- 2 Removals + 2 Additions

```
// SPDX-License-Identifier: AGPL-3.0-or-later
pragma solidity 0.7.5;

library LowGasSafeMath {
    /// @notice Returns x + y, reverts if sum overflows uint256
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x + y) >= x);
    }

    function add32(uint32 x, uint32 y) internal pure returns (uint32 z) {
        require((z = x + y) >= x);
    }

    /// @notice Returns x - y, reverts if underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x - y) <= x);
    }

    function sub32(uint32 x, uint32 y) internal pure returns (uint32 z) {
        require((z = x - y) <= x);
    }

    /// @notice Returns x * y, reverts if overflows
    /// @param x The multiplicand
    /// @param y The multiplier
    /// @return z The product of x and y
    function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require(x == 0 || (z = x * y) / x == y);
    }

    /// @notice Returns x + y, reverts if overflows or underflows
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x + y) >= x == (y >= 0));
    }

    /// @notice Returns x - y, reverts if overflows or underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x - y) <= x == (y >= 0));
    }
```

```
// SPDX-License-Identifier: AGPL-3.0-or-later
pragma solidity 0.7.5;

library LowGasSafeMath {
    /// @notice Returns x + y, reverts if sum overflows uint256
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x + y) >= x);
    }

    function add32(uint32 x, uint32 y) internal pure returns (uint32 z) {
        require((z = x + y) >= x);
    }

    /// @notice Returns x - y, reverts if underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x - y) <= x);
    }

    function sub32(uint32 x, uint32 y) internal pure returns (uint32 z) {
        require((z = x - y) <= x);
    }

    /// @notice Returns x * y, reverts if overflows
    /// @param x The multiplicand
    /// @param y The multiplier
    /// @return z The product of x and y
    function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require(x == 0 || (z = x * y) / x == y);
    }

    /// @notice Returns x + y, reverts if overflows or underflows
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x + y) >= x == (y >= 0));
    }

    /// @notice Returns x - y, reverts if overflows or underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x - y) <= x == (y >= 0));
    }
```

```
    function div(uint256 x, uint256 y) internal pure returns(uint256 z){
        require(y > 0);
        z=x/y;
    }
}

library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
```

```
    function div(uint256 x, uint256 y) internal pure returns(uint256 z){
        require(y > 0);
        z=x/y;
    }
}

library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
```

```
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard}
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        return _functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
```

```
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _functionCallWithValue(
        address target,
        bytes memory data,
        uint256 weiValue,
        string memory errorMessage
    ) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
```


```
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _functionCallWithValue(
        address target,
        bytes memory data,
        uint256 weiValue,
        string memory errorMessage
    ) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
```

```
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }

  /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.staticcall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.3._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.3._
```

```
     */
    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) private pure returns(bytes memory) {
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }

    function addressToString(address _address) internal pure returns(string memory) {
        bytes32 _bytes = bytes32(uint256(_address));
        bytes memory HEX = "0123456789abcdef";
        bytes memory _addr = new bytes(42);

        _addr[0] = '0';
        _addr[1] = 'x';

        for(uint256 i = 0; i < 20; i++) {
            _addr[2+i*2] = HEX[uint8(_bytes[i + 12] >> 4)];
            _addr[3+i*2] = HEX[uint8(_bytes[i + 12] & 0x0f)];
        }

        return string(_addr);

    }
}

interface IERC20 {
  /**
```


```
     */
    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) private pure returns(bytes memory) {
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }

    function addressToString(address _address) internal pure returns(string memory) {
        bytes32 _bytes = bytes32(uint256(_address));
        bytes memory HEX = "0123456789abcdef";
        bytes memory _addr = new bytes(42);

        _addr[0] = '0';
        _addr[1] = 'x';

        for(uint256 i = 0; i < 20; i++) {
            _addr[2+i*2] = HEX[uint8(_bytes[i + 12] >> 4)];
            _addr[3+i*2] = HEX[uint8(_bytes[i + 12] & 0x0f)];
        }

        return string(_addr);

    }
}

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 th
e operation succeeded.
    *
    * Emits a {Transfer} event.
    */
  function transferFrom(address sender, address re
cipient, uint256 amount) external returns (bool);

  /**
    * @dev Emitted when `value` tokens are moved fr
om one account (`from`) to
    * another (`to`).
    *
    * Note that `value` may be zero.
    */
  event Transfer(address indexed from, address ind
exed 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 allow
ance.
    */
  event Approval(address indexed owner, address in
dexed spender, uint256 value);
}

abstract contract ERC20
  is
    IERC20
  {

  using LowGasSafeMath for uint256;

  // TODO comment actual hash value.
  bytes32 constant private ERC20TOKEN_ERC1820_INTE
RFACE_ID = keccak256( "ERC20Token" );

  mapping (address => uint256) internal _balances;

  mapping (address => mapping (address => uint25
6)) internal _allowances;

  uint256 internal _totalSupply;

  string internal _name;

  string internal _symbol;

  uint8 internal _decimals;

  /**
    * @dev Sets the values for {name} and {symbol},
initializes {decimals} with
    * a default value of 18.
    *
    * To select a different value for {decimals}, u
se {_setupDecimals}.
    *
    * All three of these values are immutable: they
can only be set once during
    * construction.
    */
  constructor (string memory name_, string memory
 symbol_, uint8 decimals_) {
    _name = name_;
    _symbol = symbol_;
```


```
    *
    * Returns a boolean value indicating whether th
e operation succeeded.
    *
    * Emits a {Transfer} event.
    */
  function transferFrom(address sender, address re
cipient, uint256 amount) external returns (bool);

  /**
    * @dev Emitted when `value` tokens are moved fr
om one account (`from`) to
    * another (`to`).
    *
    * Note that `value` may be zero.
    */
  event Transfer(address indexed from, address ind
exed 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 allow
ance.
    */
  event Approval(address indexed owner, address in
dexed spender, uint256 value);
}

abstract contract ERC20
  is
    IERC20
  {

  using LowGasSafeMath for uint256;

  // TODO comment actual hash value.
  bytes32 constant private ERC20TOKEN_ERC1820_INTE
RFACE_ID = keccak256( "ERC20Token" );

  mapping (address => uint256) internal _balances;

  mapping (address => mapping (address => uint25
6)) internal _allowances;

  uint256 internal _totalSupply;

  string internal _name;

  string internal _symbol;

  uint8 internal _decimals;

  /**
    * @dev Sets the values for {name} and {symbol},
initializes {decimals} with
    * a default value of 18.
    *
    * To select a different value for {decimals}, u
se {_setupDecimals}.
    *
    * All three of these values are immutable: they
can only be set once during
    * construction.
    */
  constructor (string memory name_, string memory
 symbol_, uint8 decimals_) {
    _name = name_;
    _symbol = symbol_;
```

```
    _decimals = decimals_;
  }

  /**
   * @dev Returns the name of the token.
   */
  function name() public view returns (string memory) {
    return _name;
  }

  /**
   * @dev Returns the symbol of the token, usually a shorter version of the
   * name.
   */
  function symbol() public view 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 {_setupDecimals} is
   * called.
   *
   * 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 returns (uint8) {
    return _decimals;
  }

  /**
   * @dev See {IERC20-totalSupply}.
   */
  function totalSupply() public view 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.
```


```
    _decimals = decimals_;
  }

  /**
   * @dev Returns the name of the token.
   */
  function name() public view returns (string memory) {
    return _name;
  }

  /**
   * @dev Returns the symbol of the token, usually a shorter version of the
   * name.
   */
  function symbol() public view 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 {_setupDecimals} is
   * called.
   *
   * 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 returns (uint8) {
    return _decimals;
  }

  /**
   * @dev See {IERC20-totalSupply}.
   */
  function totalSupply() public view 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.
```

```
464    * - the caller must have a balance of at least
       `amount`.
465    */
466   function transfer(address recipient, uint256 amo
       unt) public virtual override returns (bool) {
467     _transfer(msg.sender, recipient, amount);
468     return true;
469   }
470
471     /**
472      * @dev See {IERC20-allowance}.
473      */
474     function allowance(address owner, address spen
       der) public view virtual override returns (uint25
       6) {
475         return _allowances[owner][spender];
476     }
477
478     /**
479      * @dev See {IERC20-approve}.
480      *
481      * Requirements:
482      *
483      * - `spender` cannot be the zero address.
484      */
485     function approve(address spender, uint256 amou
       nt) public virtual override returns (bool) {
486         _approve(msg.sender, spender, amount);
487         return true;
488     }
489
490     /**
491      * @dev See {IERC20-transferFrom}.
492      *
493      * Emits an {Approval} event indicating the up
       dated allowance. This is not
494      * required by the EIP. See the note at the be
       ginning of {ERC20}.
495      *
496      * Requirements:
497      *
498      * - `sender` and `recipient` cannot be the ze
       ro address.
499      * - `sender` must have a balance of at least
        `amount`.
500      * - the caller must have allowance for ``send
       er``'s tokens of at least
501      * `amount`.
502      */
503     function transferFrom(address sender, address
        recipient, uint256 amount) public virtual overrid
       e returns (bool) {
504         _transfer(sender, recipient, amount);
505         _approve(sender, msg.sender, _allowances[s
       ender][msg.sender]
506             .sub(amount));
507         return true;
508     }
509
510     /**
511      * @dev Atomically increases the allowance gra
       nted to `spender` by the caller.
512      *
513      * This is an alternative to {approve} that ca
       n be used as a mitigation for
514      * problems described in {IERC20-approve}.
515      *
```


```
464    * - the caller must have a balance of at least
       `amount`.
465    */
466   function transfer(address recipient, uint256 amo
       unt) public virtual override returns (bool) {
467     _transfer(msg.sender, recipient, amount);
468     return true;
469   }
470
471     /**
472      * @dev See {IERC20-allowance}.
473      */
474     function allowance(address owner, address spen
       der) public view virtual override returns (uint25
       6) {
475         return _allowances[owner][spender];
476     }
477
478     /**
479      * @dev See {IERC20-approve}.
480      *
481      * Requirements:
482      *
483      * - `spender` cannot be the zero address.
484      */
485     function approve(address spender, uint256 amou
       nt) public virtual override returns (bool) {
486         _approve(msg.sender, spender, amount);
487         return true;
488     }
489
490     /**
491      * @dev See {IERC20-transferFrom}.
492      *
493      * Emits an {Approval} event indicating the up
       dated allowance. This is not
494      * required by the EIP. See the note at the be
       ginning of {ERC20}.
495      *
496      * Requirements:
497      *
498      * - `sender` and `recipient` cannot be the ze
       ro address.
499      * - `sender` must have a balance of at least
        `amount`.
500      * - the caller must have allowance for ``send
       er``'s tokens of at least
501      * `amount`.
502      */
503     function transferFrom(address sender, address
        recipient, uint256 amount) public virtual overrid
       e returns (bool) {
504         _transfer(sender, recipient, amount);
505         _approve(sender, msg.sender, _allowances[s
       ender][msg.sender]
506             .sub(amount));
507         return true;
508     }
509
510     /**
511      * @dev Atomically increases the allowance gra
       nted to `spender` by the caller.
512      *
513      * This is an alternative to {approve} that ca
       n be used as a mitigation for
514      * problems described in {IERC20-approve}.
515      *
```

```
     * 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(msg.sender, spender, _allowances[msg.sender][spender].add(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) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender]
            .sub(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);
```

```
566
567     _balances[sender] = _balances[sender].sub(amou
    nt);
568     _balances[recipient] = _balances[recipient].ad
    d(amount);
569     emit Transfer(sender, recipient, amount);
570   }
571
572     /** @dev Creates `amount` tokens and assigns t
    hem to `account`, increasing
573      * the total supply.
574      *
575      * Emits a {Transfer} event with `from` set to
    the zero address.
576      *
577      * Requirements:
578      *
579      * - `to` cannot be the zero address.
580      */
581     function _mint(address account_, uint256 ammou
    nt_) internal virtual {
582         require(account_ != address(0), "ERC20: mi
    nt to the zero address");
583         _beforeTokenTransfer(address( this ), acco
    unt_, ammount_);
584         _totalSupply = _totalSupply.add(ammount_);
585         _balances[account_] = _balances[account_].
    add(ammount_);
586         emit Transfer(address( 0 ), account_, ammo
    unt_);
587     }
588
589     /**
590      * @dev Destroys `amount` tokens from `account
    `, reducing the
591      * total supply.
592      *
593      * Emits a {Transfer} event with `to` set to t
    he zero address.
594      *
595      * Requirements:
596      *
597      * - `account` cannot be the zero address.
598      * - `account` must have at least `amount` tok
    ens.
599      */
600     function _burn(address account, uint256 amoun
    t) internal virtual {
601         require(account != address(0), "ERC20: bur
    n from the zero address");
602
603         _beforeTokenTransfer(account, address(0),
     amount);
604
605         _balances[account] = _balances[account].su
    b(amount);
606         _totalSupply = _totalSupply.sub(amount);
607         emit Transfer(account, address(0), amoun
    t);
608     }
609
610     /**
611      * @dev Sets `amount` as the allowance of `spe
    nder` over the `owner` s tokens.
612      *
613      * This internal function is equivalent to `ap
    prove`, and can be used to
614      * e.g. set automatic allowances for certain s
    ubsystems, etc.
```


```
566
567     _balances[sender] = _balances[sender].sub(amou
    nt);
568     _balances[recipient] = _balances[recipient].ad
    d(amount);
569     emit Transfer(sender, recipient, amount);
570   }
571
572     /** @dev Creates `amount` tokens and assigns t
    hem to `account`, increasing
573      * the total supply.
574      *
575      * Emits a {Transfer} event with `from` set to
    the zero address.
576      *
577      * Requirements:
578      *
579      * - `to` cannot be the zero address.
580      */
581     function _mint(address account_, uint256 ammou
    nt_) internal virtual {
582         require(account_ != address(0), "ERC20: mi
    nt to the zero address");
583         _beforeTokenTransfer(address( this ), acco
    unt_, ammount_);
584         _totalSupply = _totalSupply.add(ammount_);
585         _balances[account_] = _balances[account_].
    add(ammount_);
586         emit Transfer(address( 0 ), account_, ammo
    unt_);
587     }
588
589     /**
590      * @dev Destroys `amount` tokens from `account
    `, reducing the
591      * total supply.
592      *
593      * Emits a {Transfer} event with `to` set to t
    he zero address.
594      *
595      * Requirements:
596      *
597      * - `account` cannot be the zero address.
598      * - `account` must have at least `amount` tok
    ens.
599      */
600     function _burn(address account, uint256 amoun
    t) internal virtual {
601         require(account != address(0), "ERC20: bur
    n from the zero address");
602
603         _beforeTokenTransfer(account, address(0),
     amount);
604
605         _balances[account] = _balances[account].su
    b(amount);
606         _totalSupply = _totalSupply.sub(amount);
607         emit Transfer(account, address(0), amoun
    t);
608     }
609
610     /**
611      * @dev Sets `amount` as the allowance of `spe
    nder` over the `owner` s tokens.
612      *
613      * This internal function is equivalent to `ap
    prove`, and can be used to
614      * e.g. set automatic allowances for certain s
    ubsystems, 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 Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */

  /**
   * @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 { }
}

library Counters {
    using LowGasSafeMath for uint256;

    struct Counter {
        // This variable should never be directly accessed by users of the library: interactions must be restricted to
        // the library's function. As of Solidity v0.5.2, this cannot be enforced, though there is a proposal to add
        // this feature: see https://github.com/ethereum/solidity/issues/4637
        uint256 _value; // default: 0
    }

```

```
    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        // The {SafeMath} overflow check can be skipped here, see the comment at the top
        counter._value += 1;
    }

    function decrement(Counter storage counter) internal {
        counter._value = counter._value.sub(1);
    }
}

interface IERC2612Permit {
    /**
     * @dev Sets `amount` as the allowance of `spender` over `owner`'s tokens,
     * given `owner`'s signed approval.
     *
     * IMPORTANT: The same issues {IERC20-approve} has related to transaction
     * ordering also apply here.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     * - `deadline` must be a timestamp in the future.
     * - `v`, `r` and `s` must be a valid `secp256k1` signature from `owner`
     * over the EIP712-formatted function arguments.
     * - the signature must use ``owner``'s current nonce (see {nonces}).
     *
     * For more information on the signature format, see the
     * https://eips.ethereum.org/EIPS/eip-2612#specification[relevant EIP
     * section].
     */
    function permit(
        address owner,
        address spender,
        uint256 amount,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    /**
     * @dev Returns the current ERC2612 nonce for `owner`. This value must be
     * included whenever a signature is generated for {permit}.
     *
     * Every successful call to {permit} increases ``owner``'s nonce by one. This
```


```
    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        // The {SafeMath} overflow check can be skipped here, see the comment at the top
        counter._value += 1;
    }

    function decrement(Counter storage counter) internal {
        counter._value = counter._value.sub(1);
    }
}

interface IERC2612Permit {
    /**
     * @dev Sets `amount` as the allowance of `spender` over `owner`'s tokens,
     * given `owner`'s signed approval.
     *
     * IMPORTANT: The same issues {IERC20-approve} has related to transaction
     * ordering also apply here.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     * - `deadline` must be a timestamp in the future.
     * - `v`, `r` and `s` must be a valid `secp256k1` signature from `owner`
     * over the EIP712-formatted function arguments.
     * - the signature must use ``owner``'s current nonce (see {nonces}).
     *
     * For more information on the signature format, see the
     * https://eips.ethereum.org/EIPS/eip-2612#specification[relevant EIP
     * section].
     */
    function permit(
        address owner,
        address spender,
        uint256 amount,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    /**
     * @dev Returns the current ERC2612 nonce for `owner`. This value must be
     * included whenever a signature is generated for {permit}.
     *
     * Every successful call to {permit} increases ``owner``'s nonce by one. This
```

```
     * prevents a signature from being used multiple times.
     */
    function nonces(address owner) external view returns (uint256);
}

abstract contract ERC20Permit is ERC20, IERC2612Permit {
    using Counters for Counters.Counter;

    mapping(address => Counters.Counter) private _nonces;

    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;

    bytes32 public immutable DOMAIN_SEPARATOR;

    constructor() {

        uint256 chainID;
        assembly {
            chainID := chainid()
        }

        DOMAIN_SEPARATOR = keccak256(abi.encode(
            keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"),
            keccak256(bytes(name())),
            keccak256(bytes("1")), // Version
            chainID,
            address(this)
        ));
    }

    /**
     * @dev See {IERC2612Permit-permit}.
     *
     */
    function permit(
        address owner,
        address spender,
        uint256 amount,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public virtual override {
        require(block.timestamp <= deadline, "Permit: expired deadline");

        bytes32 hashStruct =
            keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, amount, _nonces[owner].current(), deadline));

        bytes32 _hash = keccak256(abi.encodePacked(uint16(0x1901), DOMAIN_SEPARATOR, hashStruct));

        address signer = ecrecover(_hash, v, r, s);
        require(signer != address(0) && signer == owner, "ERC20Permit: Invalid signature");

```

```
        _nonces[owner].increment();
        _approve(owner, spender, amount);
    }

    /**
     * @dev See {IERC2612Permit-nonces}.
     */
    function nonces(address owner) public view override returns (uint256) {
        return _nonces[owner].current();
    }
}

contract OwnableData {
    address public owner;
    address public pendingOwner;
}

contract Ownable is OwnableData {
    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /// @notice `owner` defaults to msg.sender on construction.
    constructor() {
        owner = msg.sender;
        emit OwnershipTransferred(address(0), msg.sender);
    }

    /// @notice Transfers ownership to `newOwner`. Either directly or claimable by the new pending owner.
    /// Can only be invoked by the current `owner`.
    /// @param newOwner Address of the new owner.
    /// @param direct True if `newOwner` should be set immediately. False if `newOwner` needs to use `claimOwnership`.
    /// @param renounce Allows the `newOwner` to be `address(0)` if `direct` and `renounce` is True. Has no effect otherwise.
    function transferOwnership(
        address newOwner,
        bool direct,
        bool renounce
    ) public onlyOwner {
        if (direct) {
            // Checks
            require(newOwner != address(0) || renounce, "Ownable: zero address");

            // Effects
            emit OwnershipTransferred(owner, newOwner);
            owner = newOwner;
            pendingOwner = address(0);
        } else {
            // Effects
            pendingOwner = newOwner;
        }
    }

    /// @notice Needs to be called by `pendingOwner` to claim ownership.
    function claimOwnership() public {
        address _pendingOwner = pendingOwner;

        // Checks
```

```
        require(msg.sender == _pendingOwner, "Ownable: caller != pending owner");

        // Effects
        emit OwnershipTransferred(owner, _pendingOwner);
        owner = _pendingOwner;
        pendingOwner = address(0);
    }

    /// @notice Only allows the `owner` to execute the function.
    modifier onlyOwner() {
        require(msg.sender == owner, "Ownable: caller is not the owner");
        _;
    }
}

contract MEMOries is ERC20Permit, Ownable {

    using LowGasSafeMath for uint256;

    modifier onlyStakingContract() {
        require( msg.sender == stakingContract, "OSC" );
        _;
    }

    address public stakingContract;
    address public initializer;

    event LogSupply(uint256 indexed epoch, uint256 timestamp, uint256 totalSupply );
    event LogRebase( uint256 indexed epoch, uint256 rebase, uint256 index );
    event LogStakingContractUpdated( address stakingContract );
    event LogSetIndex(uint256 indexed index );

    struct Rebase {
        uint epoch;
        uint rebase; // 18 decimals
        uint totalStakedBefore;
        uint totalStakedAfter;
        uint amountRebased;
        uint index;
        uint32 timeOccured;
    }
    Rebase[] public rebases;

    uint public INDEX;

    uint256 private constant MAX_UINT256 = ~uint256(0);
    uint256 private constant INITIAL_FRAGMENTS_SUPPLY = 5000000 * 10**9;

    // TOTAL_GONS is a multiple of INITIAL_FRAGMENTS_SUPPLY so that _gonsPerFragment is an integer.
    // Use the highest value that fits in a uint256 for max granularity.
    uint256 private constant TOTAL_GONS = MAX_UINT256 - (MAX_UINT256 % INITIAL_FRAGMENTS_SUPPLY);

    // MAX_SUPPLY = maximum integer < (sqrt(4*TOTAL_GONS + 1) - 1) / 2
    uint256 private constant MAX_SUPPLY = ~uint128(0);  // (2^128) - 1
```

```
        require(msg.sender == _pendingOwner, "Ownable: caller != pending owner");

        // Effects
        emit OwnershipTransferred(owner, _pendingOwner);
        owner = _pendingOwner;
        pendingOwner = address(0);
    }

    /// @notice Only allows the `owner` to execute the function.
    modifier onlyOwner() {
        require(msg.sender == owner, "Ownable: caller is not the owner");
        _;
    }
}

contract sMaia is ERC20Permit, Ownable {

    using LowGasSafeMath for uint256;

    modifier onlyStakingContract() {
        require( msg.sender == stakingContract, "OSC" );
        _;
    }

    address public stakingContract;
    address public initializer;

    event LogSupply(uint256 indexed epoch, uint256 timestamp, uint256 totalSupply );
    event LogRebase( uint256 indexed epoch, uint256 rebase, uint256 index );
    event LogStakingContractUpdated( address stakingContract );
    event LogSetIndex(uint256 indexed index );

    struct Rebase {
        uint epoch;
        uint rebase; // 18 decimals
        uint totalStakedBefore;
        uint totalStakedAfter;
        uint amountRebased;
        uint index;
        uint32 timeOccured;
    }
    Rebase[] public rebases;

    uint public INDEX;

    uint256 private constant MAX_UINT256 = ~uint256(0);
    uint256 private constant INITIAL_FRAGMENTS_SUPPLY = 5000000 * 10**9;

    // TOTAL_GONS is a multiple of INITIAL_FRAGMENTS_SUPPLY so that _gonsPerFragment is an integer.
    // Use the highest value that fits in a uint256 for max granularity.
    uint256 private constant TOTAL_GONS = MAX_UINT256 - (MAX_UINT256 % INITIAL_FRAGMENTS_SUPPLY);

    // MAX_SUPPLY = maximum integer < (sqrt(4*TOTAL_GONS + 1) - 1) / 2
    uint256 private constant MAX_SUPPLY = ~uint128(0);  // (2^128) - 1
```

```
    uint256 private _gonsPerFragment;
    mapping(address => uint256) private _gonBalances;

    mapping ( address => mapping ( address => uint256 ) ) private _allowedValue;

    constructor() ERC20("MEMOries", "MEMO", 9) ERC20Permit() {
        initializer = msg.sender;
        _totalSupply = INITIAL_FRAGMENTS_SUPPLY;
        _gonsPerFragment = TOTAL_GONS.div(_totalSupply);
    }

    function initialize( address stakingContract_ ) external returns ( bool ) {
        require( msg.sender == initializer, "NA" );
        require( stakingContract_ != address(0), "IA" );
        stakingContract = stakingContract_;
        _gonBalances[ stakingContract ] = TOTAL_GONS;

        emit Transfer( address(0x0), stakingContract, _totalSupply );
        emit LogStakingContractUpdated( stakingContract_ );

        initializer = address(0);
        return true;
    }

    function setIndex( uint _INDEX ) external onlyOwner() {
        require( INDEX == 0, "INZ");
        INDEX = gonsForBalance( _INDEX );
        emit LogSetIndex(INDEX);
    }

    /**
        @notice increases MEMOries supply to increase staking balances relative to profit_
        @param profit_ uint256
        @return uint256
     */
    function rebase( uint256 profit_, uint epoch_ ) public onlyStakingContract() returns ( uint256 ) {
        uint256 rebaseAmount;
        uint256 circulatingSupply_ = circulatingSupply();

        if ( profit_ == 0 ) {
            emit LogSupply( epoch_, block.timestamp, _totalSupply );
            emit LogRebase( epoch_, 0, index() );
            return _totalSupply;
        } else if ( circulatingSupply_ > 0 ){
            rebaseAmount = profit_.mul( _totalSupply ).div( circulatingSupply_ );
        } else {
            rebaseAmount = profit_;
        }

        _totalSupply = _totalSupply.add( rebaseAmount );
```

```
    uint256 private _gonsPerFragment;
    mapping(address => uint256) private _gonBalances;

    mapping ( address => mapping ( address => uint256 ) ) private _allowedValue;

    constructor() ERC20("Staked Maia", "MAIA", 9) ERC20Permit() {
        initializer = msg.sender;
        _totalSupply = INITIAL_FRAGMENTS_SUPPLY;
        _gonsPerFragment = TOTAL_GONS.div(_totalSupply);
    }

    function initialize( address stakingContract_ ) external returns ( bool ) {
        require( msg.sender == initializer, "NA" );
        require( stakingContract_ != address(0), "IA" );
        stakingContract = stakingContract_;
        _gonBalances[ stakingContract ] = TOTAL_GONS;

        emit Transfer( address(0x0), stakingContract, _totalSupply );
        emit LogStakingContractUpdated( stakingContract_ );

        initializer = address(0);
        return true;
    }

    function setIndex( uint _INDEX ) external onlyOwner() {
        require( INDEX == 0, "INZ");
        INDEX = gonsForBalance( _INDEX );
        emit LogSetIndex(INDEX);
    }

    /**
        @notice increases MEMOries supply to increase staking balances relative to profit_
        @param profit_ uint256
        @return uint256
     */
    function rebase( uint256 profit_, uint epoch_ ) public onlyStakingContract() returns ( uint256 ) {
        uint256 rebaseAmount;
        uint256 circulatingSupply_ = circulatingSupply();

        if ( profit_ == 0 ) {
            emit LogSupply( epoch_, block.timestamp, _totalSupply );
            emit LogRebase( epoch_, 0, index() );
            return _totalSupply;
        } else if ( circulatingSupply_ > 0 ){
            rebaseAmount = profit_.mul( _totalSupply ).div( circulatingSupply_ );
        } else {
            rebaseAmount = profit_;
        }

        _totalSupply = _totalSupply.add( rebaseAmount );
```

```
        if ( _totalSupply > MAX_SUPPLY ) {
            _totalSupply = MAX_SUPPLY;
        }

        _gonsPerFragment = TOTAL_GONS.div( _totalSupply );

        _storeRebase( circulatingSupply_, profit_, epoch_ );

        return _totalSupply;
    }

    /**
        @notice emits event with data about rebase
        @param previousCirculating_ uint
        @param profit_ uint
        @param epoch_ uint
        @return bool
     */
    function _storeRebase( uint previousCirculating_, uint profit_, uint epoch_ ) internal returns ( bool ) {
        uint rebasePercent = profit_.mul( 1e18 ).div( previousCirculating_ );

        rebases.push( Rebase ( {
            epoch: epoch_,
            rebase: rebasePercent, // 18 decimals
            totalStakedBefore: previousCirculating_,
            totalStakedAfter: circulatingSupply(),
            amountRebased: profit_,
            index: index(),
            timeOccured: uint32(block.timestamp)
        }));

        emit LogSupply( epoch_, block.timestamp, _totalSupply );
        emit LogRebase( epoch_, rebasePercent, index() );

        return true;
    }

    function balanceOf( address who ) public view override returns ( uint256 ) {
        return _gonBalances[ who ].div( _gonsPerFragment );
    }

    function gonsForBalance( uint amount ) public view returns ( uint ) {
        return amount.mul( _gonsPerFragment );
    }

    function balanceForGons( uint gons ) public view returns ( uint ) {
        return gons.div( _gonsPerFragment );
    }

    // Staking contract holds excess MEMOries
    function circulatingSupply() public view returns ( uint ) {
        return _totalSupply.sub( balanceOf( stakingContract ) );
    }

    function index() public view returns ( uint ) {
```

```
        return balanceForGons( INDEX );
    }

    function transfer( address to, uint256 value ) public override returns (bool) {
        uint256 gonValue = value.mul( _gonsPerFragment );
        _gonBalances[ msg.sender ] = _gonBalances[ msg.sender ].sub( gonValue );
        _gonBalances[ to ] = _gonBalances[ to ].add( gonValue );
        emit Transfer( msg.sender, to, value );
        return true;
    }

    function allowance( address owner_, address spender ) public view override returns ( uint256 ) {
        return _allowedValue[ owner_ ][ spender ];
    }

    function transferFrom( address from, address to, uint256 value ) public override returns ( bool ) {
       _allowedValue[ from ][ msg.sender ] = _allowedValue[ from ][ msg.sender ].sub( value );
       emit Approval( from, msg.sender,  _allowedValue[ from ][ msg.sender ] );

        uint256 gonValue = gonsForBalance( value );
        _gonBalances[ from ] = _gonBalances[from].sub( gonValue );
        _gonBalances[ to ] = _gonBalances[to].add( gonValue );
        emit Transfer( from, to, value );

        return true;
    }

    function approve( address spender, uint256 value ) public override returns (bool) {
         _allowedValue[ msg.sender ][ spender ] = value;
         emit Approval( msg.sender, spender, value );
         return true;
    }

    // What gets called in a permit
    function _approve( address owner, address spender, uint256 value ) internal override virtual {
        _allowedValue[owner][spender] = value;
        emit Approval( owner, spender, value );
    }

    function increaseAllowance( address spender, uint256 addedValue ) public override returns (bool) {
        _allowedValue[ msg.sender ][ spender ] = _allowedValue[ msg.sender ][ spender ].add( addedValue );
        emit Approval( msg.sender, spender, _allowedValue[ msg.sender ][ spender ] );
        return true;
    }

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

```
1034        uint256 oldValue = _allowedValue[ msg.send
    er ][ spender ];
1035        if (subtractedValue >= oldValue) {
1036            _allowedValue[ msg.sender ][ spender ]
    = 0;
1037        } else {
1038            _allowedValue[ msg.sender ][ spender ]
    = oldValue.sub( subtractedValue );
1039        }
1040        emit Approval( msg.sender, spender, _allow
    edValue[ msg.sender ][ spender ] );
1041        return true;
1042    }
1043 }
```


```
1034        uint256 oldValue = _allowedValue[ msg.send
    er ][ spender ];
1035        if (subtractedValue >= oldValue) {
1036            _allowedValue[ msg.sender ][ spender ]
    = 0;
1037        } else {
1038            _allowedValue[ msg.sender ][ spender ]
    = oldValue.sub( subtractedValue );
1039        }
1040        emit Approval( msg.sender, spender, _allow
    edValue[ msg.sender ][ spender ] );
1041        return true;
1042    }
1043 }
```