

```

1 /**
2  *Submitted for verification at Etherscan.io on 202
3  1-06-12
4  */
5  // SPDX-License-Identifier: MIT
6  pragma solidity 0.7.5;
7
8  /**
9   * @dev Interface of the ERC20 standard as defined
10  in the EIP.
11  */
12  interface IERC20 {
13      /**
14       * @dev Returns the amount of tokens in existen
15       ce.
16       */
17       function totalSupply() external view returns (u
18         int256);
19
20       /**
21        * @dev Returns the amount of tokens owned by `
22        account`.
23        */
24        function balanceOf(address account) external vi
25        ew returns (uint256);
26
27        /**
28         * @dev Moves `amount` tokens from the caller's
29         account to `recipient`.
30         */
31         * Returns a boolean value indicating whether t
32         he operation succeeded.
33         */
34         * Emits a {Transfer} event.
35         */
36         function transfer(address recipient, uint256 am
37         ount) external returns (bool);
38
39         /**
40          * @dev Returns the remaining number of tokens
41          that `spender` will be
42          * allowed to spend on behalf of `owner` throug
43          h {transferFrom}. This is
44          * zero by default.
45          *
46          * This value changes when {approve} or {transf
47          erFrom} are called.
48          */
49          function allowance(address owner, address spend
50          er) external view returns (uint256);
51
52          /**
53           * @dev Sets `amount` as the allowance of `spen
54           der` over the caller's tokens.
55           */
56           * Returns a boolean value indicating whether t
57           he operation succeeded.
58           */
59           * IMPORTANT: Beware that changing an allowance
60           with this method brings the risk
61           * that someone may use both the old and the ne
62           w allowance by unfortunate

```

```

1 /**
2  *Submitted for verification at Etherscan.io on 202
3  1-06-12
4  */
5  // SPDX-License-Identifier: MIT
6  pragma solidity 0.7.5;
7
8  /**
9   * @dev Interface of the ERC20 standard as defined
10  in the EIP.
11  */
12  interface IERC20 {
13      /**
14       * @dev Returns the amount of tokens in existen
15       ce.
16       */
17       function totalSupply() external view returns (u
18         int256);
19
20       /**
21        * @dev Returns the amount of tokens owned by `
22        account`.
23        */
24        function balanceOf(address account) external vi
25        ew returns (uint256);
26
27        /**
28         * @dev Moves `amount` tokens from the caller's
29         account to `recipient`.
30         */
31         * Returns a boolean value indicating whether t
32         he operation succeeded.
33         */
34         * Emits a {Transfer} event.
35         */
36         function transfer(address recipient, uint256 am
37         ount) external returns (bool);
38
39         /**
40          * @dev Returns the remaining number of tokens
41          that `spender` will be
42          * allowed to spend on behalf of `owner` throug
43          h {transferFrom}. This is
44          * zero by default.
45          *
46          * This value changes when {approve} or {transf
47          erFrom} are called.
48          */
49          function allowance(address owner, address spend
50          er) external view returns (uint256);
51
52          /**
53           * @dev Sets `amount` as the allowance of `spen
54           der` over the caller's tokens.
55           */
56           * Returns a boolean value indicating whether t
57           he operation succeeded.
58           */
59           * IMPORTANT: Beware that changing an allowance
60           with this method brings the risk
61           * that someone may use both the old and the ne
62           w allowance by unfortunate

```

```

47     * transaction ordering. One possible solution
    to mitigate this race
48     * condition is to first reduce the spender's a
    llowance to 0 and set the
49     * desired value afterwards:
50     * https://github.com/ethereum/EIPs/issues/20#i
    ssuecomment-263524729
51     *
52     * Emits an {Approval} event.
53     */
54     function approve(address spender, uint256 amoun
    t) external returns (bool);
55
56     /**
57     * @dev Moves `amount` tokens from `sender` to
    `recipient` using the
58     * allowance mechanism. `amount` is then deduct
    ed from the caller's
59     * allowance.
60     *
61     * Returns a boolean value indicating whether t
    he operation succeeded.
62     *
63     * Emits a {Transfer} event.
64     */
65     function transferFrom(address sender, address r
    ecipient, uint256 amount) external returns (bool);
66
67     /**
68     * @dev Emitted when `value` tokens are moved f
    rom one account (`from`) to
69     * another (`to`).
70     *
71     * Note that `value` may be zero.
72     */
73     event Transfer(address indexed from, address in
    dexed to, uint256 value);
74
75     /**
76     * @dev Emitted when the allowance of a `spende
    r` for an `owner` is set by
77     * a call to {approve}. `value` is the new allo
    wance.
78     */
79     event Approval(address indexed owner, address i
    ndexed spender, uint256 value);
80 }
81
82 library LowGasSafeMath {
83     /// @notice Returns x + y, reverts if sum overf
    lows uint256
84     /// @param x The augend
85     /// @param y The addend
86     /// @return z The sum of x and y
87     function add(uint256 x, uint256 y) internal pur
    e returns (uint256 z) {
88         require((z = x + y) >= x);
89     }
90
91     function add32(uint32 x, uint32 y) internal pur
    e returns (uint32 z) {
92         require((z = x + y) >= x);
93     }
94
95     /// @notice Returns x - y, reverts if underflow
    s
96     /// @param x The minuend
97     /// @param y The subtrahend
98     /// @return z The difference of x and y

```

```

47     * transaction ordering. One possible solution
    to mitigate this race
48     * condition is to first reduce the spender's a
    llowance to 0 and set the
49     * desired value afterwards:
50     * https://github.com/ethereum/EIPs/issues/20#i
    ssuecomment-263524729
51     *
52     * Emits an {Approval} event.
53     */
54     function approve(address spender, uint256 amoun
    t) external returns (bool);
55
56     /**
57     * @dev Moves `amount` tokens from `sender` to
    `recipient` using the
58     * allowance mechanism. `amount` is then deduct
    ed from the caller's
59     * allowance.
60     *
61     * Returns a boolean value indicating whether t
    he operation succeeded.
62     *
63     * Emits a {Transfer} event.
64     */
65     function transferFrom(address sender, address r
    ecipient, uint256 amount) external returns (bool);
66
67     /**
68     * @dev Emitted when `value` tokens are moved f
    rom one account (`from`) to
69     * another (`to`).
70     *
71     * Note that `value` may be zero.
72     */
73     event Transfer(address indexed from, address in
    dexed to, uint256 value);
74
75     /**
76     * @dev Emitted when the allowance of a `spende
    r` for an `owner` is set by
77     * a call to {approve}. `value` is the new allo
    wance.
78     */
79     event Approval(address indexed owner, address i
    ndexed spender, uint256 value);
80 }
81
82 library LowGasSafeMath {
83     /// @notice Returns x + y, reverts if sum overf
    lows uint256
84     /// @param x The augend
85     /// @param y The addend
86     /// @return z The sum of x and y
87     function add(uint256 x, uint256 y) internal pur
    e returns (uint256 z) {
88         require((z = x + y) >= x);
89     }
90
91     function add32(uint32 x, uint32 y) internal pur
    e returns (uint32 z) {
92         require((z = x + y) >= x);
93     }
94
95     /// @notice Returns x - y, reverts if underflow
    s
96     /// @param x The minuend
97     /// @param y The subtrahend
98     /// @return z The difference of x and y

```

```

99     function sub(uint256 x, uint256 y) internal pure
    returns (uint256 z) {
100         require((z = x - y) <= x);
101     }
102
103     function sub32(uint32 x, uint32 y) internal pure
    returns (uint32 z) {
104         require((z = x - y) <= x);
105     }
106
107     /// @notice Returns x * y, reverts if overflows
108     /// @param x The multiplicand
109     /// @param y The multiplier
110     /// @return z The product of x and y
111     function mul(uint256 x, uint256 y) internal pure
    returns (uint256 z) {
112         require(x == 0 || (z = x * y) / x == y);
113     }
114
115     /// @notice Returns x + y, reverts if overflows
    or underflows
116     /// @param x The augend
117     /// @param y The addend
118     /// @return z The sum of x and y
119     function add(int256 x, int256 y) internal pure
    returns (int256 z) {
120         require((z = x + y) >= x == (y >= 0));
121     }
122
123     /// @notice Returns x - y, reverts if overflows
    or underflows
124     /// @param x The minuend
125     /// @param y The subtrahend
126     /// @return z The difference of x and y
127     function sub(int256 x, int256 y) internal pure
    returns (int256 z) {
128         require((z = x - y) <= x == (y >= 0));
129     }
130
131     function div(uint256 x, uint256 y) internal pure
    returns(uint256 z){
132         require(y > 0);
133         z=x/y;
134     }
135 }
136
137 /**
138  * @dev Collection of functions related to the address type
139  */
140 library Address {
141     /**
142      * @dev Returns true if `account` is a contract.
143      *
144      * [IMPORTANT]
145      * ====
146      * It is unsafe to assume that an address for which this function returns
147      * false is an externally-owned account (EOA) and not a contract.
148      *
149      * Among others, `isContract` will return false for the following
150      * types of addresses:
151      *
152      * - an externally-owned account
153      * - a contract in construction

```

```

99     function sub(uint256 x, uint256 y) internal pure
    returns (uint256 z) {
100         require((z = x - y) <= x);
101     }
102
103     function sub32(uint32 x, uint32 y) internal pure
    returns (uint32 z) {
104         require((z = x - y) <= x);
105     }
106
107     /// @notice Returns x * y, reverts if overflows
108     /// @param x The multiplicand
109     /// @param y The multiplier
110     /// @return z The product of x and y
111     function mul(uint256 x, uint256 y) internal pure
    returns (uint256 z) {
112         require(x == 0 || (z = x * y) / x == y);
113     }
114
115     /// @notice Returns x + y, reverts if overflows
    or underflows
116     /// @param x The augend
117     /// @param y The addend
118     /// @return z The sum of x and y
119     function add(int256 x, int256 y) internal pure
    returns (int256 z) {
120         require((z = x + y) >= x == (y >= 0));
121     }
122
123     /// @notice Returns x - y, reverts if overflows
    or underflows
124     /// @param x The minuend
125     /// @param y The subtrahend
126     /// @return z The difference of x and y
127     function sub(int256 x, int256 y) internal pure
    returns (int256 z) {
128         require((z = x - y) <= x == (y >= 0));
129     }
130
131     function div(uint256 x, uint256 y) internal pure
    returns(uint256 z){
132         require(y > 0);
133         z=x/y;
134     }
135 }
136
137 /**
138  * @dev Collection of functions related to the address type
139  */
140 library Address {
141     /**
142      * @dev Returns true if `account` is a contract.
143      *
144      * [IMPORTANT]
145      * ====
146      * It is unsafe to assume that an address for which this function returns
147      * false is an externally-owned account (EOA) and not a contract.
148      *
149      * Among others, `isContract` will return false for the following
150      * types of addresses:
151      *
152      * - an externally-owned account
153      * - a contract in construction

```

```

154     * - an address where a contract will be creat
ed
155     * - an address where a contract lived, but wa
s destroyed
156     * ====
157     */
158     function isContract(address account) internal v
iew returns (bool) {
159         // This method relies in extcodesize, which
returns 0 for contracts in
160         // construction, since the code is only sto
red at the end of the
161         // constructor execution.
162
163         uint256 size;
164         // solhint-disable-next-line no-inline-asse
mbly
165         assembly { size := extcodesize(account) }
166         return size > 0;
167     }
168
169     /**
170     * @dev Replacement for Solidity's `transfer`:
sends `amount` wei to
171     * `recipient`, forwarding all available gas an
d reverting on errors.
172     *
173     * https://eips.ethereum.org/EIPS/eip-1884[EIP1
884] increases the gas cost
174     * of certain opcodes, possibly making contract
s go over the 2300 gas limit
175     * imposed by `transfer`, making them unable to
receive funds via
176     * `transfer`. {sendValue} removes this limitat
ion.
177     *
178     * https://diligence.consensys.net/posts/2019/0
9/stop-using-soliditys-transfer-now/[Learn more].
179     *
180     * IMPORTANT: because control is transferred to
`recipient`, care must be
181     * taken to not create reentrancy vulnerabiliti
es. Consider using
182     * {ReentrancyGuard} or the
183     * https://solidity.readthedocs.io/en/v0.5.11/s
ecurity-considerations.html#use-the-checks-effects-
interactions-pattern[checks-effects-interactions pa
ttern].
184     */
185     function sendValue(address payable recipient, u
int256 amount) internal {
186         require(address(this).balance >= amount, "A
ddress: insufficient balance");
187
188         // solhint-disable-next-line avoid-low-leve
l-calls, avoid-call-value
189         (bool success, ) = recipient.call{ value: a
mount }("");
190         require(success, "Address: unable to send v
alue, recipient may have reverted");
191     }
192
193     /**
194     * @dev Performs a Solidity function call using
a low level `call`. A
195     * plain `call` is an unsafe replacement for a f
unction call: use this
196     * function instead.

```

```

154     * - an address where a contract will be creat
ed
155     * - an address where a contract lived, but wa
s destroyed
156     * ====
157     */
158     function isContract(address account) internal v
iew returns (bool) {
159         // This method relies in extcodesize, which
returns 0 for contracts in
160         // construction, since the code is only sto
red at the end of the
161         // constructor execution.
162
163         uint256 size;
164         // solhint-disable-next-line no-inline-asse
mbly
165         assembly { size := extcodesize(account) }
166         return size > 0;
167     }
168
169     /**
170     * @dev Replacement for Solidity's `transfer`:
sends `amount` wei to
171     * `recipient`, forwarding all available gas an
d reverting on errors.
172     *
173     * https://eips.ethereum.org/EIPS/eip-1884[EIP1
884] increases the gas cost
174     * of certain opcodes, possibly making contract
s go over the 2300 gas limit
175     * imposed by `transfer`, making them unable to
receive funds via
176     * `transfer`. {sendValue} removes this limitat
ion.
177     *
178     * https://diligence.consensys.net/posts/2019/0
9/stop-using-soliditys-transfer-now/[Learn more].
179     *
180     * IMPORTANT: because control is transferred to
`recipient`, care must be
181     * taken to not create reentrancy vulnerabiliti
es. Consider using
182     * {ReentrancyGuard} or the
183     * https://solidity.readthedocs.io/en/v0.5.11/s
ecurity-considerations.html#use-the-checks-effects-
interactions-pattern[checks-effects-interactions pa
ttern].
184     */
185     function sendValue(address payable recipient, u
int256 amount) internal {
186         require(address(this).balance >= amount, "A
ddress: insufficient balance");
187
188         // solhint-disable-next-line avoid-low-leve
l-calls, avoid-call-value
189         (bool success, ) = recipient.call{ value: a
mount }("");
190         require(success, "Address: unable to send v
alue, recipient may have reverted");
191     }
192
193     /**
194     * @dev Performs a Solidity function call using
a low level `call`. A
195     * plain `call` is an unsafe replacement for a f
unction call: use this
196     * function instead.

```

```

197     *
198     * If `target` reverts with a revert reason, it
    is bubbled up by this
199     * function (like regular Solidity function cal
    ls).
200     *
201     * Returns the raw returned data. To convert to
    the expected return value,
202     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
203     *
204     * Requirements:
205     *
206     * - `target` must be a contract.
207     * - calling `target` with `data` must not revert.
208     *
209     * _Available since v3.1._
210     */
211     function functionCall(address target, bytes memory data) internal returns (bytes memory) {
212         return functionCall(target, data, "Address: low-level call failed");
213     }
214
215     /**
216     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
217     * `errorMessage` as a fallback revert reason when `target` reverts.
218     *
219     * _Available since v3.1._
220     */
221     function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {
222         return _functionCallWithValue(target, data, 0, errorMessage);
223     }
224
225     /**
226     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
227     * but also transferring `value` wei to `target`.
228     *
229     * Requirements:
230     *
231     * - the calling contract must have an ETH balance of at least `value`.
232     * - the called Solidity function must be `payable`.
233     *
234     * _Available since v3.1._
235     */
236     function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
237         return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
238     }
239
240     /**

```

```

197     *
198     * If `target` reverts with a revert reason, it
    is bubbled up by this
199     * function (like regular Solidity function call
    s).
200     *
201     * Returns the raw returned data. To convert to
    the expected return value,
202     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
203     *
204     * Requirements:
205     *
206     * - `target` must be a contract.
207     * - calling `target` with `data` must not revert.
208     *
209     * _Available since v3.1._
210     */
211     function functionCall(address target, bytes memory data) internal returns (bytes memory) {
212         return functionCall(target, data, "Address: low-level call failed");
213     }
214
215     /**
216     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
217     * `errorMessage` as a fallback revert reason when `target` reverts.
218     *
219     * _Available since v3.1._
220     */
221     function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {
222         return _functionCallWithValue(target, data, 0, errorMessage);
223     }
224
225     /**
226     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
227     * but also transferring `value` wei to `target`.
228     *
229     * Requirements:
230     *
231     * - the calling contract must have an ETH balance of at least `value`.
232     * - the called Solidity function must be `payable`.
233     *
234     * _Available since v3.1._
235     */
236     function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
237         return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
238     }
239
240     /**

```

```

241     * @dev Same as {xref-Address-functionCallWithValue
    alue-address-bytes-uint256-}[`functionCallWithValue
    `], but
242     * with `errorMessage` as a fallback revert rea
    son when `target` reverts.
243     *
244     * _Available since v3.1._
245     */
246     function functionCallWithValue(address target,
    bytes memory data, uint256 value, string memory er
    rorMessage) internal returns (bytes memory) {
247         require(address(this).balance >= value, "Ad
    dress: insufficient balance for call");
248         return _functionCallWithValue(target, data,
    value, errorMessage);
249     }
250
251     function _functionCallWithValue(address target,
    bytes memory data, uint256 weiValue, string memory
    errorMessage) private returns (bytes memory) {
252         require(isContract(target), "Address: call
    to non-contract");
253
254         // solhint-disable-next-line avoid-low-leve
    l-calls
255         (bool success, bytes memory returndata) = t
    arget.call{ value: weiValue }(data);
256         if (success) {
257             return returndata;
258         } else {
259             // Look for revert reason and bubble it
    up if present
260             if (returndata.length > 0) {
261                 // The easiest way to bubble the re
    vert reason is using memory via assembly
262
263                 // solhint-disable-next-line no-inl
    ine-assembly
264                 assembly {
265                     let returndata_size := mload(re
    turndata)
266                     revert(add(32, returndata), ret
    urndata_size)
267                 }
268             } else {
269                 revert(errorMessage);
270             }
271         }
272     }
273 }
274
275 /**
276  * @dev Implementation of the {IERC20} interface.
277  *
278  * This implementation is agnostic to the way token
    s are created. This means
279  * that a supply mechanism has to be added in a der
    ived contract using {_mint}.
280  * For a generic mechanism see {ERC20PresetMinterPa
    user}.
281  *
282  * TIP: For a detailed writeup see our guide
283  * https://forum.zeppelin.solutions/t/how-to-implem
    ent-erc20-supply-mechanisms/226[How
284  * to implement supply mechanisms].
285  *
286  * We have followed general OpenZeppelin guideline
    s: functions revert instead

```

```

241     * @dev Same as {xref-Address-functionCallWithValue
    alue-address-bytes-uint256-}[`functionCallWithValue
    `], but
242     * with `errorMessage` as a fallback revert rea
    son when `target` reverts.
243     *
244     * _Available since v3.1._
245     */
246     function functionCallWithValue(address target,
    bytes memory data, uint256 value, string memory er
    rorMessage) internal returns (bytes memory) {
247         require(address(this).balance >= value, "Ad
    dress: insufficient balance for call");
248         return _functionCallWithValue(target, data,
    value, errorMessage);
249     }
250
251     function _functionCallWithValue(address target,
    bytes memory data, uint256 weiValue, string memory
    errorMessage) private returns (bytes memory) {
252         require(isContract(target), "Address: call
    to non-contract");
253
254         // solhint-disable-next-line avoid-low-leve
    l-calls
255         (bool success, bytes memory returndata) = t
    arget.call{ value: weiValue }(data);
256         if (success) {
257             return returndata;
258         } else {
259             // Look for revert reason and bubble it
    up if present
260             if (returndata.length > 0) {
261                 // The easiest way to bubble the re
    vert reason is using memory via assembly
262
263                 // solhint-disable-next-line no-inl
    ine-assembly
264                 assembly {
265                     let returndata_size := mload(re
    turndata)
266                     revert(add(32, returndata), ret
    urndata_size)
267                 }
268             } else {
269                 revert(errorMessage);
270             }
271         }
272     }
273 }
274
275 /**
276  * @dev Implementation of the {IERC20} interface.
277  *
278  * This implementation is agnostic to the way token
    s are created. This means
279  * that a supply mechanism has to be added in a der
    ived contract using {_mint}.
280  * For a generic mechanism see {ERC20PresetMinterPa
    user}.
281  *
282  * TIP: For a detailed writeup see our guide
283  * https://forum.zeppelin.solutions/t/how-to-implem
    ent-erc20-supply-mechanisms/226[How
284  * to implement supply mechanisms].
285  *
286  * We have followed general OpenZeppelin guideline
    s: functions revert instead

```

```

287 * of returning `false` on failure. This behavior i
s nonetheless conventional
288 * and does not conflict with the expectations of E
RC20 applications.
289 *
290 * Additionally, an {Approval} event is emitted on
calls to {transferFrom}.
291 * This allows applications to reconstruct the allo
wance for all accounts just
292 * by listening to said events. Other implementatio
ns of the EIP may not emit
293 * these events, as it isn't required by the specif
ication.
294 *
295 * Finally, the non-standard {decreaseAllowance} an
d {increaseAllowance}
296 * functions have been added to mitigate the well-k
nown issues around setting
297 * allowances. See {IERC20-approve}.
298 */
299 contract ERC20 is IERC20 {
300     using LowGasSafeMath for uint256;
301
302     mapping (address => uint256) private _balances;
303
304     mapping (address => mapping (address => uint25
6)) private _allowances;
305
306     uint256 private _totalSupply;
307
308     string private _name;
309     string private _symbol;
310     uint8 public immutable decimals;
311
312     /**
313      * @dev Sets the values for {name} and {symbo
l}, initializes {decimals} with
314      * a default value of 18.
315      *
316      * To select a different value for {decimals},
use {_setupDecimals}.
317      *
318      * All three of these values are immutable: the
y can only be set once during
319      * construction.
320      */
321     constructor (string memory name, string memory
symbol) {
322         _name = name;
323         _symbol = symbol;
324         decimals = 18;
325     }
326
327     /**
328      * @dev Returns the name of the token.
329      */
330     function name() public view returns (string mem
ory) {
331         return _name;
332     }
333
334     /**
335      * @dev Returns the symbol of the token, usuall
y a shorter version of the
336      * name.
337      */
338     function symbol() public view returns (string m
emory) {
339         return _symbol;

```

```

287 * of returning `false` on failure. This behavior i
s nonetheless conventional
288 * and does not conflict with the expectations of E
RC20 applications.
289 *
290 * Additionally, an {Approval} event is emitted on
calls to {transferFrom}.
291 * This allows applications to reconstruct the allo
wance for all accounts just
292 * by listening to said events. Other implementatio
ns of the EIP may not emit
293 * these events, as it isn't required by the specif
ication.
294 *
295 * Finally, the non-standard {decreaseAllowance} an
d {increaseAllowance}
296 * functions have been added to mitigate the well-k
nown issues around setting
297 * allowances. See {IERC20-approve}.
298 */
299 contract ERC20 is IERC20 {
300     using LowGasSafeMath for uint256;
301
302     mapping (address => uint256) private _balances;
303
304     mapping (address => mapping (address => uint25
6)) private _allowances;
305
306     uint256 private _totalSupply;
307
308     string private _name;
309     string private _symbol;
310     uint8 public immutable decimals;
311
312     /**
313      * @dev Sets the values for {name} and {symbo
l}, initializes {decimals} with
314      * a default value of 18.
315      *
316      * To select a different value for {decimals},
use {_setupDecimals}.
317      *
318      * All three of these values are immutable: the
y can only be set once during
319      * construction.
320      */
321     constructor (string memory name, string memory
symbol) {
322         _name = name;
323         _symbol = symbol;
324         decimals = 18;
325     }
326
327     /**
328      * @dev Returns the name of the token.
329      */
330     function name() public view returns (string mem
ory) {
331         return _name;
332     }
333
334     /**
335      * @dev Returns the symbol of the token, usuall
y a shorter version of the
336      * name.
337      */
338     function symbol() public view returns (string m
emory) {
339         return _symbol;

```

```

340     }
341
342     /**
343      * @dev See {IERC20-totalSupply}.
344      */
345     function totalSupply() public view override returns (uint256) {
346         return _totalSupply;
347     }
348
349     /**
350      * @dev See {IERC20-balanceOf}.
351      */
352     function balanceOf(address account) public view override returns (uint256) {
353         return _balances[account];
354     }
355
356     /**
357      * @dev See {IERC20-transfer}.
358      *
359      * Requirements:
360      *
361      * - `recipient` cannot be the zero address.
362      * - the caller must have a balance of at least `amount`.
363      */
364     function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
365         _transfer(msg.sender, recipient, amount);
366         return true;
367     }
368
369     /**
370      * @dev See {IERC20-allowance}.
371      */
372     function allowance(address owner, address spender) public view virtual override returns (uint256) {
373         return _allowances[owner][spender];
374     }
375
376     /**
377      * @dev See {IERC20-approve}.
378      *
379      * Requirements:
380      *
381      * - `spender` cannot be the zero address.
382      */
383     function approve(address spender, uint256 amount) public virtual override returns (bool) {
384         _approve(msg.sender, spender, amount);
385         return true;
386     }
387
388     /**
389      * @dev See {IERC20-transferFrom}.
390      *
391      * Emits an {Approval} event indicating the updated allowance. This is not
392      * required by the EIP. See the note at the beginning of {ERC20};
393      *
394      * Requirements:
395      *
396      * - `sender` and `recipient` cannot be the zero address.
397      * - `sender` must have a balance of at least `amount`.

```

```

340     }
341
342     /**
343      * @dev See {IERC20-totalSupply}.
344      */
345     function totalSupply() public view override returns (uint256) {
346         return _totalSupply;
347     }
348
349     /**
350      * @dev See {IERC20-balanceOf}.
351      */
352     function balanceOf(address account) public view override returns (uint256) {
353         return _balances[account];
354     }
355
356     /**
357      * @dev See {IERC20-transfer}.
358      *
359      * Requirements:
360      *
361      * - `recipient` cannot be the zero address.
362      * - the caller must have a balance of at least `amount`.
363      */
364     function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
365         _transfer(msg.sender, recipient, amount);
366         return true;
367     }
368
369     /**
370      * @dev See {IERC20-allowance}.
371      */
372     function allowance(address owner, address spender) public view virtual override returns (uint256) {
373         return _allowances[owner][spender];
374     }
375
376     /**
377      * @dev See {IERC20-approve}.
378      *
379      * Requirements:
380      *
381      * - `spender` cannot be the zero address.
382      */
383     function approve(address spender, uint256 amount) public virtual override returns (bool) {
384         _approve(msg.sender, spender, amount);
385         return true;
386     }
387
388     /**
389      * @dev See {IERC20-transferFrom}.
390      *
391      * Emits an {Approval} event indicating the updated allowance. This is not
392      * required by the EIP. See the note at the beginning of {ERC20};
393      *
394      * Requirements:
395      *
396      * - `sender` and `recipient` cannot be the zero address.
397      * - `sender` must have a balance of at least `amount`.

```



```

397     * - the caller must have allowance for ``sende
r``s tokens of at least
398     * `amount`.
399     */
400     function transferFrom(address sender, address r
ecipient, uint256 amount) public virtual override r
eturns (bool) {
401         _transfer(sender, recipient, amount);
402         _approve(sender, msg.sender, _allowances[se
nder][msg.sender].sub(amount));
403         return true;
404     }
405
406     /**
407     * @dev Atomically increases the allowance gran
ted to `spender` by the caller.
408     *
409     * This is an alternative to {approve} that can
be used as a mitigation for
410     * problems described in {IERC20-approve}.
411     *
412     * Emits an {Approval} event indicating the upd
ated allowance.
413     *
414     * Requirements:
415     *
416     * - `spender` cannot be the zero address.
417     */
418     function increaseAllowance(address spender, uin
t256 addedValue) public virtual returns (bool) {
419         _approve(msg.sender, spender, _allowances[m
sg.sender][spender].add(addedValue));
420         return true;
421     }
422
423     /**
424     * @dev Atomically decreases the allowance gran
ted to `spender` by the caller.
425     *
426     * This is an alternative to {approve} that can
be used as a mitigation for
427     * problems described in {IERC20-approve}.
428     *
429     * Emits an {Approval} event indicating the upd
ated allowance.
430     *
431     * Requirements:
432     *
433     * - `spender` cannot be the zero address.
434     * - `spender` must have allowance for the call
er of at least
435     * `subtractedValue`.
436     */
437     function decreaseAllowance(address spender, uin
t256 subtractedValue) public virtual returns (bool)
{
438         _approve(msg.sender, spender, _allowances[m
sg.sender][spender].sub(subtractedValue));
439         return true;
440     }
441
442     /**
443     * @dev Moves tokens `amount` from `sender` to
`recipient`.
444     *
445     * This is internal function is equivalent to
{transfer}, and can be used to

```

```

397     * - the caller must have allowance for ``sende
r``s tokens of at least
398     * `amount`.
399     */
400     function transferFrom(address sender, address r
ecipient, uint256 amount) public virtual override r
eturns (bool) {
401         _transfer(sender, recipient, amount);
402         _approve(sender, msg.sender, _allowances[se
nder][msg.sender].sub(amount));
403         return true;
404     }
405
406     /**
407     * @dev Atomically increases the allowance gran
ted to `spender` by the caller.
408     *
409     * This is an alternative to {approve} that can
be used as a mitigation for
410     * problems described in {IERC20-approve}.
411     *
412     * Emits an {Approval} event indicating the upd
ated allowance.
413     *
414     * Requirements:
415     *
416     * - `spender` cannot be the zero address.
417     */
418     function increaseAllowance(address spender, uin
t256 addedValue) public virtual returns (bool) {
419         _approve(msg.sender, spender, _allowances[m
sg.sender][spender].add(addedValue));
420         return true;
421     }
422
423     /**
424     * @dev Atomically decreases the allowance gran
ted to `spender` by the caller.
425     *
426     * This is an alternative to {approve} that can
be used as a mitigation for
427     * problems described in {IERC20-approve}.
428     *
429     * Emits an {Approval} event indicating the upd
ated allowance.
430     *
431     * Requirements:
432     *
433     * - `spender` cannot be the zero address.
434     * - `spender` must have allowance for the call
er of at least
435     * `subtractedValue`.
436     */
437     function decreaseAllowance(address spender, uin
t256 subtractedValue) public virtual returns (bool)
{
438         _approve(msg.sender, spender, _allowances[m
sg.sender][spender].sub(subtractedValue));
439         return true;
440     }
441
442     /**
443     * @dev Moves tokens `amount` from `sender` to
`recipient`.
444     *
445     * This is internal function is equivalent to
{transfer}, and can be used to

```

```

446     * e.g. implement automatic token fees, slashing
      mechanisms, etc.
447     *
448     * Emits a {Transfer} event.
449     *
450     * Requirements:
451     *
452     * - `sender` cannot be the zero address.
453     * - `recipient` cannot be the zero address.
454     * - `sender` must have a balance of at least `
amount`.
455     */
456     function _transfer(address sender, address reci
      pient, uint256 amount) internal virtual {
457         require(sender != address(0), "ERC20: trans
      fer from the zero address");
458         require(recipient != address(0), "ERC20: tr
      ansfer to the zero address");
459
460         _beforeTokenTransfer(sender, recipient, amo
      unt);
461
462         _balances[sender] = _balances[sender].sub(a
      mount);
463         _balances[recipient] = _balances[recipien
      t].add(amount);
464         emit Transfer(sender, recipient, amount);
465     }
466
467     /** @dev Creates `amount` tokens and assigns th
      em to `account`, increasing
468     * the total supply.
469     *
470     * Emits a {Transfer} event with `from` set to
      the zero address.
471     *
472     * Requirements
473     *
474     * - `to` cannot be the zero address.
475     */
476     function _mint(address account, uint256 amount)
      internal virtual {
477         require(account != address(0), "ERC20: mint
      to the zero address");
478
479         _beforeTokenTransfer(address(0), account, a
      mount);
480
481         _totalSupply = _totalSupply.add(amount);
482         _balances[account] = _balances[account].add
      (amount);
483         emit Transfer(address(0), account, amount);
484     }
485
486     /**
487     * @dev Destroys `amount` tokens from `account
      `, reducing the
488     * total supply.
489     *
490     * Emits a {Transfer} event with `to` set to th
      e zero address.
491     *
492     * Requirements
493     *
494     * - `account` cannot be the zero address.
495     * - `account` must have at least `amount` toke
      ns.
496     */

```

```

446     * e.g. implement automatic token fees, slashing
      mechanisms, etc.
447     *
448     * Emits a {Transfer} event.
449     *
450     * Requirements:
451     *
452     * - `sender` cannot be the zero address.
453     * - `recipient` cannot be the zero address.
454     * - `sender` must have a balance of at least `
amount`.
455     */
456     function _transfer(address sender, address reci
      pient, uint256 amount) internal virtual {
457         require(sender != address(0), "ERC20: trans
      fer from the zero address");
458         require(recipient != address(0), "ERC20: tr
      ansfer to the zero address");
459
460         _beforeTokenTransfer(sender, recipient, amo
      unt);
461
462         _balances[sender] = _balances[sender].sub(a
      mount);
463         _balances[recipient] = _balances[recipien
      t].add(amount);
464         emit Transfer(sender, recipient, amount);
465     }
466
467     /** @dev Creates `amount` tokens and assigns th
      em to `account`, increasing
468     * the total supply.
469     *
470     * Emits a {Transfer} event with `from` set to
      the zero address.
471     *
472     * Requirements
473     *
474     * - `to` cannot be the zero address.
475     */
476     function _mint(address account, uint256 amount)
      internal virtual {
477         require(account != address(0), "ERC20: mint
      to the zero address");
478
479         _beforeTokenTransfer(address(0), account, a
      mount);
480
481         _totalSupply = _totalSupply.add(amount);
482         _balances[account] = _balances[account].add
      (amount);
483         emit Transfer(address(0), account, amount);
484     }
485
486     /**
487     * @dev Destroys `amount` tokens from `account
      `, reducing the
488     * total supply.
489     *
490     * Emits a {Transfer} event with `to` set to th
      e zero address.
491     *
492     * Requirements
493     *
494     * - `account` cannot be the zero address.
495     * - `account` must have at least `amount` toke
      ns.
496     */

```

```

497     function _burn(address account, uint256 amount)
internal virtual {
498         require(account != address(0), "ERC20: burn
from the zero address");
499
500         _beforeTokenTransfer(account, address(0), a
mount);
501
502         _balances[account] = _balances[account].sub
(amount);
503         _totalSupply = _totalSupply.sub(amount);
504         emit Transfer(account, address(0), amount);
505     }
506
507     /**
508      * @dev Sets `amount` as the allowance of `spen
der` over the `owner`'s tokens.
509      *
510      * This internal function is equivalent to `app
rove`, and can be used to
511      * e.g. set automatic allowances for certain su
bsystems, etc.
512      *
513      * Emits an {Approval} event.
514      *
515      * Requirements:
516      *
517      * - `owner` cannot be the zero address.
518      * - `spender` cannot be the zero address.
519      */
520     function _approve(address owner, address spende
r, uint256 amount) internal virtual {
521         require(owner != address(0), "ERC20: approv
e from the zero address");
522         require(spender != address(0), "ERC20: appr
ove to the zero address");
523
524         _allowances[owner][spender] = amount;
525         emit Approval(owner, spender, amount);
526     }
527
528     /**
529      * @dev Hook that is called before any transfer
of tokens. This includes
530      * minting and burning.
531      *
532      * Calling conditions:
533      *
534      * - when `from` and `to` are both non-zero, `a
mount` of `from`'s tokens
535      *   will be transferred to `to`.
536      * - when `from` is zero, `amount` tokens will
be minted for `to`.
537      * - when `to` is zero, `amount` of `from`'s
tokens will be burned.
538      * - `from` and `to` are never both zero.
539      *
540      * To learn more about hooks, head to xref:R00
T:extending-contracts.adoc#using-hooks[Using Hook
s].
541      */
542     function _beforeTokenTransfer(address from, add
ress to, uint256 amount) internal virtual { }
543 }
544
545 /**
546  * @title SafeERC20
547  * @dev Wrappers around ERC20 operations that throw
on failure (when the token

```

```

497     function _burn(address account, uint256 amount)
internal virtual {
498         require(account != address(0), "ERC20: burn
from the zero address");
499
500         _beforeTokenTransfer(account, address(0), a
mount);
501
502         _balances[account] = _balances[account].sub
(amount);
503         _totalSupply = _totalSupply.sub(amount);
504         emit Transfer(account, address(0), amount);
505     }
506
507     /**
508      * @dev Sets `amount` as the allowance of `spen
der` over the `owner`'s tokens.
509      *
510      * This internal function is equivalent to `app
rove`, and can be used to
511      * e.g. set automatic allowances for certain su
bsystems, etc.
512      *
513      * Emits an {Approval} event.
514      *
515      * Requirements:
516      *
517      * - `owner` cannot be the zero address.
518      * - `spender` cannot be the zero address.
519      */
520     function _approve(address owner, address spende
r, uint256 amount) internal virtual {
521         require(owner != address(0), "ERC20: approv
e from the zero address");
522         require(spender != address(0), "ERC20: appr
ove to the zero address");
523
524         _allowances[owner][spender] = amount;
525         emit Approval(owner, spender, amount);
526     }
527
528     /**
529      * @dev Hook that is called before any transfer
of tokens. This includes
530      * minting and burning.
531      *
532      * Calling conditions:
533      *
534      * - when `from` and `to` are both non-zero, `a
mount` of `from`'s tokens
535      *   will be transferred to `to`.
536      * - when `from` is zero, `amount` tokens will
be minted for `to`.
537      * - when `to` is zero, `amount` of `from`'s
tokens will be burned.
538      * - `from` and `to` are never both zero.
539      *
540      * To learn more about hooks, head to xref:R00
T:extending-contracts.adoc#using-hooks[Using Hook
s].
541      */
542     function _beforeTokenTransfer(address from, add
ress to, uint256 amount) internal virtual { }
543 }
544
545 /**
546  * @title SafeERC20
547  * @dev Wrappers around ERC20 operations that throw
on failure (when the token

```

```

548 * contract returns false). Tokens that return no v
    alue (and instead revert or
549 * throw on failure) are also supported, non-revert
    ing calls are assumed to be
550 * successful.
551 * To use this library you can add a `using SafeERC
    20 for IERC20;` statement to your contract,
552 * which allows you to call the safe operations as
    `token.safeTransfer(...)` , etc.
553 */
554 library SafeERC20 {
555     using LowGasSafeMath for uint256;
556     using Address for address;
557
558     function safeTransfer(IERC20 token, address to,
    uint256 value) internal {
559         _callOptionalReturn(token, abi.encodeWithSe
    lector(token.transfer.selector, to, value));
560     }
561
562     function safeTransferFrom(IERC20 token, address
    from, address to, uint256 value) internal {
563         _callOptionalReturn(token, abi.encodeWithSe
    lector(token.transferFrom.selector, from, to, valu
    e));
564     }
565
566     /**
567      * @dev Deprecated. This function has issues si
    milar to the ones found in
568      * {IERC20-approve}, and its usage is discourag
    ed.
569      *
570      * Whenever possible, use {safeIncreaseAllowanc
    e} and
571      * {safeDecreaseAllowance} instead.
572      */
573     function safeApprove(IERC20 token, address spen
    der, uint256 value) internal {
574         // safeApprove should only be called when s
    etting an initial allowance,
575         // or when resetting it to zero. To increas
    e and decrease it, use
576         // 'safeIncreaseAllowance' and 'safeDecreas
    eAllowance'
577         // solhint-disable-next-line max-line-lengt
    h
578         require((value == 0) || (token.allowance(ad
    dress(this), spender) == 0),
579             "SafeERC20: approve from non-zero to no
    n-zero allowance"
580         );
581         _callOptionalReturn(token, abi.encodeWithSe
    lector(token.approve.selector, spender, value));
582     }
583
584     function safeIncreaseAllowance(IERC20 token, ad
    dress spender, uint256 value) internal {
585         uint256 newAllowance = token.allowance(addr
    ess(this), spender).add(value);
586         _callOptionalReturn(token, abi.encodeWithSe
    lector(token.approve.selector, spender, newAllowanc
    e));
587     }
588
589     function safeDecreaseAllowance(IERC20 token, ad
    dress spender, uint256 value) internal {

```

```

548 * contract returns false). Tokens that return no v
    alue (and instead revert or
549 * throw on failure) are also supported, non-revert
    ing calls are assumed to be
550 * successful.
551 * To use this library you can add a `using SafeERC
    20 for IERC20;` statement to your contract,
552 * which allows you to call the safe operations as
    `token.safeTransfer(...)` , etc.
553 */
554 library SafeERC20 {
555     using LowGasSafeMath for uint256;
556     using Address for address;
557
558     function safeTransfer(IERC20 token, address to,
    uint256 value) internal {
559         _callOptionalReturn(token, abi.encodeWithSe
    lector(token.transfer.selector, to, value));
560     }
561
562     function safeTransferFrom(IERC20 token, address
    from, address to, uint256 value) internal {
563         _callOptionalReturn(token, abi.encodeWithSe
    lector(token.transferFrom.selector, from, to, valu
    e));
564     }
565
566     /**
567      * @dev Deprecated. This function has issues si
    milar to the ones found in
568      * {IERC20-approve}, and its usage is discourag
    ed.
569      *
570      * Whenever possible, use {safeIncreaseAllowanc
    e} and
571      * {safeDecreaseAllowance} instead.
572      */
573     function safeApprove(IERC20 token, address spen
    der, uint256 value) internal {
574         // safeApprove should only be called when s
    etting an initial allowance,
575         // or when resetting it to zero. To increas
    e and decrease it, use
576         // 'safeIncreaseAllowance' and 'safeDecreas
    eAllowance'
577         // solhint-disable-next-line max-line-lengt
    h
578         require((value == 0) || (token.allowance(ad
    dress(this), spender) == 0),
579             "SafeERC20: approve from non-zero to no
    n-zero allowance"
580         );
581         _callOptionalReturn(token, abi.encodeWithSe
    lector(token.approve.selector, spender, value));
582     }
583
584     function safeIncreaseAllowance(IERC20 token, ad
    dress spender, uint256 value) internal {
585         uint256 newAllowance = token.allowance(addr
    ess(this), spender).add(value);
586         _callOptionalReturn(token, abi.encodeWithSe
    lector(token.approve.selector, spender, newAllowanc
    e));
587     }
588
589     function safeDecreaseAllowance(IERC20 token, ad
    dress spender, uint256 value) internal {

```

```

590         uint256 newAllowance = token.allowance(addr
ess(this), spender).sub(value);
591         _callOptionalReturn(token, abi.encodeWithSe
lector(token.approve.selector, spender, newAllowanc
e));
592     }
593
594     /**
595      * @dev Imitates a Solidity high-level call (i.
e. a regular function call to a contract), relaxing
the requirement
596      * on the return value: the return value is opt
ional (but if data is returned, it must not be fals
e).
597      * @param token The token targeted by the call.
598      * @param data The call data (encoded using ab
i.encode or one of its variants).
599      */
600     function _callOptionalReturn(IERC20 token, byte
s memory data) private {
601         // We need to perform a low level call her
e, to bypass Solidity's return data size checking m
echanism, since
602         // we're implementing it ourselves. We use
{Address.functionCall} to perform this call, which
verifies that
603         // the target address contains contract cod
e and also asserts for success in the low-level cal
l.
604
605         bytes memory returndata = address(token).fu
nctionCall(data, "SafeERC20: low-level call faile
d");
606         if (returndata.length > 0) { // Return data
is optional
607             // solhint-disable-next-line max-line-l
ength
608             require(abi.decode(returndata, (bool)),
"SafeERC20: ERC20 operation did not succeed");
609         }
610     }
611 }
612
613 interface IMEMO is IERC20 {
614     function index() external view returns ( uint
);
615 }
616
617 contract wMEMO is ERC20 {
618     using SafeERC20 for IMEMO;
619     using LowGasSafeMath for uint;
620
621     IMEMO public immutable MEMO;
622     event Wrap(address indexed recipient, uint256 a
mountMemo, uint256 amountWmemo);
623     event UnWrap(address indexed recipient,uint256
amountWmemo, uint256 amountMemo);
624
625     constructor( address _MEMO ) ERC20( 'Wrapped ME
MO', 'wMEMO' ) {
626         require( _MEMO != address(0) );
627         MEMO = IMEMO(_MEMO);
628     }
629
630     /**
631      @notice wrap MEMO
632      @param _amount uint
633      @return uint
634      */

```

```

590         uint256 newAllowance = token.allowance(addr
ess(this), spender).sub(value);
591         _callOptionalReturn(token, abi.encodeWithSe
lector(token.approve.selector, spender, newAllowanc
e));
592     }
593
594     /**
595      * @dev Imitates a Solidity high-level call (i.
e. a regular function call to a contract), relaxing
the requirement
596      * on the return value: the return value is opt
ional (but if data is returned, it must not be fals
e).
597      * @param token The token targeted by the call.
598      * @param data The call data (encoded using ab
i.encode or one of its variants).
599      */
600     function _callOptionalReturn(IERC20 token, byte
s memory data) private {
601         // We need to perform a low level call her
e, to bypass Solidity's return data size checking m
echanism, since
602         // we're implementing it ourselves. We use
{Address.functionCall} to perform this call, which
verifies that
603         // the target address contains contract cod
e and also asserts for success in the low-level cal
l.
604
605         bytes memory returndata = address(token).fu
nctionCall(data, "SafeERC20: low-level call faile
d");
606         if (returndata.length > 0) { // Return data
is optional
607             // solhint-disable-next-line max-line-l
ength
608             require(abi.decode(returndata, (bool)),
"SafeERC20: ERC20 operation did not succeed");
609         }
610     }
611 }
612
613 interface IMEMO is IERC20 {
614     function index() external view returns ( uint
);
615 }
616
617 contract wsMAIA is ERC20 {
618     using SafeERC20 for IMEMO;
619     using LowGasSafeMath for uint;
620
621     IMEMO public immutable MEMO;
622     event Wrap(address indexed recipient, uint256 a
mountMemo, uint256 amountWmemo);
623     event UnWrap(address indexed recipient,uint256
amountWmemo, uint256 amountMemo);
624
625     constructor( address _MEMO ) ERC20( 'Wrapped St
aked Maia', 'wsMAIA' ) {
626         require( _MEMO != address(0) );
627         MEMO = IMEMO(_MEMO);
628     }
629
630     /**
631      @notice wrap MEMO
632      @param _amount uint
633      @return uint
634      */

```

```

635     function wrap( uint _amount ) external returns
        ( uint ) {
636         MEMO.safeTransferFrom( msg.sender, address
            (this), _amount );
637
638         uint value = MEMOTowMEMO( _amount );
639         _mint( msg.sender, value );
640         emit Wrap(msg.sender, _amount, value);
641         return value;
642     }
643
644     /**
645         @notice unwrap MEMO
646         @param _amount uint
647         @return uint
648     */
649     function unwrap( uint _amount ) external return
        s ( uint ) {
650         _burn( msg.sender, _amount );
651
652         uint value = wMEMOToMEMO( _amount );
653         MEMO.safeTransfer( msg.sender, value );
654         emit UnWrap(msg.sender, _amount, value);
655         return value;
656     }
657
658     /**
659         @notice converts wMEMO amount to MEMO
660         @param _amount uint
661         @return uint
662     */
663     function wMEMOToMEMO( uint _amount ) public vie
        w returns ( uint ) {
664         return _amount.mul( MEMO.index() ).div( 10
            ** decimals );
665     }
666
667     /**
668         @notice converts MEMO amount to wMEMO
669         @param _amount uint
670         @return uint
671     */
672     function MEMOTowMEMO( uint _amount ) public vie
        w returns ( uint ) {
673         return _amount.mul( 10 ** decimals ).div( M
            EMO.index() );
674     }
675
676 }

```

```

635     function wrap( uint _amount ) external returns
        ( uint ) {
636         MEMO.safeTransferFrom( msg.sender, address
            (this), _amount );
637
638         uint value = MEMOTowMEMO( _amount );
639         _mint( msg.sender, value );
640         emit Wrap(msg.sender, _amount, value);
641         return value;
642     }
643
644     /**
645         @notice unwrap MEMO
646         @param _amount uint
647         @return uint
648     */
649     function unwrap( uint _amount ) external return
        s ( uint ) {
650         _burn( msg.sender, _amount );
651
652         uint value = wMEMOToMEMO( _amount );
653         MEMO.safeTransfer( msg.sender, value );
654         emit UnWrap(msg.sender, _amount, value);
655         return value;
656     }
657
658     /**
659         @notice converts wMEMO amount to MEMO
660         @param _amount uint
661         @return uint
662     */
663     function wMEMOToMEMO( uint _amount ) public vie
        w returns ( uint ) {
664         return _amount.mul( MEMO.index() ).div( 10
            ** decimals );
665     }
666
667     /**
668         @notice converts MEMO amount to wMEMO
669         @param _amount uint
670         @return uint
671     */
672     function MEMOTowMEMO( uint _amount ) public vie
        w returns ( uint ) {
673         return _amount.mul( 10 ** decimals ).div( M
            EMO.index() );
674     }
675
676 }

```