

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3
4 library LowGasSafeMath {
5     /// @notice Returns x + y, reverts if sum over
    flows uint256
6     /// @param x The augend
7     /// @param y The addend
8     /// @return z The sum of x and y
9     function add(uint256 x, uint256 y) internal pu
    re returns (uint256 z) {
10         require((z = x + y) >= x);
11     }
12
13     function add32(uint32 x, uint32 y) internal pu
    re returns (uint32 z) {
14         require((z = x + y) >= x);
15     }
16
17     /// @notice Returns x - y, reverts if underflo
    ws
18     /// @param x The minuend
19     /// @param y The subtrahend
20     /// @return z The difference of x and y
21     function sub(uint256 x, uint256 y) internal pu
    re returns (uint256 z) {
22         require((z = x - y) <= x);
23     }
24
25     function sub32(uint32 x, uint32 y) internal pu
    re returns (uint32 z) {
26         require((z = x - y) <= x);
27     }
28
29     /// @notice Returns x * y, reverts if overflow
    s
30     /// @param x The multiplicand
31     /// @param y The multiplier
32     /// @return z The product of x and y
33     function mul(uint256 x, uint256 y) internal pu
    re returns (uint256 z) {
34         require(x == 0 || (z = x * y) / x == y);
35     }
36
37     /// @notice Returns x + y, reverts if overflow
    s or underflows
38     /// @param x The augend
39     /// @param y The addend
40     /// @return z The sum of x and y
41     function add(int256 x, int256 y) internal pure
    returns (int256 z) {
42         require((z = x + y) >= x == (y >= 0));
43     }
44
45     /// @notice Returns x - y, reverts if overflow
    s or underflows
46     /// @param x The minuend
47     /// @param y The subtrahend
48     /// @return z The difference of x and y
49     function sub(int256 x, int256 y) internal pure
    returns (int256 z) {
50         require((z = x - y) <= x == (y >= 0));
51     }

```

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3
4 library LowGasSafeMath {
5     /// @notice Returns x + y, reverts if sum over
    flows uint256
6     /// @param x The augend
7     /// @param y The addend
8     /// @return z The sum of x and y
9     function add(uint256 x, uint256 y) internal pu
    re returns (uint256 z) {
10         require((z = x + y) >= x);
11     }
12
13     function add32(uint32 x, uint32 y) internal pu
    re returns (uint32 z) {
14         require((z = x + y) >= x);
15     }
16
17     /// @notice Returns x - y, reverts if underflo
    ws
18     /// @param x The minuend
19     /// @param y The subtrahend
20     /// @return z The difference of x and y
21     function sub(uint256 x, uint256 y) internal pu
    re returns (uint256 z) {
22         require((z = x - y) <= x);
23     }
24
25     function sub32(uint32 x, uint32 y) internal pu
    re returns (uint32 z) {
26         require((z = x - y) <= x);
27     }
28
29     /// @notice Returns x * y, reverts if overflow
    s
30     /// @param x The multiplicand
31     /// @param y The multiplier
32     /// @return z The product of x and y
33     function mul(uint256 x, uint256 y) internal pu
    re returns (uint256 z) {
34         require(x == 0 || (z = x * y) / x == y);
35     }
36
37     /// @notice Returns x + y, reverts if overflow
    s or underflows
38     /// @param x The augend
39     /// @param y The addend
40     /// @return z The sum of x and y
41     function add(int256 x, int256 y) internal pure
    returns (int256 z) {
42         require((z = x + y) >= x == (y >= 0));
43     }
44
45     /// @notice Returns x - y, reverts if overflow
    s or underflows
46     /// @param x The minuend
47     /// @param y The subtrahend
48     /// @return z The difference of x and y
49     function sub(int256 x, int256 y) internal pure
    returns (int256 z) {
50         require((z = x - y) <= x == (y >= 0));
51     }

```

```

52
53     function div(uint256 x, uint256 y) internal pu
re returns(uint256 z){
54         require(y > 0);
55         z=x/y;
56     }
57 }
58
59 library Address {
60     /**
61      * @dev Returns true if `account` is a contrac
t.
62      *
63      * [IMPORTANT]
64      * ====
65      * It is unsafe to assume that an address for
which this function returns
66      * false is an externally-owned account (EOA)
and not a contract.
67      *
68      * Among others, `isContract` will return fals
e for the following
69      * types of addresses:
70      *
71      * - an externally-owned account
72      * - a contract in construction
73      * - an address where a contract will be crea
ted
74      * - an address where a contract lived, but w
as destroyed
75      * ====
76      */
77     function isContract(address account) internal
view returns (bool) {
78         // This method relies in extcodesize, whic
h returns 0 for contracts in
79         // construction, since the code is only st
ored at the end of the
80         // constructor execution.
81
82         uint256 size;
83         // solhint-disable-next-line no-inline-ass
embly
84         assembly { size := extcodesize(account) }
85         return size > 0;
86     }
87
88     /**
89      * @dev Replacement for Solidity's `transfer`:
sends `amount` wei to
90      * `recipient`, forwarding all available gas a
nd reverting on errors.
91      *
92      * https://eips.ethereum.org/EIPS/eip-1884[EIP
1884] increases the gas cost
93      * of certain opcodes, possibly making contrac
ts go over the 2300 gas limit
94      * imposed by `transfer`, making them unable t
o receive funds via
95      * `transfer`. {sendValue} removes this limita
tion.
96      *
97      * https://diligence.consensys.net/posts/2019/
09/stop-using-soliditys-transfer-now/[Learn more].
98      *
99      * IMPORTANT: because control is transferred t
o `recipient`, care must be

```

```

52
53     function div(uint256 x, uint256 y) internal pu
re returns(uint256 z){
54         require(y > 0);
55         z=x/y;
56     }
57 }
58
59 library Address {
60     /**
61      * @dev Returns true if `account` is a contrac
t.
62      *
63      * [IMPORTANT]
64      * ====
65      * It is unsafe to assume that an address for
which this function returns
66      * false is an externally-owned account (EOA)
and not a contract.
67      *
68      * Among others, `isContract` will return fals
e for the following
69      * types of addresses:
70      *
71      * - an externally-owned account
72      * - a contract in construction
73      * - an address where a contract will be crea
ted
74      * - an address where a contract lived, but w
as destroyed
75      * ====
76      */
77     function isContract(address account) internal
view returns (bool) {
78         // This method relies in extcodesize, whic
h returns 0 for contracts in
79         // construction, since the code is only st
ored at the end of the
80         // constructor execution.
81
82         uint256 size;
83         // solhint-disable-next-line no-inline-ass
embly
84         assembly { size := extcodesize(account) }
85         return size > 0;
86     }
87
88     /**
89      * @dev Replacement for Solidity's `transfer`:
sends `amount` wei to
90      * `recipient`, forwarding all available gas a
nd reverting on errors.
91      *
92      * https://eips.ethereum.org/EIPS/eip-1884[EIP
1884] increases the gas cost
93      * of certain opcodes, possibly making contrac
ts go over the 2300 gas limit
94      * imposed by `transfer`, making them unable t
o receive funds via
95      * `transfer`. {sendValue} removes this limita
tion.
96      *
97      * https://diligence.consensys.net/posts/2019/
09/stop-using-soliditys-transfer-now/[Learn more].
98      *
99      * IMPORTANT: because control is transferred t
o `recipient`, care must be

```

```

100     * taken to not create reentrancy vulnerabilit
ies. Consider using
101     * {ReentrancyGuard}
102     */
103     function sendValue(address payable recipient,
uint256 amount) internal {
104         require(address(this).balance >= amount,
"Address: insufficient balance");
105
106         // solhint-disable-next-line avoid-low-lev
el-calls, avoid-call-value
107         (bool success, ) = recipient.call{ value:
amount }("");
108         require(success, "Address: unable to send
value, recipient may have reverted");
109     }
110
111     /**
112     * @dev Performs a Solidity function call usin
g a low level `call`. A
113     * plain `call` is an unsafe replacement for a
function call: use this
114     * function instead.
115     *
116     * If `target` reverts with a revert reason, i
t is bubbled up by this
117     * function (like regular Solidity function ca
lls).
118     *
119     * Requirements:
120     *
121     * - `target` must be a contract.
122     * - calling `target` with `data` must not rev
ert.
123     *
124     * _Available since v3.1._
125     */
126     function functionCall(address target, bytes me
mory data) internal returns (bytes memory) {
127         return functionCall(target, data, "Address:
low-level call failed");
128     }
129
130     /**
131     * @dev Same as {xref-Address-functionCall-add
ress-bytes-}[`functionCall`], but with
132     * `errorMessage` as a fallback revert reason
when `target` reverts.
133     *
134     * _Available since v3.1._
135     */
136     function functionCall(
137         address target,
138         bytes memory data,
139         string memory errorMessage
140     ) internal returns (bytes memory) {
141         return _functionCallWithValue(target, dat
a, 0, errorMessage);
142     }
143
144     /**
145     * @dev Same as {xref-Address-functionCall-add
ress-bytes-}[`functionCall`],
146     * but also transferring `value` wei to `targe
t`.
147     *
148     * Requirements:
149     *

```

```

100     * taken to not create reentrancy vulnerabilit
ies. Consider using
101     * {ReentrancyGuard}
102     */
103     function sendValue(address payable recipient,
uint256 amount) internal {
104         require(address(this).balance >= amount,
"Address: insufficient balance");
105
106         // solhint-disable-next-line avoid-low-lev
el-calls, avoid-call-value
107         (bool success, ) = recipient.call{ value:
amount }("");
108         require(success, "Address: unable to send
value, recipient may have reverted");
109     }
110
111     /**
112     * @dev Performs a Solidity function call usin
g a low level `call`. A
113     * plain `call` is an unsafe replacement for a
function call: use this
114     * function instead.
115     *
116     * If `target` reverts with a revert reason, i
t is bubbled up by this
117     * function (like regular Solidity function ca
lls).
118     *
119     * Requirements:
120     *
121     * - `target` must be a contract.
122     * - calling `target` with `data` must not rev
ert.
123     *
124     * _Available since v3.1._
125     */
126     function functionCall(address target, bytes me
mory data) internal returns (bytes memory) {
127         return functionCall(target, data, "Address:
low-level call failed");
128     }
129
130     /**
131     * @dev Same as {xref-Address-functionCall-add
ress-bytes-}[`functionCall`], but with
132     * `errorMessage` as a fallback revert reason
when `target` reverts.
133     *
134     * _Available since v3.1._
135     */
136     function functionCall(
137         address target,
138         bytes memory data,
139         string memory errorMessage
140     ) internal returns (bytes memory) {
141         return _functionCallWithValue(target, dat
a, 0, errorMessage);
142     }
143
144     /**
145     * @dev Same as {xref-Address-functionCall-add
ress-bytes-}[`functionCall`],
146     * but also transferring `value` wei to `targe
t`.
147     *
148     * Requirements:
149     *

```

```

150     * - the calling contract must have an ETH bal
    ance of at least `value`.
151     * - the called Solidity function must be `pay
    able`.
152     *
153     * _Available since v3.1._
154     */
155     function functionCallWithValue(address target,
    bytes memory data, uint256 value) internal returns
    (bytes memory) {
156         return functionCallWithValue(target, data,
    value, "Address: low-level call with value faile
    d");
157     }
158
159     /**
160     * @dev Same as {xref-Address-functionCallWith
    Value-address-bytes-uint256-}[`functionCallWithVal
    ue`], but
161     * with `errorMessage` as a fallback revert re
    ason when `target` reverts.
162     *
163     * _Available since v3.1._
164     */
165     function functionCallWithValue(
166         address target,
167         bytes memory data,
168         uint256 value,
169         string memory errorMessage
170     ) internal returns (bytes memory) {
171         require(address(this).balance >= value, "A
    ddress: insufficient balance for call");
172         require(isContract(target), "Address: call
    to non-contract");
173
174         // solhint-disable-next-line avoid-low-lev
    el-calls
175         (bool success, bytes memory returndata) =
    target.call{ value: value }(data);
176         return _verifyCallResult(success, returnda
    ta, errorMessage);
177     }
178
179     function _functionCallWithValue(
180         address target,
181         bytes memory data,
182         uint256 weiValue,
183         string memory errorMessage
184     ) private returns (bytes memory) {
185         require(isContract(target), "Address: call
    to non-contract");
186
187         // solhint-disable-next-line avoid-low-lev
    el-calls
188         (bool success, bytes memory returndata) =
    target.call{ value: weiValue }(data);
189         if (success) {
190             return returndata;
191         } else {
192             // Look for revert reason and bubble i
    t up if present
193             if (returndata.length > 0) {
194                 // The easiest way to bubble the r
    evert reason is using memory via assembly
195
196                 // solhint-disable-next-line no-in
    line-assembly
197                 assembly {

```

```

150     * - the calling contract must have an ETH bal
    ance of at least `value`.
151     * - the called Solidity function must be `pay
    able`.
152     *
153     * _Available since v3.1._
154     */
155     function functionCallWithValue(address target,
    bytes memory data, uint256 value) internal returns
    (bytes memory) {
156         return functionCallWithValue(target, data,
    value, "Address: low-level call with value faile
    d");
157     }
158
159     /**
160     * @dev Same as {xref-Address-functionCallWith
    Value-address-bytes-uint256-}[`functionCallWithVal
    ue`], but
161     * with `errorMessage` as a fallback revert re
    ason when `target` reverts.
162     *
163     * _Available since v3.1._
164     */
165     function functionCallWithValue(
166         address target,
167         bytes memory data,
168         uint256 value,
169         string memory errorMessage
170     ) internal returns (bytes memory) {
171         require(address(this).balance >= value, "A
    ddress: insufficient balance for call");
172         require(isContract(target), "Address: call
    to non-contract");
173
174         // solhint-disable-next-line avoid-low-lev
    el-calls
175         (bool success, bytes memory returndata) =
    target.call{ value: value }(data);
176         return _verifyCallResult(success, returnda
    ta, errorMessage);
177     }
178
179     function _functionCallWithValue(
180         address target,
181         bytes memory data,
182         uint256 weiValue,
183         string memory errorMessage
184     ) private returns (bytes memory) {
185         require(isContract(target), "Address: call
    to non-contract");
186
187         // solhint-disable-next-line avoid-low-lev
    el-calls
188         (bool success, bytes memory returndata) =
    target.call{ value: weiValue }(data);
189         if (success) {
190             return returndata;
191         } else {
192             // Look for revert reason and bubble i
    t up if present
193             if (returndata.length > 0) {
194                 // The easiest way to bubble the r
    evert reason is using memory via assembly
195
196                 // solhint-disable-next-line no-in
    line-assembly
197                 assembly {

```

```

198         let returndata_size := mload(r
eturndata)
199         revert(add(32, returndata), re
turndata_size)
200     }
201     } else {
202         revert(errorMessage);
203     }
204 }
205 }
206
207 /**
208  * @dev Same as {xref-Address-functionCall-add
ress-bytes-}[`functionCall`],
209  * but performing a static call.
210  *
211  * _Available since v3.3._
212  */
213 function functionStaticCall(address target, by
tes memory data) internal view returns (bytes memo
ry) {
214     return functionStaticCall(target, data, "A
ddress: low-level static call failed");
215 }
216
217 /**
218  * @dev Same as {xref-Address-functionCall-add
ress-bytes-string-}[`functionCall`],
219  * but performing a static call.
220  *
221  * _Available since v3.3._
222  */
223 function functionStaticCall(
224     address target,
225     bytes memory data,
226     string memory errorMessage
227 ) internal view returns (bytes memory) {
228     require(isContract(target), "Address: stat
ic call to non-contract");
229
230     // solhint-disable-next-line avoid-low-lev
el-calls
231     (bool success, bytes memory returndata) =
target.staticcall(data);
232     return _verifyCallResult(success, returnda
ta, errorMessage);
233 }
234
235 /**
236  * @dev Same as {xref-Address-functionCall-add
ress-bytes-}[`functionCall`],
237  * but performing a delegate call.
238  *
239  * _Available since v3.3._
240  */
241 function functionDelegateCall(address target,
bytes memory data) internal returns (bytes memor
y) {
242     return functionDelegateCall(target, data,
"Address: low-level delegate call failed");
243 }
244
245 /**
246  * @dev Same as {xref-Address-functionCall-add
ress-bytes-string-}[`functionCall`],
247  * but performing a delegate call.
248  *
249  * _Available since v3.3._

```

```

198         let returndata_size := mload(r
eturndata)
199         revert(add(32, returndata), re
turndata_size)
200     }
201     } else {
202         revert(errorMessage);
203     }
204 }
205 }
206
207 /**
208  * @dev Same as {xref-Address-functionCall-add
ress-bytes-}[`functionCall`],
209  * but performing a static call.
210  *
211  * _Available since v3.3._
212  */
213 function functionStaticCall(address target, by
tes memory data) internal view returns (bytes memo
ry) {
214     return functionStaticCall(target, data, "A
ddress: low-level static call failed");
215 }
216
217 /**
218  * @dev Same as {xref-Address-functionCall-add
ress-bytes-string-}[`functionCall`],
219  * but performing a static call.
220  *
221  * _Available since v3.3._
222  */
223 function functionStaticCall(
224     address target,
225     bytes memory data,
226     string memory errorMessage
227 ) internal view returns (bytes memory) {
228     require(isContract(target), "Address: stat
ic call to non-contract");
229
230     // solhint-disable-next-line avoid-low-lev
el-calls
231     (bool success, bytes memory returndata) =
target.staticcall(data);
232     return _verifyCallResult(success, returnda
ta, errorMessage);
233 }
234
235 /**
236  * @dev Same as {xref-Address-functionCall-add
ress-bytes-}[`functionCall`],
237  * but performing a delegate call.
238  *
239  * _Available since v3.3._
240  */
241 function functionDelegateCall(address target,
bytes memory data) internal returns (bytes memor
y) {
242     return functionDelegateCall(target, data,
"Address: low-level delegate call failed");
243 }
244
245 /**
246  * @dev Same as {xref-Address-functionCall-add
ress-bytes-string-}[`functionCall`],
247  * but performing a delegate call.
248  *
249  * _Available since v3.3._

```

```

250     */
251     function functionDelegateCall(
252         address target,
253         bytes memory data,
254         string memory errorMessage
255     ) internal returns (bytes memory) {
256         require(isContract(target), "Address: dele
gate call to non-contract");
257
258         // solhint-disable-next-line avoid-low-lev
el-calls
259         (bool success, bytes memory returndata) =
target.delegatecall(data);
260         return _verifyCallResult(success, returnda
ta, errorMessage);
261     }
262
263     function _verifyCallResult(
264         bool success,
265         bytes memory returndata,
266         string memory errorMessage
267     ) private pure returns(bytes memory) {
268         if (success) {
269             return returndata;
270         } else {
271             // Look for revert reason and bubble i
t up if present
272             if (returndata.length > 0) {
273                 // The easiest way to bubble the r
ever reason is using memory via assembly
274
275                 // solhint-disable-next-line no-in
line-assembly
276                 assembly {
277                     let returndata_size := mload(r
eturndata)
278                     revert(add(32, returndata), re
turndata_size)
279                 }
280             } else {
281                 revert(errorMessage);
282             }
283         }
284     }
285
286     function addressToString(address _address) int
ernal pure returns(string memory) {
287         bytes32 _bytes = bytes32(uint256(_adres
s));
288         bytes memory HEX = "0123456789abcdef";
289         bytes memory _addr = new bytes(42);
290
291         _addr[0] = '0';
292         _addr[1] = 'x';
293
294         for(uint256 i = 0; i < 20; i++) {
295             _addr[2+i*2] = HEX[uint8(_bytes[i + 1
2] >> 4)];
296             _addr[3+i*2] = HEX[uint8(_bytes[i + 1
2] & 0x0f)];
297         }
298
299         return string(_addr);
300     }
301 }
302 }
303
304 interface IERC20 {
305     /**

```

```

250     */
251     function functionDelegateCall(
252         address target,
253         bytes memory data,
254         string memory errorMessage
255     ) internal returns (bytes memory) {
256         require(isContract(target), "Address: dele
gate call to non-contract");
257
258         // solhint-disable-next-line avoid-low-lev
el-calls
259         (bool success, bytes memory returndata) =
target.delegatecall(data);
260         return _verifyCallResult(success, returnda
ta, errorMessage);
261     }
262
263     function _verifyCallResult(
264         bool success,
265         bytes memory returndata,
266         string memory errorMessage
267     ) private pure returns(bytes memory) {
268         if (success) {
269             return returndata;
270         } else {
271             // Look for revert reason and bubble i
t up if present
272             if (returndata.length > 0) {
273                 // The easiest way to bubble the r
ever reason is using memory via assembly
274
275                 // solhint-disable-next-line no-in
line-assembly
276                 assembly {
277                     let returndata_size := mload(r
eturndata)
278                     revert(add(32, returndata), re
turndata_size)
279                 }
280             } else {
281                 revert(errorMessage);
282             }
283         }
284     }
285
286     function addressToString(address _address) int
ernal pure returns(string memory) {
287         bytes32 _bytes = bytes32(uint256(_adres
s));
288         bytes memory HEX = "0123456789abcdef";
289         bytes memory _addr = new bytes(42);
290
291         _addr[0] = '0';
292         _addr[1] = 'x';
293
294         for(uint256 i = 0; i < 20; i++) {
295             _addr[2+i*2] = HEX[uint8(_bytes[i + 1
2] >> 4)];
296             _addr[3+i*2] = HEX[uint8(_bytes[i + 1
2] & 0x0f)];
297         }
298
299         return string(_addr);
300     }
301 }
302 }
303
304 interface IERC20 {
305     /**

```

```

306     * @dev Returns the amount of tokens in existenc
e.
307     */
308     function totalSupply() external view returns (ui
nt256);
309
310     /**
311     * @dev Returns the amount of tokens owned by `a
ccount`.
312     */
313     function balanceOf(address account) external vie
w returns (uint256);
314
315     /**
316     * @dev Moves `amount` tokens from the caller's
account to `recipient`.
317     *
318     * Returns a boolean value indicating whether th
e operation succeeded.
319     *
320     * Emits a {Transfer} event.
321     */
322     function transfer(address recipient, uint256 amo
unt) external returns (bool);
323
324     /**
325     * @dev Returns the remaining number of tokens t
hat `spender` will be
326     * allowed to spend on behalf of `owner` through
{transferFrom}. This is
327     * zero by default.
328     *
329     * This value changes when {approve} or {transfe
rFrom} are called.
330     */
331     function allowance(address owner, address spende
r) external view returns (uint256);
332
333     /**
334     * @dev Sets `amount` as the allowance of `spend
er` over the caller's tokens.
335     *
336     * Returns a boolean value indicating whether th
e operation succeeded.
337     *
338     * IMPORTANT: Beware that changing an allowance
with this method brings the risk
339     * that someone may use both the old and the new
allowance by unfortunate
340     * transaction ordering. One possible solution t
o mitigate this race
341     * condition is to first reduce the spender's al
lowance to 0 and set the
342     * desired value afterwards:
343     * https://github.com/ethereum/EIPs/issues/20#is
suecomment-263524729
344     *
345     * Emits an {Approval} event.
346     */
347     function approve(address spender, uint256 amoun
t) external returns (bool);
348
349     /**
350     * @dev Moves `amount` tokens from `sender` to `
recipient` using the
351     * allowance mechanism. `amount` is then deducte
d from the caller's
352     * allowance.

```

```

306     * @dev Returns the amount of tokens in existenc
e.
307     */
308     function totalSupply() external view returns (ui
nt256);
309
310     /**
311     * @dev Returns the amount of tokens owned by `a
ccount`.
312     */
313     function balanceOf(address account) external vie
w returns (uint256);
314
315     /**
316     * @dev Moves `amount` tokens from the caller's
account to `recipient`.
317     *
318     * Returns a boolean value indicating whether th
e operation succeeded.
319     *
320     * Emits a {Transfer} event.
321     */
322     function transfer(address recipient, uint256 amo
unt) external returns (bool);
323
324     /**
325     * @dev Returns the remaining number of tokens t
hat `spender` will be
326     * allowed to spend on behalf of `owner` through
{transferFrom}. This is
327     * zero by default.
328     *
329     * This value changes when {approve} or {transfe
rFrom} are called.
330     */
331     function allowance(address owner, address spende
r) external view returns (uint256);
332
333     /**
334     * @dev Sets `amount` as the allowance of `spend
er` over the caller's tokens.
335     *
336     * Returns a boolean value indicating whether th
e operation succeeded.
337     *
338     * IMPORTANT: Beware that changing an allowance
with this method brings the risk
339     * that someone may use both the old and the new
allowance by unfortunate
340     * transaction ordering. One possible solution t
o mitigate this race
341     * condition is to first reduce the spender's al
lowance to 0 and set the
342     * desired value afterwards:
343     * https://github.com/ethereum/EIPs/issues/20#is
suecomment-263524729
344     *
345     * Emits an {Approval} event.
346     */
347     function approve(address spender, uint256 amoun
t) external returns (bool);
348
349     /**
350     * @dev Moves `amount` tokens from `sender` to `
recipient` using the
351     * allowance mechanism. `amount` is then deducte
d from the caller's
352     * allowance.

```

```

353     *
354     * Returns a boolean value indicating whether the operation succeeded.
355     *
356     * Emits a {Transfer} event.
357     */
358     function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
359
360     /**
361     * @dev Emitted when `value` tokens are moved from one account (`from`) to
362     * another (`to`).
363     *
364     * Note that `value` may be zero.
365     */
366     event Transfer(address indexed from, address indexed to, uint256 value);
367
368     /**
369     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
370     * a call to {approve}. `value` is the new allowance.
371     */
372     event Approval(address indexed owner, address indexed spender, uint256 value);
373 }
374
375 abstract contract ERC20
376 is
377     IERC20
378 {
379
380     using LowGasSafeMath for uint256;
381
382     // TODO comment actual hash value.
383     bytes32 constant private ERC20TOKEN_ERC1820_INTERFACED_ID = keccak256( "ERC20Token" );
384
385     mapping (address => uint256) internal _balances;
386
387     mapping (address => mapping (address => uint256)) internal _allowances;
388
389     uint256 internal _totalSupply;
390
391     string internal _name;
392
393     string internal _symbol;
394
395     uint8 internal _decimals;
396
397     /**
398     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
399     * a default value of 18.
400     *
401     * To select a different value for {decimals}, use {_setupDecimals}.
402     *
403     * All three of these values are immutable: they can only be set once during
404     * construction.
405     */
406     constructor (string memory name_, string memory symbol_, uint8 decimals_) {
407         _name = name_;
408         _symbol = symbol_;

```

```

353     *
354     * Returns a boolean value indicating whether the operation succeeded.
355     *
356     * Emits a {Transfer} event.
357     */
358     function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
359
360     /**
361     * @dev Emitted when `value` tokens are moved from one account (`from`) to
362     * another (`to`).
363     *
364     * Note that `value` may be zero.
365     */
366     event Transfer(address indexed from, address indexed to, uint256 value);
367
368     /**
369     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
370     * a call to {approve}. `value` is the new allowance.
371     */
372     event Approval(address indexed owner, address indexed spender, uint256 value);
373 }
374
375 abstract contract ERC20
376 is
377     IERC20
378 {
379
380     using LowGasSafeMath for uint256;
381
382     // TODO comment actual hash value.
383     bytes32 constant private ERC20TOKEN_ERC1820_INTERFACED_ID = keccak256( "ERC20Token" );
384
385     mapping (address => uint256) internal _balances;
386
387     mapping (address => mapping (address => uint256)) internal _allowances;
388
389     uint256 internal _totalSupply;
390
391     string internal _name;
392
393     string internal _symbol;
394
395     uint8 internal _decimals;
396
397     /**
398     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
399     * a default value of 18.
400     *
401     * To select a different value for {decimals}, use {_setupDecimals}.
402     *
403     * All three of these values are immutable: they can only be set once during
404     * construction.
405     */
406     constructor (string memory name_, string memory symbol_, uint8 decimals_) {
407         _name = name_;
408         _symbol = symbol_;

```



```

409     _decimals = decimals_;
410 }
411
412 /**
413  * @dev Returns the name of the token.
414  */
415 function name() public view returns (string memo
ry) {
416     return _name;
417 }
418
419 /**
420  * @dev Returns the symbol of the token, usually
a shorter version of the
421  * name.
422  */
423 function symbol() public view returns (string me
mory) {
424     return _symbol;
425 }
426
427 /**
428  * @dev Returns the number of decimals used to g
et its user representation.
429  * For example, if `decimals` equals `2`, a bala
nce of `505` tokens should
430  * be displayed to a user as `5,05` (`505 / 10 *
2`).
431  *
432  * Tokens usually opt for a value of 18, imitati
ng the relationship between
433  * Ether and Wei. This is the value {ERC20} use
s, unless {_setupDecimals} is
434  * called.
435  *
436  * NOTE: This information is only used for _disp
lay_ purposes: it in
437  * no way affects any of the arithmetic of the c
ontract, including
438  * {IERC20-balanceOf} and {IERC20-transfer}.
439  */
440 function decimals() public view returns (uint8)
{
441     return _decimals;
442 }
443
444 /**
445  * @dev See {IERC20-totalSupply}.
446  */
447 function totalSupply() public view override retu
rns (uint256) {
448     return _totalSupply;
449 }
450
451 /**
452  * @dev See {IERC20-balanceOf}.
453  */
454 function balanceOf(address account) public view
virtual override returns (uint256) {
455     return _balances[account];
456 }
457
458 /**
459  * @dev See {IERC20-transfer}.
460  *
461  * Requirements:
462  *
463  * - `recipient` cannot be the zero address.

```

```

409     _decimals = decimals_;
410 }
411
412 /**
413  * @dev Returns the name of the token.
414  */
415 function name() public view returns (string memo
ry) {
416     return _name;
417 }
418
419 /**
420  * @dev Returns the symbol of the token, usually
a shorter version of the
421  * name.
422  */
423 function symbol() public view returns (string me
mory) {
424     return _symbol;
425 }
426
427 /**
428  * @dev Returns the number of decimals used to g
et its user representation.
429  * For example, if `decimals` equals `2`, a bala
nce of `505` tokens should
430  * be displayed to a user as `5,05` (`505 / 10 *
2`).
431  *
432  * Tokens usually opt for a value of 18, imitati
ng the relationship between
433  * Ether and Wei. This is the value {ERC20} use
s, unless {_setupDecimals} is
434  * called.
435  *
436  * NOTE: This information is only used for _disp
lay_ purposes: it in
437  * no way affects any of the arithmetic of the c
ontract, including
438  * {IERC20-balanceOf} and {IERC20-transfer}.
439  */
440 function decimals() public view returns (uint8)
{
441     return _decimals;
442 }
443
444 /**
445  * @dev See {IERC20-totalSupply}.
446  */
447 function totalSupply() public view override retu
rns (uint256) {
448     return _totalSupply;
449 }
450
451 /**
452  * @dev See {IERC20-balanceOf}.
453  */
454 function balanceOf(address account) public view
virtual override returns (uint256) {
455     return _balances[account];
456 }
457
458 /**
459  * @dev See {IERC20-transfer}.
460  *
461  * Requirements:
462  *
463  * - `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  *

```

```

516     * Emits an {Approval} event indicating the up
dated allowance.
517     *
518     * Requirements:
519     *
520     * - `spender` cannot be the zero address.
521     */
522     function increaseAllowance(address spender, ui
nt256 addedValue) public virtual returns (bool) {
523         _approve(msg.sender, spender, _allowances
[msg.sender][spender].add(addedValue));
524         return true;
525     }
526
527     /**
528     * @dev Atomically decreases the allowance gra
nted to `spender` by the caller.
529     *
530     * This is an alternative to {approve} that ca
n be used as a mitigation for
531     * problems described in {IERC20-approve}.
532     *
533     * Emits an {Approval} event indicating the up
dated allowance.
534     *
535     * Requirements:
536     *
537     * - `spender` cannot be the zero address.
538     * - `spender` must have allowance for the cal
ler of at least
539     * `subtractedValue`.
540     */
541     function decreaseAllowance(address spender, ui
nt256 subtractedValue) public virtual returns (boo
l) {
542         _approve(msg.sender, spender, _allowances
[msg.sender][spender]
543             .sub(subtractedValue));
544         return true;
545     }
546
547     /**
548     * @dev Moves tokens `amount` from `sender` to `
recipient`.
549     *
550     * This is internal function is equivalent to {t
ransfer}, and can be used to
551     * e.g. implement automatic token fees, slashing
mechanisms, etc.
552     *
553     * Emits a {Transfer} event.
554     *
555     * Requirements:
556     *
557     * - `sender` cannot be the zero address.
558     * - `recipient` cannot be the zero address.
559     * - `sender` must have a balance of at least `a
mount`.
560     */
561     function _transfer(address sender, address recip
ient, uint256 amount) internal virtual {
562         require(sender != address(0), "ERC20: transfer
from the zero address");
563         require(recipient != address(0), "ERC20: trans
fer to the zero address");
564         _beforeTokenTransfer(sender, recipient, amoun
t);

```

```

516     * Emits an {Approval} event indicating the up
dated allowance.
517     *
518     * Requirements:
519     *
520     * - `spender` cannot be the zero address.
521     */
522     function increaseAllowance(address spender, ui
nt256 addedValue) public virtual returns (bool) {
523         _approve(msg.sender, spender, _allowances
[msg.sender][spender].add(addedValue));
524         return true;
525     }
526
527     /**
528     * @dev Atomically decreases the allowance gra
nted to `spender` by the caller.
529     *
530     * This is an alternative to {approve} that ca
n be used as a mitigation for
531     * problems described in {IERC20-approve}.
532     *
533     * Emits an {Approval} event indicating the up
dated allowance.
534     *
535     * Requirements:
536     *
537     * - `spender` cannot be the zero address.
538     * - `spender` must have allowance for the cal
ler of at least
539     * `subtractedValue`.
540     */
541     function decreaseAllowance(address spender, ui
nt256 subtractedValue) public virtual returns (boo
l) {
542         _approve(msg.sender, spender, _allowances
[msg.sender][spender]
543             .sub(subtractedValue));
544         return true;
545     }
546
547     /**
548     * @dev Moves tokens `amount` from `sender` to `
recipient`.
549     *
550     * This is internal function is equivalent to {t
ransfer}, and can be used to
551     * e.g. implement automatic token fees, slashing
mechanisms, etc.
552     *
553     * Emits a {Transfer} event.
554     *
555     * Requirements:
556     *
557     * - `sender` cannot be the zero address.
558     * - `recipient` cannot be the zero address.
559     * - `sender` must have a balance of at least `a
mount`.
560     */
561     function _transfer(address sender, address recip
ient, uint256 amount) internal virtual {
562         require(sender != address(0), "ERC20: transfer
from the zero address");
563         require(recipient != address(0), "ERC20: trans
fer to the zero address");
564         _beforeTokenTransfer(sender, recipient, amoun
t);

```

```

566
567     _balances[sender] = _balances[sender].sub(ammou
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 ammount
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 ammount
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.

```

```

615     *
616     * Emits an {Approval} event.
617     *
618     * Requirements:
619     *
620     * - `owner` cannot be the zero address.
621     * - `spender` cannot be the zero address.
622     */
623     function _approve(address owner, address spender, uint256 amount) internal virtual {
624         require(owner != address(0), "ERC20: approve from the zero address");
625         require(spender != address(0), "ERC20: approve to the zero address");
626
627         _allowances[owner][spender] = amount;
628         emit Approval(owner, spender, amount);
629     }
630
631     /**
632     * @dev Sets {decimals} to a value other than the default one of 18.
633     *
634     * WARNING: This function should only be called from the constructor. Most
635     * applications that interact with token contracts will not expect
636     * {decimals} to ever change, and may work incorrectly if it does.
637     */
638
639     /**
640     * @dev Hook that is called before any transfer of tokens. This includes
641     * minting and burning.
642     *
643     * Calling conditions:
644     *
645     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
646     *   will be transferred to `to`.
647     * - when `from` is zero, `amount` tokens will be minted for `to`.
648     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
649     * - `from` and `to` are never both zero.
650     *
651     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
652     */
653     function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
654 }
655
656 library Counters {
657     using LowGasSafeMath for uint256;
658
659     struct Counter {
660         // This variable should never be directly accessed by users of the library: interactions must be restricted to
661         // the library's function. As of Solidity v0.5.2, this cannot be enforced, though there is a proposal to add
662         // this feature: see https://github.com/ethereum/solidity/issues/4637
663         uint256 _value; // default: 0
664     }
665

```

```

615     *
616     * Emits an {Approval} event.
617     *
618     * Requirements:
619     *
620     * - `owner` cannot be the zero address.
621     * - `spender` cannot be the zero address.
622     */
623     function _approve(address owner, address spender, uint256 amount) internal virtual {
624         require(owner != address(0), "ERC20: approve from the zero address");
625         require(spender != address(0), "ERC20: approve to the zero address");
626
627         _allowances[owner][spender] = amount;
628         emit Approval(owner, spender, amount);
629     }
630
631     /**
632     * @dev Sets {decimals} to a value other than the default one of 18.
633     *
634     * WARNING: This function should only be called from the constructor. Most
635     * applications that interact with token contracts will not expect
636     * {decimals} to ever change, and may work incorrectly if it does.
637     */
638
639     /**
640     * @dev Hook that is called before any transfer of tokens. This includes
641     * minting and burning.
642     *
643     * Calling conditions:
644     *
645     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
646     *   will be transferred to `to`.
647     * - when `from` is zero, `amount` tokens will be minted for `to`.
648     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
649     * - `from` and `to` are never both zero.
650     *
651     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
652     */
653     function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
654 }
655
656 library Counters {
657     using LowGasSafeMath for uint256;
658
659     struct Counter {
660         // This variable should never be directly accessed by users of the library: interactions must be restricted to
661         // the library's function. As of Solidity v0.5.2, this cannot be enforced, though there is a proposal to add
662         // this feature: see https://github.com/ethereum/solidity/issues/4637
663         uint256 _value; // default: 0
664     }
665

```

```

666     function current(Counter storage counter) internal view returns (uint256) {
667         return counter._value;
668     }
669
670     function increment(Counter storage counter) internal {
671         // The {SafeMath} overflow check can be skipped here, see the comment at the top
672         counter._value += 1;
673     }
674
675     function decrement(Counter storage counter) internal {
676         counter._value = counter._value.sub(1);
677     }
678 }
679
680 interface IERC2612Permit {
681     /**
682      * @dev Sets `amount` as the allowance of `spender` over `owner`'s tokens,
683      * given `owner`'s signed approval.
684      *
685      * IMPORTANT: The same issues {IERC20-approve} has related to transaction
686      * ordering also apply here.
687      *
688      * Emits an {Approval} event.
689      *
690      * Requirements:
691      *
692      * - `owner` cannot be the zero address.
693      * - `spender` cannot be the zero address.
694      * - `deadline` must be a timestamp in the future.
695      * - `v`, `r` and `s` must be a valid `secp256k1` signature from `owner`
696      * over the EIP712-formatted function arguments.
697      * - the signature must use `owner`'s current nonce (see {nonces}).
698      *
699      * For more information on the signature format, see the
700      * https://eips.ethereum.org/EIPS/eip-2612#specification[relevant EIP
701      * section].
702      */
703     function permit(
704         address owner,
705         address spender,
706         uint256 amount,
707         uint256 deadline,
708         uint8 v,
709         bytes32 r,
710         bytes32 s
711     ) external;
712
713     /**
714      * @dev Returns the current ERC2612 nonce for `owner`. This value must be
715      * included whenever a signature is generated for {permit}.
716      *
717      * Every successful call to {permit} increases `owner`'s nonce by one. This

```

```

666     function current(Counter storage counter) internal view returns (uint256) {
667         return counter._value;
668     }
669
670     function increment(Counter storage counter) internal {
671         // The {SafeMath} overflow check can be skipped here, see the comment at the top
672         counter._value += 1;
673     }
674
675     function decrement(Counter storage counter) internal {
676         counter._value = counter._value.sub(1);
677     }
678 }
679
680 interface IERC2612Permit {
681     /**
682      * @dev Sets `amount` as the allowance of `spender` over `owner`'s tokens,
683      * given `owner`'s signed approval.
684      *
685      * IMPORTANT: The same issues {IERC20-approve} has related to transaction
686      * ordering also apply here.
687      *
688      * Emits an {Approval} event.
689      *
690      * Requirements:
691      *
692      * - `owner` cannot be the zero address.
693      * - `spender` cannot be the zero address.
694      * - `deadline` must be a timestamp in the future.
695      * - `v`, `r` and `s` must be a valid `secp256k1` signature from `owner`
696      * over the EIP712-formatted function arguments.
697      * - the signature must use `owner`'s current nonce (see {nonces}).
698      *
699      * For more information on the signature format, see the
700      * https://eips.ethereum.org/EIPS/eip-2612#specification[relevant EIP
701      * section].
702      */
703     function permit(
704         address owner,
705         address spender,
706         uint256 amount,
707         uint256 deadline,
708         uint8 v,
709         bytes32 r,
710         bytes32 s
711     ) external;
712
713     /**
714      * @dev Returns the current ERC2612 nonce for `owner`. This value must be
715      * included whenever a signature is generated for {permit}.
716      *
717      * Every successful call to {permit} increases `owner`'s nonce by one. This

```

```

718     * prevents a signature from being used multip
le times.
719     */
720     function nonces(address owner) external view r
eturns (uint256);
721 }
722
723 abstract contract ERC20Permit is ERC20, IERC2612Pe
rmit {
724     using Counters for Counters.Counter;
725
726     mapping(address => Counters.Counter) private _
nonces;
727
728     // keccak256("Permit(address owner,address spe
nder,uint256 value,uint256 nonce,uint256 deadlin
e)");
729     bytes32 public constant PERMIT_TYPEHASH = 0x6e
71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c
64845d6126c9;
730
731     bytes32 public immutable DOMAIN_SEPARATOR;
732
733     constructor() {
734
735         uint256 chainID;
736         assembly {
737             chainID := chainid()
738         }
739
740         DOMAIN_SEPARATOR = keccak256(abi.encode(
741             keccak256("EIP712Domain(string name,st
ring version,uint256 chainId,address verifyingCont
ract)"),
742             keccak256(bytes(name())),
743             keccak256(bytes("1")), // Version
744             chainID,
745             address(this)
746         ));
747     }
748
749     /**
750     * @dev See {IERC2612Permit-permit}.
751     *
752     */
753     function permit(
754         address owner,
755         address spender,
756         uint256 amount,
757         uint256 deadline,
758         uint8 v,
759         bytes32 r,
760         bytes32 s
761     ) public virtual override {
762         require(block.timestamp <= deadline, "Perm
it: expired deadline");
763
764         bytes32 hashStruct =
765             keccak256(abi.encode(PERMIT_TYPEHASH,
owner, spender, amount, _nonces[owner].current(),
deadline));
766
767         bytes32 _hash = keccak256(abi.encodePacked
(uint16(0x1901), DOMAIN_SEPARATOR, hashStruct));
768
769         address signer = ecrecover(_hash, v, r,
s);
770         require(signer != address(0) && signer ==
owner, "ERC20Permit: Invalid signature");
771

```

```

718     * prevents a signature from being used multip
le times.
719     */
720     function nonces(address owner) external view r
eturns (uint256);
721 }
722
723 abstract contract ERC20Permit is ERC20, IERC2612Pe
rmit {
724     using Counters for Counters.Counter;
725
726     mapping(address => Counters.Counter) private _
nonces;
727
728     // keccak256("Permit(address owner,address spe
nder,uint256 value,uint256 nonce,uint256 deadlin
e)");
729     bytes32 public constant PERMIT_TYPEHASH = 0x6e
71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c
64845d6126c9;
730
731     bytes32 public immutable DOMAIN_SEPARATOR;
732
733     constructor() {
734
735         uint256 chainID;
736         assembly {
737             chainID := chainid()
738         }
739
740         DOMAIN_SEPARATOR = keccak256(abi.encode(
741             keccak256("EIP712Domain(string name,st
ring version,uint256 chainId,address verifyingCont
ract)"),
742             keccak256(bytes(name())),
743             keccak256(bytes("1")), // Version
744             chainID,
745             address(this)
746         ));
747     }
748
749     /**
750     * @dev See {IERC2612Permit-permit}.
751     *
752     */
753     function permit(
754         address owner,
755         address spender,
756         uint256 amount,
757         uint256 deadline,
758         uint8 v,
759         bytes32 r,
760         bytes32 s
761     ) public virtual override {
762         require(block.timestamp <= deadline, "Perm
it: expired deadline");
763
764         bytes32 hashStruct =
765             keccak256(abi.encode(PERMIT_TYPEHASH,
owner, spender, amount, _nonces[owner].current(),
deadline));
766
767         bytes32 _hash = keccak256(abi.encodePacked
(uint16(0x1901), DOMAIN_SEPARATOR, hashStruct));
768
769         address signer = ecrecover(_hash, v, r,
s);
770         require(signer != address(0) && signer ==
owner, "ERC20Permit: Invalid signature");
771

```

```

772     _nonces[owner].increment();
773     _approve(owner, spender, amount);
774 }
775
776 /**
777  * @dev See {IERC2612Permit-nonces}.
778  */
779 function nonces(address owner) public view override returns (uint256) {
780     return _nonces[owner].current();
781 }
782 }
783
784 contract OwnableData {
785     address public owner;
786     address public pendingOwner;
787 }
788
789 contract Ownable is OwnableData {
790     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
791
792     /// @notice `owner` defaults to msg.sender on construction.
793     constructor() {
794         owner = msg.sender;
795         emit OwnershipTransferred(address(0), msg.sender);
796     }
797
798     /// @notice Transfers ownership to `newOwner`. Either directly or claimable by the new pending owner.
799     /// Can only be invoked by the current `owner`.
800     /// @param newOwner Address of the new owner.
801     /// @param direct True if `newOwner` should be set immediately. False if `newOwner` needs to use `claimOwnership`.
802     /// @param renounce Allows the `newOwner` to be `address(0)` if `direct` and `renounce` is True. Has no effect otherwise.
803     function transferOwnership(
804         address newOwner,
805         bool direct,
806         bool renounce
807     ) public onlyOwner {
808         if (direct) {
809             // Checks
810             require(newOwner != address(0) || renounce, "Ownable: zero address");
811
812             // Effects
813             emit OwnershipTransferred(owner, newOwner);
814
815             owner = newOwner;
816             pendingOwner = address(0);
817         } else {
818             // Effects
819             pendingOwner = newOwner;
820         }
821     }
822
823     /// @notice Needs to be called by `pendingOwner` to claim ownership.
824     function claimOwnership() public {
825         address _pendingOwner = pendingOwner;
826         // Checks

```

```

772     _nonces[owner].increment();
773     _approve(owner, spender, amount);
774 }
775
776 /**
777  * @dev See {IERC2612Permit-nonces}.
778  */
779 function nonces(address owner) public view override returns (uint256) {
780     return _nonces[owner].current();
781 }
782 }
783
784 contract OwnableData {
785     address public owner;
786     address public pendingOwner;
787 }
788
789 contract Ownable is OwnableData {
790     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
791
792     /// @notice `owner` defaults to msg.sender on construction.
793     constructor() {
794         owner = msg.sender;
795         emit OwnershipTransferred(address(0), msg.sender);
796     }
797
798     /// @notice Transfers ownership to `newOwner`. Either directly or claimable by the new pending owner.
799     /// Can only be invoked by the current `owner`.
800     /// @param newOwner Address of the new owner.
801     /// @param direct True if `newOwner` should be set immediately. False if `newOwner` needs to use `claimOwnership`.
802     /// @param renounce Allows the `newOwner` to be `address(0)` if `direct` and `renounce` is True. Has no effect otherwise.
803     function transferOwnership(
804         address newOwner,
805         bool direct,
806         bool renounce
807     ) public onlyOwner {
808         if (direct) {
809             // Checks
810             require(newOwner != address(0) || renounce, "Ownable: zero address");
811
812             // Effects
813             emit OwnershipTransferred(owner, newOwner);
814
815             owner = newOwner;
816             pendingOwner = address(0);
817         } else {
818             // Effects
819             pendingOwner = newOwner;
820         }
821     }
822
823     /// @notice Needs to be called by `pendingOwner` to claim ownership.
824     function claimOwnership() public {
825         address _pendingOwner = pendingOwner;
826         // Checks

```



```

827         require(msg.sender == _pendingOwner, "Ownable: caller != pending owner");
828
829         // Effects
830         emit OwnershipTransferred(owner, _pendingOwner);
831
832         owner = _pendingOwner;
833         pendingOwner = address(0);
834     }
835     /// @notice Only allows the `owner` to execute the function.
836     modifier onlyOwner() {
837         require(msg.sender == owner, "Ownable: caller is not the owner");
838         _;
839     }
840 }
841
842 contract MEMORies is ERC20Permit, Ownable {
843
844     using LowGasSafeMath for uint256;
845
846     modifier onlyStakingContract() {
847         require( msg.sender == stakingContract, "Ownable: caller is not the owner");
848         _;
849     }
850
851     address public stakingContract;
852     address public initializer;
853
854     event LogSupply(uint256 indexed epoch, uint256 timestamp, uint256 totalSupply );
855     event LogRebase( uint256 indexed epoch, uint256 rebase, uint256 index );
856     event LogStakingContractUpdated( address stakingContract );
857     event LogSetIndex(uint256 indexed index );
858
859     struct Rebase {
860         uint epoch;
861         uint rebase; // 18 decimals
862         uint totalStakedBefore;
863         uint totalStakedAfter;
864         uint amountRebased;
865         uint index;
866         uint32 timeOccured;
867     }
868     Rebase[] public rebases;
869
870     uint public INDEX;
871
872     uint256 private constant MAX_UINT256 = ~uint256(0);
873     uint256 private constant INITIAL_FRAGMENTS_SUPPLY = 5000000 * 10**9;
874
875     // TOTAL_GONS is a multiple of INITIAL_FRAGMENTS_SUPPLY so that _gonsPerFragment is an integer.
876     // Use the highest value that fits in a uint256 for max granularity.
877     uint256 private constant TOTAL_GONS = MAX_UINT256 - (MAX_UINT256 % INITIAL_FRAGMENTS_SUPPLY);
878
879     // MAX_SUPPLY = maximum integer < (sqrt(4*TOTAL_GONS + 1) - 1) / 2
880     uint256 private constant MAX_SUPPLY = ~uint256(0); // (2^128) - 1
881

```

```

827         require(msg.sender == _pendingOwner, "Ownable: caller != pending owner");
828
829         // Effects
830         emit OwnershipTransferred(owner, _pendingOwner);
831
832         owner = _pendingOwner;
833         pendingOwner = address(0);
834     }
835     /// @notice Only allows the `owner` to execute the function.
836     modifier onlyOwner() {
837         require(msg.sender == owner, "Ownable: caller is not the owner");
838         _;
839     }
840 }
841
842 contract sMaia is ERC20Permit, Ownable {
843
844     using LowGasSafeMath for uint256;
845
846     modifier onlyStakingContract() {
847         require( msg.sender == stakingContract, "Ownable: caller is not the owner");
848         _;
849     }
850
851     address public stakingContract;
852     address public initializer;
853
854     event LogSupply(uint256 indexed epoch, uint256 timestamp, uint256 totalSupply );
855     event LogRebase( uint256 indexed epoch, uint256 rebase, uint256 index );
856     event LogStakingContractUpdated( address stakingContract );
857     event LogSetIndex(uint256 indexed index );
858
859     struct Rebase {
860         uint epoch;
861         uint rebase; // 18 decimals
862         uint totalStakedBefore;
863         uint totalStakedAfter;
864         uint amountRebased;
865         uint index;
866         uint32 timeOccured;
867     }
868     Rebase[] public rebases;
869
870     uint public INDEX;
871
872     uint256 private constant MAX_UINT256 = ~uint256(0);
873     uint256 private constant INITIAL_FRAGMENTS_SUPPLY = 5000000 * 10**9;
874
875     // TOTAL_GONS is a multiple of INITIAL_FRAGMENTS_SUPPLY so that _gonsPerFragment is an integer.
876     // Use the highest value that fits in a uint256 for max granularity.
877     uint256 private constant TOTAL_GONS = MAX_UINT256 - (MAX_UINT256 % INITIAL_FRAGMENTS_SUPPLY);
878
879     // MAX_SUPPLY = maximum integer < (sqrt(4*TOTAL_GONS + 1) - 1) / 2
880     uint256 private constant MAX_SUPPLY = ~uint256(0); // (2^128) - 1
881

```

```

882     uint256 private _gonsPerFragment;
883     mapping(address => uint256) private _gonBalanc
es;
884
885     mapping ( address => mapping ( address => uint
256 ) ) private _allowedValue;
886
887     constructor() ERC20("MEM0ries", "MEM0", 9) ERC
20Permit() {
888         initializer = msg.sender;
889         _totalSupply = INITIAL_FRAGMENTS_SUPPLY;
890         _gonsPerFragment = TOTAL_GONS.div(_totalSu
pply);
891     }
892
893     function initialize( address stakingContract_
) external returns ( bool ) {
894         require( msg.sender == initializer, "NA"
);
895         require( stakingContract_ != address(0),
"IA" );
896         stakingContract = stakingContract_;
897         _gonBalances[ stakingContract ] = TOTAL_GO
NS;
898
899         emit Transfer( address(0x0), stakingContra
ct, _totalSupply );
900         emit LogStakingContractUpdated( stakingCon
tract_ );
901
902         initializer = address(0);
903         return true;
904     }
905
906     function setIndex( uint _INDEX ) external only
Owner() {
907         require( INDEX == 0, "INZ");
908         INDEX = gonsForBalance( _INDEX );
909         emit LogSetIndex(INDEX);
910     }
911
912     /**
913     @notice increases MEM0ries supply to incre
ase staking balances relative to profit_
914     @param profit_ uint256
915     @return uint256
916     */
917     function rebase( uint256 profit_, uint epoch_
) public onlyStakingContract() returns ( uint256
) {
918         uint256 rebaseAmount;
919         uint256 circulatingSupply_ = circulatingSu
pply();
920
921         if ( profit_ == 0 ) {
922             emit LogSupply( epoch_, block.timestam
p, _totalSupply );
923             emit LogRebase( epoch_, 0, index() );
924             return _totalSupply;
925         } else if ( circulatingSupply_ > 0 ){
926             rebaseAmount = profit_.mul( _totalSupp
ly ).div( circulatingSupply_ );
927         } else {
928             rebaseAmount = profit_;
929         }
930
931         _totalSupply = _totalSupply.add( rebaseAmo
unt );
932

```

```

882     uint256 private _gonsPerFragment;
883     mapping(address => uint256) private _gonBalanc
es;
884
885     mapping ( address => mapping ( address => uint
256 ) ) private _allowedValue;
886
887     constructor() ERC20("Staked Maia", "MAIA", 9)
ERC20Permit() {
888         initializer = msg.sender;
889         _totalSupply = INITIAL_FRAGMENTS_SUPPLY;
890         _gonsPerFragment = TOTAL_GONS.div(_totalSu
pply);
891     }
892
893     function initialize( address stakingContract_
) external returns ( bool ) {
894         require( msg.sender == initializer, "NA"
);
895         require( stakingContract_ != address(0),
"IA" );
896         stakingContract = stakingContract_;
897         _gonBalances[ stakingContract ] = TOTAL_GO
NS;
898
899         emit Transfer( address(0x0), stakingContra
ct, _totalSupply );
900         emit LogStakingContractUpdated( stakingCon
tract_ );
901
902         initializer = address(0);
903         return true;
904     }
905
906     function setIndex( uint _INDEX ) external only
Owner() {
907         require( INDEX == 0, "INZ");
908         INDEX = gonsForBalance( _INDEX );
909         emit LogSetIndex(INDEX);
910     }
911
912     /**
913     @notice increases MEM0ries supply to incre
ase staking balances relative to profit_
914     @param profit_ uint256
915     @return uint256
916     */
917     function rebase( uint256 profit_, uint epoch_
) public onlyStakingContract() returns ( uint256
) {
918         uint256 rebaseAmount;
919         uint256 circulatingSupply_ = circulatingSu
pply();
920
921         if ( profit_ == 0 ) {
922             emit LogSupply( epoch_, block.timestam
p, _totalSupply );
923             emit LogRebase( epoch_, 0, index() );
924             return _totalSupply;
925         } else if ( circulatingSupply_ > 0 ){
926             rebaseAmount = profit_.mul( _totalSupp
ly ).div( circulatingSupply_ );
927         } else {
928             rebaseAmount = profit_;
929         }
930
931         _totalSupply = _totalSupply.add( rebaseAmo
unt );
932

```

```

933     if ( _totalSupply > MAX_SUPPLY ) {
934         _totalSupply = MAX_SUPPLY;
935     }
936
937     _gonsPerFragment = TOTAL_GONS.div( _totalSupply );
938
939     _storeRebase( circulatingSupply_, profit_,
940 epoch_ );
941
942     return _totalSupply;
943 }
944 /**
945  @notice emits event with data about rebase
946  @param previousCirculating_ uint
947  @param profit_ uint
948  @param epoch_ uint
949  @return bool
950  */
951  function _storeRebase( uint previousCirculating_, uint profit_, uint epoch_ ) internal returns ( bool ) {
952      uint rebasePercent = profit_.mul( 1e18 ).div( previousCirculating_ );
953
954      rebases.push( Rebase ( {
955          epoch: epoch_,
956          rebase: rebasePercent, // 18 decimals
957          totalStakedBefore: previousCirculating_
958      },
959          totalStakedAfter: circulatingSupply(),
960          amountRebased: profit_,
961          index: index(),
962          timeOccured: uint32(block.timestamp)
963      ) ));
964
965      emit LogSupply( epoch_, block.timestamp, _totalSupply );
966      emit LogRebase( epoch_, rebasePercent, index() );
967
968      return true;
969  }
970
971  function balanceOf( address who ) public view override returns ( uint256 ) {
972      return _gonBalances[ who ].div( _gonsPerFragment );
973  }
974
975  function gonsForBalance( uint amount ) public view returns ( uint ) {
976      return amount.mul( _gonsPerFragment );
977  }
978
979  function balanceForGons( uint gons ) public view returns ( uint ) {
980      return gons.div( _gonsPerFragment );
981  }
982
983  // Staking contract holds excess MEMORIES
984  function circulatingSupply() public view returns ( uint ) {
985      return _totalSupply.sub( balanceOf( stakingContract ) );
986  }
987
988  function index() public view returns ( uint ) {

```

```

933     if ( _totalSupply > MAX_SUPPLY ) {
934         _totalSupply = MAX_SUPPLY;
935     }
936
937     _gonsPerFragment = TOTAL_GONS.div( _totalSupply );
938
939     _storeRebase( circulatingSupply_, profit_,
940 epoch_ );
941
942     return _totalSupply;
943 }
944 /**
945  @notice emits event with data about rebase
946  @param previousCirculating_ uint
947  @param profit_ uint
948  @param epoch_ uint
949  @return bool
950  */
951  function _storeRebase( uint previousCirculating_, uint profit_, uint epoch_ ) internal returns ( bool ) {
952      uint rebasePercent = profit_.mul( 1e18 ).div( previousCirculating_ );
953
954      rebases.push( Rebase ( {
955          epoch: epoch_,
956          rebase: rebasePercent, // 18 decimals
957          totalStakedBefore: previousCirculating_
958      },
959          totalStakedAfter: circulatingSupply(),
960          amountRebased: profit_,
961          index: index(),
962          timeOccured: uint32(block.timestamp)
963      ) ));
964
965      emit LogSupply( epoch_, block.timestamp, _totalSupply );
966      emit LogRebase( epoch_, rebasePercent, index() );
967
968      return true;
969  }
970
971  function balanceOf( address who ) public view override returns ( uint256 ) {
972      return _gonBalances[ who ].div( _gonsPerFragment );
973  }
974
975  function gonsForBalance( uint amount ) public view returns ( uint ) {
976      return amount.mul( _gonsPerFragment );
977  }
978
979  function balanceForGons( uint gons ) public view returns ( uint ) {
980      return gons.div( _gonsPerFragment );
981  }
982
983  // Staking contract holds excess MEMORIES
984  function circulatingSupply() public view returns ( uint ) {
985      return _totalSupply.sub( balanceOf( stakingContract ) );
986  }
987
988  function index() public view returns ( uint ) {

```

```

988         return balanceForGons( INDEX );
989     }
990
991     function transfer( address to, uint256 value )
    public override returns (bool) {
992         uint256 gonValue = value.mul( _gonsPerFrag
    ment );
993         _gonBalances[ msg.sender ] = _gonBalances[
    msg.sender ].sub( gonValue );
994         _gonBalances[ to ] = _gonBalances[ to ].ad
    d( gonValue );
995         emit Transfer( msg.sender, to, value );
996         return true;
997     }
998
999     function allowance( address owner_, address sp
    ender ) public view override returns ( uint256 ) {
1000         return _allowedValue[ owner_ ][ spender ];
1001     }
1002
1003     function transferFrom( address from, address t
    o, uint256 value ) public override returns ( bool
    ) {
1004         _allowedValue[ from ][ msg.sender ] = _allo
    wedValue[ from ][ msg.sender ].sub( value );
1005         emit Approval( from, msg.sender, _allowedV
    alue[ from ][ msg.sender ] );
1006
1007         uint256 gonValue = gonsForBalance( value
    );
1008         _gonBalances[ from ] = _gonBalances[from].
    sub( gonValue );
1009         _gonBalances[ to ] = _gonBalances[to].add(
    gonValue );
1010         emit Transfer( from, to, value );
1011
1012         return true;
1013     }
1014
1015     function approve( address spender, uint256 val
    ue ) public override returns (bool) {
1016         _allowedValue[ msg.sender ][ spender ] =
    value;
1017         emit Approval( msg.sender, spender, value
    );
1018         return true;
1019     }
1020
1021     // What gets called in a permit
1022     function _approve( address owner, address spen
    der, uint256 value ) internal override virtual {
1023         _allowedValue[owner][spender] = value;
1024         emit Approval( owner, spender, value );
1025     }
1026
1027     function increaseAllowance( address spender, u
    int256 addedValue ) public override returns (bool)
    {
1028         _allowedValue[ msg.sender ][ spender ] = _
    allowedValue[ msg.sender ][ spender ].add( addedVa
    lue );
1029         emit Approval( msg.sender, spender, _allow
    edValue[ msg.sender ][ spender ] );
1030         return true;
1031     }
1032
1033     function decreaseAllowance( address spender, u
    int256 subtractedValue ) public override returns
    (bool) {

```

```

988         return balanceForGons( INDEX );
989     }
990
991     function transfer( address to, uint256 value )
    public override returns (bool) {
992         uint256 gonValue = value.mul( _gonsPerFrag
    ment );
993         _gonBalances[ msg.sender ] = _gonBalances[
    msg.sender ].sub( gonValue );
994         _gonBalances[ to ] = _gonBalances[ to ].ad
    d( gonValue );
995         emit Transfer( msg.sender, to, value );
996         return true;
997     }
998
999     function allowance( address owner_, address sp
    ender ) public view override returns ( uint256 ) {
1000         return _allowedValue[ owner_ ][ spender ];
1001     }
1002
1003     function transferFrom( address from, address t
    o, uint256 value ) public override returns ( bool
    ) {
1004         _allowedValue[ from ][ msg.sender ] = _allo
    wedValue[ from ][ msg.sender ].sub( value );
1005         emit Approval( from, msg.sender, _allowedV
    alue[ from ][ msg.sender ] );
1006
1007         uint256 gonValue = gonsForBalance( value
    );
1008         _gonBalances[ from ] = _gonBalances[from].
    sub( gonValue );
1009         _gonBalances[ to ] = _gonBalances[to].add(
    gonValue );
1010         emit Transfer( from, to, value );
1011
1012         return true;
1013     }
1014
1015     function approve( address spender, uint256 val
    ue ) public override returns (bool) {
1016         _allowedValue[ msg.sender ][ spender ] =
    value;
1017         emit Approval( msg.sender, spender, value
    );
1018         return true;
1019     }
1020
1021     // What gets called in a permit
1022     function _approve( address owner, address spen
    der, uint256 value ) internal override virtual {
1023         _allowedValue[owner][spender] = value;
1024         emit Approval( owner, spender, value );
1025     }
1026
1027     function increaseAllowance( address spender, u
    int256 addedValue ) public override returns (bool)
    {
1028         _allowedValue[ msg.sender ][ spender ] = _
    allowedValue[ msg.sender ][ spender ].add( addedVa
    lue );
1029         emit Approval( msg.sender, spender, _allow
    edValue[ msg.sender ][ spender ] );
1030         return true;
1031     }
1032
1033     function decreaseAllowance( address spender, u
    int256 subtractedValue ) public override returns
    (bool) {

```

```
1034         uint256 oldValue = _allowedValue[ msg.sender ][ spender ];
1035         if (subtractedValue >= oldValue) {
1036             _allowedValue[ msg.sender ][ spender ]
1037             = 0;
1038         } else {
1039             _allowedValue[ msg.sender ][ spender ]
1040             = oldValue.sub( subtractedValue );
1041         }
1042         emit Approval( msg.sender, spender, _allowedValue[ msg.sender ][ spender ] );
1043         return true;
1044     }
1045 }
```

```
1034         uint256 oldValue = _allowedValue[ msg.sender ][ spender ];
1035         if (subtractedValue >= oldValue) {
1036             _allowedValue[ msg.sender ][ spender ]
1037             = 0;
1038         } else {
1039             _allowedValue[ msg.sender ][ spender ]
1040             = oldValue.sub( subtractedValue );
1041         }
1042         emit Approval( msg.sender, spender, _allowedValue[ msg.sender ][ spender ] );
1043         return true;
1044     }
1045 }
```

>