The address of the contract owner
    address internal owner;
    // The total token supply
    uint256 internal total_Supply;
    // The token name
    string internal name;
    // The token symbol
```

```

string internal symbol;
// The token price in wei
uint128 internal token_price;

// Events
event Transfer(address indexed from, address indexed to, uint256 value);
event Mint(address indexed to, uint256 value);
event Sell(address indexed from, uint256 value);

// mapping that maps addresses to balances
mapping(address => uint256) public balances;

// Constructor function that sets the owner, name, symbol, and token
price
// of the token contract.
constructor( string memory _name, string memory _symbol) {
    owner = msg.sender;
    name = _name;
    symbol = _symbol;
    total_Supply = 0;
    token_price = 600 wei;
}

// return the total supply of the token
function totalSupply() public view returns (uint256) {
    return total_Supply;
}

// function that returns balances of an address
function balanceOf(address _account) public view returns (uint256) {
    return balances[_account];
}

// a view function that returns a string with the token's name
function getName() public view returns (string memory) {
    return name;
}

// a view function that returns a string with the token's symbol
function getSymbol() public view returns (string memory) {
    return symbol;
}

// a view function that returns the token's price in wei
function getPrice() public view returns (uint128) {
    return token_price;
}

```

```

}

// a function that transfers tokens from the sender to another address
function transfer(address to, uint256 value) public returns (bool) {
    require(to != address(0), "Invalid address to transfer to");

    // value cannot be zero
    require(value != 0, "Cannot transfer zero tokens");
    // this should prevent underflow
    require(balanceOf(msg.sender) >= value, "Not enough balance to
transfer");
    // make sure balance is not overflowing
    require (balanceOf(to) + value >= balanceOf(to), "Overflow");
    balances[msg.sender] -= value;
    balances[to] += value;
    emit Transfer(msg.sender, to, value);
    return true;
}

// a function that enables only the owner to mint tokens to specified
address
function mint(address to, uint256 value) public onlyOwner returns (bool)
{
    require(to != address(0), "Invalid address to mint to");
    // make sure balance is not overflowing
    require (balanceOf(to) + value >= balanceOf(to), "Overflow");
    // value cannot be zero
    require(value != 0, "Cannot mint zero tokens");
    balances[to] += value;

    total_Supply += value;

    emit Mint(to, value);
    return true;
}

// a function that enables the token owners to sell their tokens
function sell(uint256 value) public payable returns (bool) {
    // value cannot be zero
    require(value != 0, "Cannot sell zero tokens");
    // this should prevent underflow
    require(balanceOf(msg.sender) >= value, "Not enough balance to
sell");

```

```

        // amount = token_price * value
        uint256 amount = token_price * value;

        // make sure the contract has enough balance to pay the seller
        require(address(this).balance >= amount, "Not enough balance to pay
the seller");

        // reduce the balance of the seller
        balances[msg.sender] -= value;

        // reduce the total supply
        total_Supply -= value;
        // transfer from the contract to the sender
        payable(msg.sender).transfer(amount);
        emit Sell(msg.sender, value);
        return true;
    }

    // close() a function that enables only the owner to destroy the
    contract; the contract's balance in wei, at the moment of destruction,
    should be transferred to the owner address
    function close() public onlyOwner {
        selfdestruct(payable(owner));
    }

    // fallback function to let the contract receive ether
    fallback() external payable {
        //
    }

    // receive function to let the contract receive ether
    receive() external payable {
        //
    }

    // ----- Modifiers -----

    modifier onlyOwner {

```

```

        require(msg.sender == owner);
    _;
}

}

```

With library

```

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "./customLib.sol";

contract Token {
    using customLib for uint256;
    using customLib for address;

    // The address of the contract owner
    address internal owner;
    // The total token supply
    uint256 internal total_Supply;
    // The token name
    string internal name;
    // The token symbol
    string internal symbol;
    // The token price in wei
    uint128 internal token_price;

    // Events
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Mint(address indexed to, uint256 value);
    event Sell(address indexed from, uint256 value);

    // mapping that maps addresses to balances
    mapping(address => uint256) public balances;

    // Constructor function that sets the owner, name, symbol, and token
    price
    // of the token contract.
    constructor( string memory _name, string memory _symbol) {
        owner = msg.sender;
        name = _name;
        symbol = _symbol;
    }
}

```

```

        total_Supply = 0;
        token_price = 600;
    }

    // return the total supply of the token
    function totalSupply() public view returns (uint256) {
        return total_Supply;
    }

    // function that returns balances of an address
    function balanceOf(address _account) public view returns (uint256) {
        return balances[_account];
    }

    // a view function that returns a string with the token's name
    function getName() public view returns (string memory) {
        return name;
    }

    // a view function that returns a string with the token's symbol
    function getSymbol() public view returns (string memory) {
        return symbol;
    }

    // a view function that returns the token's price in wei
    function getPrice() public view returns (uint128) {
        return token_price;
    }

    // a function that transfers tokens from the sender to another address
    function transfer(address to, uint256 value) public returns (bool) {
        require(to != address(0), "Invalid address to transfer to");

        // value cannot be zero
        require(value != 0, "Cannot transfer zero tokens");
        require(balanceOf(msg.sender) >= value, "Not enough balance to
transfer");
        balances[msg.sender] -= value;
        balances[to] += value;
        emit Transfer(msg.sender, to, value);
        return true;
    }

    // a function that enables only the owner to mint tokens to specified

```

address

```
function mint(address to, uint256 value) public onlyOwner returns (bool)
{
    require(to != address(0), "Invalid address to mint to");
    // make sure balance is not overflowing
    require (balanceOf(to) + value >= balanceOf(to), "Overflow");
    // value cannot be zero
    require(value != 0, "Cannot mint zero tokens");
    balances[to] += value;

    total_Supply += value;

    emit Mint(to, value);
    return true;
}

// a function that enables the token owners to sell their tokens
function sell(uint256 value) public payable returns (bool) {
    // value cannot be zero
    require(value != 0, "Cannot sell zero tokens");
    require(balanceOf(msg.sender) >= value, "Not enough balance to
sell");

    // amount = token_price * value
    uint256 amount = token_price * value;

    // make sure the contract has enough balance to pay the seller
    require(address(this).balance >= amount, "Not enough balance to pay
the seller");

    // reduce the balance of the seller
    balances[msg.sender] -= value;

    // reduce the total supply
    total_Supply -= value;
    // transfer from the contract to the sender
    bool success = amount.customSend(msg.sender);
    emit Sell(msg.sender, value);
    return success;
}

// close() a function that enables only the owner to destroy the
contract; the contract's balance in wei, at the moment of destruction,
should be transferred to the owner address
function close() public onlyOwner {
```



```
        selfdestruct(payable(owner));
    }

    // fallback function to let the contract receive ether
    fallback() external payable {
        //

    }

    // receive function to let the contract receive ether
    receive() external payable {
        //
    }


// ----- Modifiers -----

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

}
```