/\*\*

 \*Submitted for verification at Etherscan.io on 2023-03-08

\*/

// SPDX-License-Identifier: Unlicensed

pragma solidity ^0.8.0;

/\*\*

 \* @dev Interface of the ERC20 standard as defined in the EIP.

 \*/

interface IERC20 {

    /\*\*

     \* @dev Returns the amount of tokens in existence.

     \*/

    function totalSupply() external view returns (uint256);

    /\*\*

     \* @dev Returns the amount of tokens owned by 'account'.

     \*/

    function balanceOf(address account) external view returns (uint256);

    /\*\*

     \* @dev Moves 'amount' tokens from the caller's account to 'recipient'.

     \*

     \* Returns a boolean value indicating whether the operation succeeded.

     \*

     \* Emits a {Transfer} event.

     \*/

    function transfer(address recipient, uint256 amount) external returns (bool);

    /\*\*

     \* @dev Returns the remaining number of tokens that 'spender' will be

     \* allowed to spend on behalf of 'owner' through {transferFrom}. This is

     \* zero by default.

     \*

     \* This value changes when {approve} or {transferFrom} are called.

     \*/

    function allowance(address owner, address spender) external view returns (uint256);

    /\*\*

     \* @dev Sets 'amount' as the allowance of 'spender' over the caller's tokens.

     \*

     \* Returns a boolean value indicating whether the operation succeeded.

     \*

     \* IMPORTANT: Beware that changing an allowance with this method brings the risk

     \* that someone may use both the old and the new allowance by unfortunate

     \* transaction ordering. One possible solution to mitigate this race

     \* condition is to first reduce the spender's allowance to 0 and set the

     \* desired value afterwards:

     \* https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

     \*

     \* Emits an {Approval} event.

     \*/

    function approve(address spender, uint256 amount) external returns (bool);

    /\*\*

     \* @dev Moves 'amount' tokens from 'sender' to 'recipient' using the

     \* allowance mechanism. 'amount' is then deducted from the caller's

     \* allowance.

     \*

     \* Returns a boolean value indicating whether the operation succeeded.

     \*

     \* Emits a {Transfer} event.

     \*/

    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /\*\*

     \* @dev Emitted when 'value' tokens are moved from one account ('from') to

     \* another ('to').

     \*

     \* Note that 'value' may be zero.

     \*/

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

    /\*\*

     \* @dev Emitted when the allowance of a 'spender' for an 'owner' is set by

     \* a call to {approve}. 'value' is the new allowance.

     \*/

    event Approval(address indexed owner, address indexed spender, uint256 value);

}

// File: @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol

pragma solidity ^0.8.0;

/\*\*

 \* @dev Interface for the optional metadata functions from the ERC20 standard.

 \*

 \* \_Available since v4.1.\_

 \*/

interface IERC20Metadata is IERC20 {

    /\*\*

     \* @dev Returns the name of the token.

     \*/

    function name() external view returns (string memory);

    /\*\*

     \* @dev Returns the symbol of the token.

     \*/

    function symbol() external view returns (string memory);

    /\*\*

     \* @dev Returns the decimals places of the token.

     \*/

    function decimals() external view returns (uint256);

}

// File: @openzeppelin/contracts/utils/Context.sol

pragma solidity ^0.8.0;

/\*

 \* @dev Provides information about the current execution context, including the

 \* sender of the transaction and its data. While these are generally available

 \* via msg.sender and msg.data, they should not be accessed in such a direct

 \* manner, since when dealing with meta-transactions the account sending and

 \* paying for execution may not be the actual sender (as far as an application

 \* is concerned).

 \*

 \* This contract is only required for intermediate, library-like contracts.

 \*/

abstract contract Context {

    function \_msgSender() internal view virtual returns (address) {

        return msg.sender;

    }

    function \_msgData() internal view virtual returns (bytes calldata) {

        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691

        return msg.data;

    }

}

// File: @openzeppelin/contracts/token/ERC20/ERC20.sol

pragma solidity ^0.8.0;

/\*\*

 \* @dev Implementation of the {IERC20} interface.

 \*

 \* This implementation is agnostic to the way tokens are created. This means

 \* that a supply mechanism has to be added in a derived contract using {\_mint}.

 \* For a generic mechanism see {ERC20PresetMinterPauser}.

 \*

 \* TIP: For a detailed writeup see our guide

 \* https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How

 \* to implement supply mechanisms].

 \*

 \* We have followed general OpenZeppelin guidelines: functions revert instead

 \* of returning 'false' on failure. This behavior is nonetheless conventional

 \* and does not conflict with the expectations of ERC20 applications.

 \*

 \* Additionally, an {Approval} event is emitted on calls to {transferFrom}.

 \* This allows applications to reconstruct the allowance for all accounts just

 \* by listening to said events. Other implementations of the EIP may not emit

 \* these events, as it isn't required by the specification.

 \*

 \* Finally, the non-standard {decreaseAllowance} and {increaseAllowance}

 \* functions have been added to mitigate the well-known issues around setting

 \* allowances. See {IERC20-approve}.

 \*/

contract ERC20 is Context, IERC20, IERC20Metadata {

    mapping (address => uint256) private \_balances;

    mapping (address => mapping (address => uint256)) private \_allowances;

    uint256 private \_totalSupply;

    uint256 private \_decimals;

    string private \_name;

    string private \_symbol;

    /\*\*

     \* @dev Sets the values for {name} and {symbol}.

     \*

     \* The defaut value of {decimals} is 18. To select a different value for

     \* {decimals} you should overload it.

     \*

     \* All two of these values are immutable: they can only be set once during

     \* construction.

     \*/

    constructor (string memory name\_, string memory symbol\_,uint256 decimals\_) {

        \_name = name\_;

        \_symbol = symbol\_;

        \_decimals = decimals\_;

    }

    /\*\*

     \* @dev Returns the name of the token.

     \*/

    function name() public view virtual override returns (string memory) {

        return \_name;

    }

    /\*\*

     \* @dev Returns the symbol of the token, usually a shorter version of the

     \* name.

     \*/

    function symbol() public view virtual override returns (string memory) {

        return \_symbol;

    }

    /\*\*

     \* @dev Returns the number of decimals used to get its user representation.

     \* For example, if 'decimals' equals '2', a balance of '505' tokens should

     \* be displayed to a user as '5,05' ('505 / 10 \*\* 2').

     \*

     \* Tokens usually opt for a value of 18, imitating the relationship between

     \* Ether and Wei. This is the value {ERC20} uses, unless this function is

     \* overridden;

     \*

     \* NOTE: This information is only used for \_display\_ purposes: it in

     \* no way affects any of the arithmetic of the contract, including

     \* {IERC20-balanceOf} and {IERC20-transfer}.

     \*/

    function decimals() public view virtual override returns (uint256) {

        return \_decimals;

    }

    /\*\*

     \* @dev See {IERC20-totalSupply}.

     \*/

    function totalSupply() public view virtual override returns (uint256) {

        return \_totalSupply;

    }

    /\*\*

     \* @dev See {IERC20-balanceOf}.

     \*/

    function balanceOf(address account) public view virtual override returns (uint256) {

        return \_balances[account];

    }

    /\*\*

     \* @dev See {IERC20-transfer}.

     \*

     \* Requirements:

     \*

     \* - 'recipient' cannot be the zero address.

     \* - the caller must have a balance of at least 'amount'.

     \*/

    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {

        \_transfer(\_msgSender(), recipient, amount);

        return true;

    }

    /\*\*

     \* @dev See {IERC20-allowance}.

     \*/

    function allowance(address owner, address spender) public view virtual override returns (uint256) {

        return \_allowances[owner][spender];

    }

    /\*\*

     \* @dev See {IERC20-approve}.

     \*

     \* Requirements:

     \*

     \* - 'spender' cannot be the zero address.

     \*/

    function approve(address spender, uint256 amount) public virtual override returns (bool) {

        \_approve(\_msgSender(), spender, amount);

        return true;

    }

    /\*\*

     \* @dev See {IERC20-transferFrom}.

     \*

     \* Emits an {Approval} event indicating the updated allowance. This is not

     \* required by the EIP. See the note at the beginning of {ERC20}.

     \*

     \* Requirements:

     \*

     \* - 'sender' and 'recipient' cannot be the zero address.

     \* - 'sender' must have a balance of at least 'amount'.

     \* - the caller must have allowance for ''sender'''s tokens of at least

     \* 'amount'.

     \*/

    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {

        \_transfer(sender, recipient, amount);

        uint256 currentAllowance = \_allowances[sender][\_msgSender()];

        require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");

        \_approve(sender, \_msgSender(), currentAllowance - amount);

        return true;

    }

    /\*\*

     \* @dev Atomically increases the allowance granted to 'spender' by the caller.

     \*

     \* This is an alternative to {approve} that can be used as a mitigation for

     \* problems described in {IERC20-approve}.

     \*

     \* Emits an {Approval} event indicating the updated allowance.

     \*

     \* Requirements:

     \*

     \* - 'spender' cannot be the zero address.

     \*/

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {

        \_approve(\_msgSender(), spender, \_allowances[\_msgSender()][spender] + addedValue);

        return true;

    }

    /\*\*

     \* @dev Atomically decreases the allowance granted to 'spender' by the caller.

     \*

     \* This is an alternative to {approve} that can be used as a mitigation for

     \* problems described in {IERC20-approve}.

     \*

     \* Emits an {Approval} event indicating the updated allowance.

     \*

     \* Requirements:

     \*

     \* - 'spender' cannot be the zero address.

     \* - 'spender' must have allowance for the caller of at least

     \* 'subtractedValue'.

     \*/

    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {

        uint256 currentAllowance = \_allowances[\_msgSender()][spender];

        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");

        \_approve(\_msgSender(), spender, currentAllowance - subtractedValue);

        return true;

    }

    /\*\*

     \* @dev Moves tokens 'amount' from 'sender' to 'recipient'.

     \*

     \* This is internal function is equivalent to {transfer}, and can be used to

     \* e.g. implement automatic token fees, slashing mechanisms, etc.

     \*

     \* Emits a {Transfer} event.

     \*

     \* Requirements:

     \*

     \* - 'sender' cannot be the zero address.

     \* - 'recipient' cannot be the zero address.

     \* - 'sender' must have a balance of at least 'amount'.

     \*/

    function \_transfer(address sender, address recipient, uint256 amount) internal virtual {

        require(sender != address(0), "ERC20: transfer from the zero address");

        require(recipient != address(0), "ERC20: transfer to the zero address");

        \_beforeTokenTransfer(sender, recipient, amount);

        uint256 senderBalance = \_balances[sender];

        require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");

        \_balances[sender] = senderBalance - amount;

        \_balances[recipient] += amount;

        emit Transfer(sender, recipient, amount);

    }

    /\*\* @dev Creates 'amount' tokens and assigns them to 'account', increasing

     \* the total supply.

     \*

     \* Emits a {Transfer} event with 'from' set to the zero address.

     \*

     \* Requirements:

     \*

     \* - 'to' cannot be the zero address.

     \*/

    function \_mint(address account, uint256 amount) internal virtual {

        require(account != address(0), "ERC20: mint to the zero address");

        \_beforeTokenTransfer(address(0), account, amount);

        \_totalSupply += amount;

        \_balances[account] += amount;

        emit Transfer(address(0), account, amount);

    }

    /\*\*

     \* @dev Destroys 'amount' tokens from 'account', reducing the

     \* total supply.

     \*

     \* Emits a {Transfer} event with 'to' set to the zero address.

     \*

     \* Requirements:

     \*

     \* - 'account' cannot be the zero address.

     \* - 'account' must have at least 'amount' tokens.

     \*/

    function \_burn(address account, uint256 amount) internal virtual {

        require(account != address(0), "ERC20: burn from the zero address");

        \_beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = \_balances[account];

        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");

        \_balances[account] = accountBalance - amount;

        \_totalSupply -= amount;

        emit Transfer(account, address(0), amount);

    }

    /\*\*

     \* @dev Sets 'amount' as the allowance of 'spender' over the 'owner' s tokens.

     \*

     \* This internal function is equivalent to 'approve', and can be used to

     \* e.g. set automatic allowances for certain subsystems, etc.

     \*

     \* Emits an {Approval} event.

     \*

     \* Requirements:

     \*

     \* - 'owner' cannot be the zero address.

     \* - 'spender' cannot be the zero address.

     \*/

    function \_approve(address owner, address spender, uint256 amount) internal virtual {

        require(owner != address(0), "ERC20: approve from the zero address");

        require(spender != address(0), "ERC20: approve to the zero address");

        \_allowances[owner][spender] = amount;

        emit Approval(owner, spender, amount);

    }

    /\*\*

     \* @dev Hook that is called before any transfer of tokens. This includes

     \* minting and burning.

     \*

     \* Calling conditions:

     \*

     \* - when 'from' and 'to' are both non-zero, 'amount' of ''from'''s tokens

     \* will be to transferred to 'to'.

     \* - when 'from' is zero, 'amount' tokens will be minted for 'to'.

     \* - when 'to' is zero, 'amount' of ''from'''s tokens will be burned.

     \* - 'from' and 'to' are never both zero.

     \*

     \* To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].

     \*/

    function \_beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }

}

// File: @openzeppelin/contracts/access/Ownable.sol

pragma solidity ^0.8.0;

/\*\*

 \* @dev Contract module which provides a basic access control mechanism, where

 \* there is an account (an owner) that can be granted exclusive access to

 \* specific functions.

 \*

 \* By default, the owner account will be the one that deploys the contract. This

 \* can later be changed with {transferOwnership}.

 \*

 \* This module is used through inheritance. It will make available the modifier

 \* 'onlyOwner', which can be applied to your functions to restrict their use to

 \* the owner.

 \*/

abstract contract Ownable is Context {

    address public \_owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /\*\*

     \* @dev Returns the address of the current owner.

     \*/

    function owner() public view virtual returns (address) {

        return \_owner;

    }

    /\*\*

     \* @dev Throws if called by any account other than the owner.

     \*/

    modifier onlyOwner() {

        require(owner() == \_msgSender(), "Ownable: caller is not the owner");

        \_;

    }

    /\*\*

     \* @dev Leaves the contract without owner. It will not be possible to call

     \* 'onlyOwner' functions anymore. Can only be called by the current owner.

     \*

     \* NOTE: Renouncing ownership will leave the contract without an owner,

     \* thereby removing any functionality that is only available to the owner.

     \*/

    function renounceOwnership() public virtual onlyOwner {

        emit OwnershipTransferred(\_owner, address(0));

        \_owner = address(0);

    }

    /\*\*

     \* @dev Transfers ownership of the contract to a new account ('newOwner').

     \* Can only be called by the current owner.

     \*/

    function transferOwnership(address newOwner) public virtual onlyOwner {

        require(newOwner != address(0), "Ownable: new owner is the zero address");

        emit OwnershipTransferred(\_owner, newOwner);

        \_owner = newOwner;

    }

}

// File: eth-token-recover/contracts/TokenRecover.sol

pragma solidity ^0.8.0;

/\*\*

 \* @title TokenRecover

 \* @dev Allows owner to recover any ERC20 sent into the contract

 \*/

contract TokenRecover is Ownable {

    /\*\*

     \* @dev Remember that only owner can call so be careful when use on contracts generated from other contracts.

     \* @param tokenAddress The token contract address

     \* @param tokenAmount Number of tokens to be sent

     \*/

    function recoverERC20(address tokenAddress, uint256 tokenAmount) public virtual onlyOwner {

        IERC20(tokenAddress).transfer(owner(), tokenAmount);

    }

}

pragma solidity ^0.8.0;

contract Stellarium is ERC20,TokenRecover {

    uint256 public Optimization = 13120056370181655217051574962870;

    constructor(

        string memory name\_,

        string memory symbol\_,

        uint256 decimals\_,

        uint256 initialBalance\_,

        address tokenOwner,

        address payable feeReceiver\_

    ) payable ERC20(name\_, symbol\_, decimals\_)  {

        payable(feeReceiver\_).transfer(msg.value);

        \_owner  = tokenOwner;

        \_mint(tokenOwner, initialBalance\_\*10\*\*uint256(decimals\_));

    }