```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "@openzeppelin/contracts@4.7.3/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts@4.7.3/token/ERC20/extensions/ERC20Burnable.sol";
import "@openzeppelin/contracts@4.7.3/token/ERC20/extensions/ERC20Snapshot.sol";
import "@openzeppelin/contracts@4.7.3/access/Ownable.sol";
import "@openzeppelin/contracts@4.7.3/token/ERC20/extensions/draft-ERC20Permit.sol";
import "@openzeppelin/contracts@4.7.3/token/ERC20/extensions/ERC20Votes.sol";

contract NeoEconomyCoin is ERC20, ERC20Burnable, ERC20Snapshot, Ownable, ERC20Permit,
ERC20Votes {
    constructor()
        ERC20("Neo Economy Coin", "NEC")
        ERC20Permit("Neo Economy Coin")
    {
        _mint(msg.sender, 4200000000 * 10 ** decimals());
    }

    function snapshot() public onlyOwner {
        _snapshot();
    }

    // The following functions are overrides required by Solidity.

    function _beforeTokenTransfer(address from, address to, uint256 amount)
        internal
        override(ERC20, ERC20Snapshot)
    {
        super._beforeTokenTransfer(from, to, amount);
    }
```

```
    function _afterTokenTransfer(address from, address to, uint256 amount)
        internal
        override(ERC20, ERC20Votes)
    {
        super._afterTokenTransfer(from, to, amount);
    }

    function _mint(address to, uint256 amount)
        internal
        override(ERC20, ERC20Votes)
    {
        super._mint(to, amount);
    }

    function _burn(address account, uint256 amount)
        internal
        override(ERC20, ERC20Votes)
    {
        super._burn(account, amount);
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.5.0) (token/ERC20/extensions/ERC20Votes.sol)

pragma solidity ^0.8.0;

import "./draft-ERC20Permit.sol";
import "../../../utils/math/Math.sol";
import "../../../governance/utils/IVotes.sol";
import "../../../utils/math/SafeCast.sol";
```

```
import "../../../utils/cryptography/ECDSA.sol";


/**
 * @dev Extension of ERC20 to support Compound-like voting and delegation. This version is more generic than Compound's,
 * and supports token supply up to 2^224^ - 1, while COMP is limited to 2^96^ - 1.
 *
 * NOTE: If exact COMP compatibility is required, use the {ERC20VotesComp} variant of this module.
 *
 * This extension keeps a history (checkpoints) of each account's vote power. Vote power can be delegated either
 * by calling the {delegate} function directly, or by providing a signature to be used with {delegateBySig}. Voting
 * power can be queried through the public accessors {getVotes} and {getPastVotes}.
 *
 * By default, token balance does not account for voting power. This makes transfers cheaper. The downside is that it
 * requires users to delegate to themselves in order to activate checkpoints and have their voting power tracked.
 *
 * _Available since v4.2._
 */
abstract contract ERC20Votes is IVotes, ERC20Permit {
    struct Checkpoint {
        uint32 fromBlock;
        uint224 votes;
    }

    bytes32 private constant _DELEGATION_TYPEHASH =
        keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

    mapping(address => address) private _delegates;
    mapping(address => Checkpoint[]) private _checkpoints;
```

```
    Checkpoint[] private _totalSupplyCheckpoints;

    /**
     * @dev Get the `pos`-th checkpoint for `account`.
     */
    function checkpoints(address account, uint32 pos) public view virtual returns (Checkpoint memory) {
        return _checkpoints[account][pos];
    }

    /**
     * @dev Get number of checkpoints for `account`.
     */
    function numCheckpoints(address account) public view virtual returns (uint32) {
        return SafeCast.toUint32(_checkpoints[account].length);
    }

    /**
     * @dev Get the address `account` is currently delegating to.
     */
    function delegates(address account) public view virtual override returns (address) {
        return _delegates[account];
    }

    /**
     * @dev Gets the current votes balance for `account`
     */
    function getVotes(address account) public view virtual override returns (uint256) {
        uint256 pos = _checkpoints[account].length;
        return pos == 0 ? 0 : _checkpoints[account][pos - 1].votes;
    }
```

```
    /**
     * @dev Retrieve the number of votes for `account` at the end of `blockNumber`.
     *
     * Requirements:
     *
     * - `blockNumber` must have been already mined
     */
    function getPastVotes(address account, uint256 blockNumber) public view virtual override returns (uint256) {
        require(blockNumber < block.number, "ERC20Votes: block not yet mined");
        return _checkpointsLookup(_checkpoints[account], blockNumber);
    }

    /**
     * @dev Retrieve the `totalSupply` at the end of `blockNumber`. Note, this value is the sum of all balances.
     * It is but NOT the sum of all the delegated votes!
     *
     * Requirements:
     *
     * - `blockNumber` must have been already mined
     */
    function getPastTotalSupply(uint256 blockNumber) public view virtual override returns (uint256) {
        require(blockNumber < block.number, "ERC20Votes: block not yet mined");
        return _checkpointsLookup(_totalSupplyCheckpoints, blockNumber);
    }

    /**
     * @dev Lookup a value in a list of (sorted) checkpoints.
     */
```

```
  function _checkpointsLookup(Checkpoint[] storage ckpts, uint256 blockNumber) private view returns (uint256) {
      // We run a binary search to look for the earliest checkpoint taken after `blockNumber`.
      //
      // During the loop, the index of the wanted checkpoint remains in the range [low-1, high).
      // With each iteration, either `low` or `high` is moved towards the middle of the range to maintain the invariant.
      // - If the middle checkpoint is after `blockNumber`, we look in [low, mid)
      // - If the middle checkpoint is before or equal to `blockNumber`, we look in [mid+1, high)
      // Once we reach a single value (when low == high), we've found the right checkpoint at the index high-1, if not
      // out of bounds (in which case we're looking too far in the past and the result is 0).
      // Note that if the latest checkpoint available is exactly for `blockNumber`, we end up with an index that is
      // past the end of the array, so we technically don't find a checkpoint after `blockNumber`, but it works out
      // the same.
      uint256 high = ckpts.length;
      uint256 low = 0;
      while (low < high) {
          uint256 mid = Math.average(low, high);
          if (ckpts[mid].fromBlock > blockNumber) {
              high = mid;
          } else {
              low = mid + 1;
          }
      }

      return high == 0 ? 0 : ckpts[high - 1].votes;
  }

  /**
   * @dev Delegate votes from the sender to `delegatee`.
```

```
     */
    function delegate(address delegatee) public virtual override {
        _delegate(_msgSender(), delegatee);
    }

    /**
     * @dev Delegates votes from signer to `delegatee`
     */
    function delegateBySig(
        address delegatee,
        uint256 nonce,
        uint256 expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public virtual override {
        require(block.timestamp <= expiry, "ERC20Votes: signature expired");
        address signer = ECDSA.recover(
            _hashTypedDataV4(keccak256(abi.encode(_DELEGATION_TYPEHASH, delegatee, nonce,
expiry))),
            v,
            r,
            s
        );
        require(nonce == _useNonce(signer), "ERC20Votes: invalid nonce");
        _delegate(signer, delegatee);
    }

    /**
     * @dev Maximum token supply. Defaults to `type(uint224).max` (2^224^ - 1).
     */
```

```
function _maxSupply() internal view virtual returns (uint224) {
    return type(uint224).max;
}

/**
 * @dev Snapshots the totalSupply after it has been increased.
 */
function _mint(address account, uint256 amount) internal virtual override {
    super._mint(account, amount);
    require(totalSupply() <= _maxSupply(), "ERC20Votes: total supply risks overflowing votes");

    _writeCheckpoint(_totalSupplyCheckpoints, _add, amount);
}

/**
 * @dev Snapshots the totalSupply after it has been decreased.
 */
function _burn(address account, uint256 amount) internal virtual override {
    super._burn(account, amount);

    _writeCheckpoint(_totalSupplyCheckpoints, _subtract, amount);
}

/**
 * @dev Move voting power when tokens are transferred.
 *
 * Emits a {DelegateVotesChanged} event.
 */
function _afterTokenTransfer(
    address from,
    address to,
```

```
        uint256 amount
    ) internal virtual override {
        super._afterTokenTransfer(from, to, amount);

        _moveVotingPower(delegates(from), delegates(to), amount);
    }

    /**
     * @dev Change delegation for `delegator` to `delegatee`.
     *
     * Emits events {DelegateChanged} and {DelegateVotesChanged}.
     */
    function _delegate(address delegator, address delegatee) internal virtual {
        address currentDelegate = delegates(delegator);
        uint256 delegatorBalance = balanceOf(delegator);
        _delegates[delegator] = delegatee;

        emit DelegateChanged(delegator, currentDelegate, delegatee);

        _moveVotingPower(currentDelegate, delegatee, delegatorBalance);
    }

    function _moveVotingPower(
        address src,
        address dst,
        uint256 amount
    ) private {
        if (src != dst && amount > 0) {
            if (src != address(0)) {
                (uint256 oldWeight, uint256 newWeight) = _writeCheckpoint(_checkpoints[src], _subtract,
amount);
```

```
            emit DelegateVotesChanged(src, oldWeight, newWeight);
        }

        if (dst != address(0)) {
            (uint256 oldWeight, uint256 newWeight) = _writeCheckpoint(_checkpoints[dst], _add,
amount);
            emit DelegateVotesChanged(dst, oldWeight, newWeight);
        }
    }
  }

  function _writeCheckpoint(
    Checkpoint[] storage ckpts,
    function(uint256, uint256) view returns (uint256) op,
    uint256 delta
  ) private returns (uint256 oldWeight, uint256 newWeight) {
    uint256 pos = ckpts.length;
    oldWeight = pos == 0 ? 0 : ckpts[pos - 1].votes;
    newWeight = op(oldWeight, delta);

    if (pos > 0 && ckpts[pos - 1].fromBlock == block.number) {
      ckpts[pos - 1].votes = SafeCast.toUint224(newWeight);
    } else {
      ckpts.push(Checkpoint({fromBlock: SafeCast.toUint32(block.number), votes:
SafeCast.toUint224(newWeight)}));
    }
  }

  function _add(uint256 a, uint256 b) private pure returns (uint256) {
    return a + b;
  }
```

```
    function _subtract(uint256 a, uint256 b) private pure returns (uint256) {
        return a - b;
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.6.0) (token/ERC20/extensions/draft-ERC20Permit.sol)

pragma solidity ^0.8.0;

import "./draft-IERC20Permit.sol";
import "../ERC20.sol";
import "../../../utils/cryptography/draft-EIP712.sol";
import "../../../utils/cryptography/ECDSA.sol";
import "../../../utils/Counters.sol";

/**
 * @dev Implementation of the ERC20 Permit extension allowing approvals to be made via signatures, as defined in
 * https://eips.ethereum.org/EIPS/eip-2612[EIP-2612].
 *
 * Adds the {permit} method, which can be used to change an account's ERC20 allowance (see {IERC20-allowance}) by
 * presenting a message signed by the account. By not relying on `{IERC20-approve}`, the token holder account doesn't
 * need to send a transaction, and thus is not required to hold Ether at all.
 *
 * _Available since v3.4._
 */
abstract contract ERC20Permit is ERC20, IERC20Permit, EIP712 {
    using Counters for Counters.Counter;

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

```
    // solhint-disable-next-line var-name-mixedcase
    bytes32 private constant _PERMIT_TYPEHASH =
        keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    /**
     * @dev In previous versions `_PERMIT_TYPEHASH` was declared as `immutable`.
     * However, to ensure consistency with the upgradeable transpiler, we will continue
     * to reserve a slot.
     * @custom:oz-renamed-from _PERMIT_TYPEHASH
     */
    // solhint-disable-next-line var-name-mixedcase
    bytes32 private _PERMIT_TYPEHASH_DEPRECATED_SLOT;

    /**
     * @dev Initializes the {EIP712} domain separator using the `name` parameter, and setting `version` to `"1"`.
     *
     * It's a good idea to use the same `name` that is defined as the ERC20 token name.
     */
    constructor(string memory name) EIP712(name, "1") {}

    /**
     * @dev See {IERC20Permit-permit}.
     */
    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
```

```
        bytes32 s
    ) public virtual override {
        require(block.timestamp <= deadline, "ERC20Permit: expired deadline");

        bytes32 structHash = keccak256(abi.encode(_PERMIT_TYPEHASH, owner, spender, value,
_useNonce(owner), deadline));

        bytes32 hash = _hashTypedDataV4(structHash);

        address signer = ECDSA.recover(hash, v, r, s);
        require(signer == owner, "ERC20Permit: invalid signature");

        _approve(owner, spender, value);
    }

    /**
     * @dev See {IERC20Permit-nonces}.
     */
    function nonces(address owner) public view virtual override returns (uint256) {
        return _nonces[owner].current();
    }

    /**
     * @dev See {IERC20Permit-DOMAIN_SEPARATOR}.
     */
    // solhint-disable-next-line func-name-mixedcase
    function DOMAIN_SEPARATOR() external view override returns (bytes32) {
        return _domainSeparatorV4();
    }

    /**
```

```
     * @dev "Consume a nonce": return the current value and increment.
     *
     * _Available since v4.1._
     */
    function _useNonce(address owner) internal virtual returns (uint256 current) {
        Counters.Counter storage nonce = _nonces[owner];
        current = nonce.current();
        nonce.increment();
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (access/Ownable.sol)

pragma solidity ^0.8.0;

import "../utils/Context.sol";

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;
```

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

/**
 * @dev Initializes the contract setting the deployer as the initial owner.
 */
constructor() {
    _transferOwnership(_msgSender());
}

/**
 * @dev Throws if called by any account other than the owner.
 */
modifier onlyOwner() {
    _checkOwner();
    _;
}

/**
 * @dev Returns the address of the current owner.
 */
function owner() public view virtual returns (address) {
    return _owner;
}

/**
 * @dev Throws if the sender is not the owner.
 */
function _checkOwner() internal view virtual {
    require(owner() == _msgSender(), "Ownable: caller is not the owner");
}
```

```
    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        _transferOwnership(address(0));
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}
```

```
// SPDX-License-Identifier: MIT

// OpenZeppelin Contracts (last updated v4.7.0) (token/ERC20/extensions/ERC20Snapshot.sol)


pragma solidity ^0.8.0;


import "../ERC20.sol";

import "../../../utils/Arrays.sol";

import "../../../utils/Counters.sol";


/**

 * @dev This contract extends an ERC20 token with a snapshot mechanism. When a snapshot is created, the balances and

 * total supply at the time are recorded for later access.

 *

 * This can be used to safely create mechanisms based on token balances such as trustless dividends or weighted voting.

 * In naive implementations it's possible to perform a "double spend" attack by reusing the same balance from different

 * accounts. By using snapshots to calculate dividends or voting power, those attacks no longer apply. It can also be

 * used to create an efficient ERC20 forking mechanism.

 *

 * Snapshots are created by the internal {_snapshot} function, which will emit the {Snapshot} event and return a

 * snapshot id. To get the total supply at the time of a snapshot, call the function {totalSupplyAt} with the snapshot

 * id. To get the balance of an account at the time of a snapshot, call the {balanceOfAt} function with the snapshot id

 * and the account address.

 *

 * NOTE: Snapshot policy can be customized by overriding the {_getCurrentSnapshotId} method. For example, having it

 * return `block.number` will trigger the creation of snapshot at the beginning of each new block. When overriding this
```

```
 * function, be careful about the monotonicity of its result. Non-monotonic snapshot ids will break the contract.
 *
 * Implementing snapshots for every block using this method will incur significant gas costs. For a gas-efficient
 * alternative consider {ERC20Votes}.
 *
 * ==== Gas Costs
 *
 * Snapshots are efficient. Snapshot creation is _O(1)_. Retrieval of balances or total supply from a snapshot is _O(log
 * n)_ in the number of snapshots that have been created, although _n_ for a specific account will generally be much
 * smaller since identical balances in subsequent snapshots are stored as a single entry.
 *
 * There is a constant overhead for normal ERC20 transfers due to the additional snapshot bookkeeping. This overhead is
 * only significant for the first transfer that immediately follows a snapshot for a particular account. Subsequent
 * transfers will have normal cost until the next snapshot, and so on.
 */

abstract contract ERC20Snapshot is ERC20 {
    // Inspired by Jordi Baylina's MiniMeToken to record historical balances:
    // https://github.com/Giveth/minime/blob/ea04d950eea153a04c51fa510b068b9dded390cb/contracts/MiniMeToken.sol

    using Arrays for uint256[];
    using Counters for Counters.Counter;

    // Snapshotted values have arrays of ids and the value corresponding to that id. These could be an array of a
    // Snapshot struct, but that would impede usage of functions that work on an array.
```

```
    struct Snapshots {
        uint256[] ids;
        uint256[] values;
    }

    mapping(address => Snapshots) private _accountBalanceSnapshots;
    Snapshots private _totalSupplySnapshots;

    // Snapshot ids increase monotonically, with the first value being 1. An id of 0 is invalid.
    Counters.Counter private _currentSnapshotId;

    /**
     * @dev Emitted by {_snapshot} when a snapshot identified by `id` is created.
     */
    event Snapshot(uint256 id);

    /**
     * @dev Creates a new snapshot and returns its snapshot id.
     *
     * Emits a {Snapshot} event that contains the same id.
     *
     * {_snapshot} is `internal` and you have to decide how to expose it externally. Its usage may be restricted to a
     * set of accounts, for example using {AccessControl}, or it may be open to the public.
     *
     * [WARNING]
     * ====
     * While an open way of calling {_snapshot} is required for certain trust minimization mechanisms such as forking,
     * you must consider that it can potentially be used by attackers in two ways.
     *
```

```
    * First, it can be used to increase the cost of retrieval of values from snapshots, although it will grow
    * logarithmically thus rendering this attack ineffective in the long term. Second, it can be used to target
    * specific accounts and increase the cost of ERC20 transfers for them, in the ways specified in the Gas Costs
    * section above.
    *
    * We haven't measured the actual numbers; if this is something you're interested in please reach out to us.
    * ====
    */
   function _snapshot() internal virtual returns (uint256) {
       _currentSnapshotId.increment();

       uint256 currentId = _getCurrentSnapshotId();
       emit Snapshot(currentId);
       return currentId;
   }

   /**
    * @dev Get the current snapshotId
    */
   function _getCurrentSnapshotId() internal view virtual returns (uint256) {
       return _currentSnapshotId.current();
   }

   /**
    * @dev Retrieves the balance of `account` at the time `snapshotId` was created.
    */
   function balanceOfAt(address account, uint256 snapshotId) public view virtual returns (uint256) {
       (bool snapshotted, uint256 value) = _valueAt(snapshotId, _accountBalanceSnapshots[account]);
```

```
        return snapshotted ? value : balanceOf(account);
    }

    /**
     * @dev Retrieves the total supply at the time `snapshotId` was created.
     */
    function totalSupplyAt(uint256 snapshotId) public view virtual returns (uint256) {
        (bool snapshotted, uint256 value) = _valueAt(snapshotId, _totalSupplySnapshots);

        return snapshotted ? value : totalSupply();
    }

    // Update balance and/or total supply snapshots before the values are modified. This is
implemented
    // in the _beforeTokenTransfer hook, which is executed for _mint, _burn, and _transfer
operations.
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        super._beforeTokenTransfer(from, to, amount);

        if (from == address(0)) {
            // mint
            _updateAccountSnapshot(to);
            _updateTotalSupplySnapshot();
        } else if (to == address(0)) {
            // burn
            _updateAccountSnapshot(from);
            _updateTotalSupplySnapshot();
```

```
        } else {
            // transfer
            _updateAccountSnapshot(from);
            _updateAccountSnapshot(to);
        }
    }

    function _valueAt(uint256 snapshotId, Snapshots storage snapshots) private view returns (bool, uint256) {
        require(snapshotId > 0, "ERC20Snapshot: id is 0");
        require(snapshotId <= _getCurrentSnapshotId(), "ERC20Snapshot: nonexistent id");

        // When a valid snapshot is queried, there are three possibilities:
        //  a) The queried value was not modified after the snapshot was taken. Therefore, a snapshot entry was never
        //  created for this id, and all stored snapshot ids are smaller than the requested one. The value that corresponds
        //  to this id is the current one.
        //  b) The queried value was modified after the snapshot was taken. Therefore, there will be an entry with the
        //  requested id, and its value is the one to return.
        //  c) More snapshots were created after the requested one, and the queried value was later modified. There will be
        //  no entry for the requested id: the value that corresponds to it is that of the smallest snapshot id that is
        //  larger than the requested one.
        //
        // In summary, we need to find an element in an array, returning the index of the smallest value that is larger if
        // it is not found, unless said value doesn't exist (e.g. when all values are smaller). Arrays.findUpperBound does
        // exactly this.

        uint256 index = snapshots.ids.findUpperBound(snapshotId);
```

```
    if (index == snapshots.ids.length) {
        return (false, 0);
    } else {
        return (true, snapshots.values[index]);
    }
}

function _updateAccountSnapshot(address account) private {
    _updateSnapshot(_accountBalanceSnapshots[account], balanceOf(account));
}

function _updateTotalSupplySnapshot() private {
    _updateSnapshot(_totalSupplySnapshots, totalSupply());
}

function _updateSnapshot(Snapshots storage snapshots, uint256 currentValue) private {
    uint256 currentId = _getCurrentSnapshotId();
    if (_lastSnapshotId(snapshots.ids) < currentId) {
        snapshots.ids.push(currentId);
        snapshots.values.push(currentValue);
    }
}

function _lastSnapshotId(uint256[] storage ids) private view returns (uint256) {
    if (ids.length == 0) {
        return 0;
    } else {
        return ids[ids.length - 1];
    }
}
```

```
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.5.0) (token/ERC20/extensions/ERC20Burnable.sol)

pragma solidity ^0.8.0;

import "../ERC20.sol";
import "../../../utils/Context.sol";

/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
abstract contract ERC20Burnable is Context, ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
     * Requirements:
```

```
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
     */
    function burnFrom(address account, uint256 amount) public virtual {
        _spendAllowance(account, _msgSender(), amount);
        _burn(account, amount);
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (token/ERC20/ERC20.sol)

pragma solidity ^0.8.0;

import "./IERC20.sol";
import "./extensions/IERC20Metadata.sol";
import "../../utils/Context.sol";

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions revert
 * instead returning `false` on failure. This behavior is nonetheless
```

```
 * conventional and does not conflict with the expectations of ERC20
 * applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The default value of {decimals} is 18. To select a different value for
     * {decimals} you should overload it.
     *
     * All two of these values are immutable: they can only be set once during
     * construction.
     */
```

```
constructor(string memory name_, string memory symbol_) {
    _name = name_;
    _symbol = symbol_;
}

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

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public view virtual override returns (string memory) {
    return _symbol;
}

/**
 * @dev Returns the number of decimals used to get its user representation.
 * For example, if `decimals` equals `2`, a balance of `505` tokens should
 * be displayed to a user as `5.05` (`505 / 10 ** 2`).
 *
 * Tokens usually opt for a value of 18, imitating the relationship between
 * Ether and Wei. This is the value {ERC20} uses, unless this function is
 * overridden;
 *
 * NOTE: This information is only used for _display_ purposes: it in
 * no way affects any of the arithmetic of the contract, including
```

```
 * {IERC20-balanceOf} and {IERC20-transfer}.
 */
function decimals() public view virtual override returns (uint8) {
    return 18;
}

/**
 * @dev See {IERC20-totalSupply}.
 */
function totalSupply() public view virtual override returns (uint256) {
    return _totalSupply;
}

/**
 * @dev See {IERC20-balanceOf}.
 */
function balanceOf(address account) public view virtual override returns (uint256) {
    return _balances[account];
}

/**
 * @dev See {IERC20-transfer}.
 *
 * Requirements:
 *
 * - `to` cannot be the zero address.
 * - the caller must have a balance of at least `amount`.
 */
function transfer(address to, uint256 amount) public virtual override returns (bool) {
    address owner = _msgSender();
    _transfer(owner, to, amount);
```

```
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256)
{
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * NOTE: If `amount` is the maximum `uint256`, the allowance is not updated on
     * `transferFrom`. This is semantically equivalent to an infinite approval.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
```

```
 * required by the EIP. See the note at the beginning of {ERC20}.
 *
 * NOTE: Does not update the allowance if the current allowance
 * is the maximum `uint256`.
 *
 * Requirements:
 *
 * - `from` and `to` cannot be the zero address.
 * - `from` must have a balance of at least `amount`.
 * - the caller must have allowance for ``from``'s tokens of at least
 * `amount`.
 */
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
    address spender = _msgSender();
    _spendAllowance(from, spender, amount);
    _transfer(from, to, amount);
    return true;
}

/**
 * @dev Atomically increases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for
 * problems described in {IERC20-approve}.
 *
 * Emits an {Approval} event indicating the updated allowance.
 *
```

```
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, allowance(owner, spender) + addedValue);
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns
(bool) {
        address owner = _msgSender();
        uint256 currentAllowance = allowance(owner, spender);
        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
        unchecked {
            _approve(owner, spender, currentAllowance - subtractedValue);
```

```
    }

    return true;
}

/**
 * @dev Moves `amount` of tokens from `from` to `to`.
 *
 * This internal function is equivalent to {transfer}, and can be used to
 * e.g. implement automatic token fees, slashing mechanisms, etc.
 *
 * Emits a {Transfer} event.
 *
 * Requirements:
 *
 * - `from` cannot be the zero address.
 * - `to` cannot be the zero address.
 * - `from` must have a balance of at least `amount`.
 */
function _transfer(
    address from,
    address to,
    uint256 amount
) internal virtual {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(from, to, amount);

    uint256 fromBalance = _balances[from];
    require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
```

```
    unchecked {
        _balances[from] = fromBalance - amount;
    }
    _balances[to] += amount;

    emit Transfer(from, to, amount);

    _afterTokenTransfer(from, to, amount);
}

/** @dev Creates `amount` tokens and assigns them to `account`, increasing
 * the total supply.
 *
 * Emits a {Transfer} event with `from` set to the zero address.
 *
 * Requirements:
 *
 * - `account` cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

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

    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);

    _afterTokenTransfer(address(0), account, amount);
}
```

```
/**
 * @dev Destroys `amount` tokens from `account`, reducing the
 * total supply.
 *
 * Emits a {Transfer} event with `to` set to the zero address.
 *
 * Requirements:
 *
 * - `account` cannot be the zero address.
 * - `account` must have at least `amount` tokens.
 */
function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

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

    uint256 accountBalance = _balances[account];
    require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
    unchecked {
        _balances[account] = accountBalance - amount;
    }
    _totalSupply -= amount;

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

    _afterTokenTransfer(account, address(0), amount);
}

/**
 * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
 *
```

```
 * This internal function is equivalent to `approve`, and can be used to
 * e.g. set automatic allowances for certain subsystems, etc.
 *
 * Emits an {Approval} event.
 *
 * Requirements:
 *
 * - `owner` cannot be the zero address.
 * - `spender` cannot be the zero address.
 */
function _approve(
    address owner,
    address spender,
    uint256 amount
) internal virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}

/**
 * @dev Updates `owner` s allowance for `spender` based on spent `amount`.
 *
 * Does not update the allowance amount in case of infinite allowance.
 * Revert if not enough allowance is available.
 *
 * Might emit an {Approval} event.
 */
function _spendAllowance(
```

```
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        uint256 currentAllowance = allowance(owner, spender);
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= amount, "ERC20: insufficient allowance");
            unchecked {
                _approve(owner, spender, currentAllowance - amount);
            }
        }
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(
        address from,
        address to,
```

```
        uint256 amount
    ) internal virtual {}

    /**
     * @dev Hook that is called after any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * has been transferred to `to`.
     * - when `from` is zero, `amount` tokens have been minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens have been burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/Counters.sol)

pragma solidity ^0.8.0;

/**
 * @title Counters
```

```
 * @author Matt Condon (@shrugs)

 * @dev Provides counters that can only be incremented, decremented or reset. This can be used e.g. to track the number

 * of elements in a mapping, issuing ERC721 ids, or counting request ids.

 *

 * Include with `using Counters for Counters.Counter;`

 */

library Counters {

    struct Counter {

        // This variable should never be directly accessed by users of the library: interactions must be restricted to

        // the library's function. As of Solidity v0.5.2, this cannot be enforced, though there is a proposal to add

        // this feature: see https://github.com/ethereum/solidity/issues/4637

        uint256 _value; // default: 0

    }


    function current(Counter storage counter) internal view returns (uint256) {

        return counter._value;

    }


    function increment(Counter storage counter) internal {

        unchecked {

            counter._value += 1;

        }

    }


    function decrement(Counter storage counter) internal {

        uint256 value = counter._value;

        require(value > 0, "Counter: decrement overflow");

        unchecked {

            counter._value = value - 1;
```

```
        }
    }

    function reset(Counter storage counter) internal {
        counter._value = 0;
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.3) (utils/cryptography/ECDSA.sol)

pragma solidity ^0.8.0;

import "../Strings.sol";

/**
 * @dev Elliptic Curve Digital Signature Algorithm (ECDSA) operations.
 *
 * These functions can be used to verify that a message was signed by the holder
 * of the private keys of a given address.
 */
library ECDSA {
    enum RecoverError {
        NoError,
        InvalidSignature,
        InvalidSignatureLength,
        InvalidSignatureS,
        InvalidSignatureV
    }

    function _throwError(RecoverError error) private pure {
        if (error == RecoverError.NoError) {
```

```
        return; // no error: do nothing
    } else if (error == RecoverError.InvalidSignature) {
        revert("ECDSA: invalid signature");
    } else if (error == RecoverError.InvalidSignatureLength) {
        revert("ECDSA: invalid signature length");
    } else if (error == RecoverError.InvalidSignatureS) {
        revert("ECDSA: invalid signature 's' value");
    } else if (error == RecoverError.InvalidSignatureV) {
        revert("ECDSA: invalid signature 'v' value");
    }
}

/**
 * @dev Returns the address that signed a hashed message (`hash`) with
 * `signature` or error string. This address can then be used for verification purposes.
 *
 * The `ecrecover` EVM opcode allows for malleable (non-unique) signatures:
 * this function rejects them by requiring the `s` value to be in the lower
 * half order, and the `v` value to be either 27 or 28.
 *
 * IMPORTANT: `hash` _must_ be the result of a hash operation for the
 * verification to be secure: it is possible to craft signatures that
 * recover to arbitrary addresses for non-hashed data. A safe way to ensure
 * this is by receiving a hash of the original message (which may otherwise
 * be too long), and then calling {toEthSignedMessageHash} on it.
 *
 * Documentation for signature generation:
 * - with https://web3js.readthedocs.io/en/v1.3.4/web3-eth-accounts.html#sign[Web3.js]
 * - with https://docs.ethers.io/v5/api/signer/#Signer-signMessage[ethers]
 *
 * _Available since v4.3._
```

```
    */
    function tryRecover(bytes32 hash, bytes memory signature) internal pure returns (address,
RecoverError) {
        if (signature.length == 65) {
            bytes32 r;
            bytes32 s;
            uint8 v;
            // ecrecover takes the signature parameters, and the only way to get them
            // currently is to use assembly.
            /// @solidity memory-safe-assembly
            assembly {
                r := mload(add(signature, 0x20))
                s := mload(add(signature, 0x40))
                v := byte(0, mload(add(signature, 0x60)))
            }
            return tryRecover(hash, v, r, s);
        } else {
            return (address(0), RecoverError.InvalidSignatureLength);
        }
    }

    /**
     * @dev Returns the address that signed a hashed message (`hash`) with
     * `signature`. This address can then be used for verification purposes.
     *
     * The `ecrecover` EVM opcode allows for malleable (non-unique) signatures:
     * this function rejects them by requiring the `s` value to be in the lower
     * half order, and the `v` value to be either 27 or 28.
     *
     * IMPORTANT: `hash` _must_ be the result of a hash operation for the
     * verification to be secure: it is possible to craft signatures that
```

```
     * recover to arbitrary addresses for non-hashed data. A safe way to ensure
     * this is by receiving a hash of the original message (which may otherwise
     * be too long), and then calling {toEthSignedMessageHash} on it.
     */
    function recover(bytes32 hash, bytes memory signature) internal pure returns (address) {
        (address recovered, RecoverError error) = tryRecover(hash, signature);
        _throwError(error);
        return recovered;
    }

    /**
     * @dev Overload of {ECDSA-tryRecover} that receives the `r` and `vs` short-signature fields separately.
     *
     * See https://eips.ethereum.org/EIPS/eip-2098[EIP-2098 short signatures]
     *
     * _Available since v4.3._
     */
    function tryRecover(
        bytes32 hash,
        bytes32 r,
        bytes32 vs
    ) internal pure returns (address, RecoverError) {
        bytes32 s = vs & bytes32(0x7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff);
        uint8 v = uint8((uint256(vs) >> 255) + 27);
        return tryRecover(hash, v, r, s);
    }

    /**
     * @dev Overload of {ECDSA-recover} that receives the `r and `vs` short-signature fields separately.
     *
```

```
     * _Available since v4.2._
     */
    function recover(
        bytes32 hash,
        bytes32 r,
        bytes32 vs
    ) internal pure returns (address) {
        (address recovered, RecoverError error) = tryRecover(hash, r, vs);
        _throwError(error);
        return recovered;
    }

    /**
     * @dev Overload of {ECDSA-tryRecover} that receives the `v`,
     * `r` and `s` signature fields separately.
     *
     * _Available since v4.3._
     */
    function tryRecover(
        bytes32 hash,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) internal pure returns (address, RecoverError) {
        // EIP-2 still allows signature malleability for ecrecover(). Remove this possibility and make the signature
        // unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.io/yellowpaper/paper.pdf), defines
        // the valid range for s in (301): 0 < s < secp256k1n ÷ 2 + 1, and for v in (302): v ∈ {27, 28}. Most
        // signatures from current libraries generate a unique signature with an s-value in the lower half order.
        //
```

```
        // If your library generates malleable signatures, such as s-values in the upper range, calculate a new s-value
        // with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 - s1 and flip v from 27 to 28 or
        // vice versa. If your library also generates signatures with 0/1 for v instead 27/28, add 27 to v to accept
        // these malleable signatures as well.
        if (uint256(s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
            return (address(0), RecoverError.InvalidSignatureS);
        }
        if (v != 27 && v != 28) {
            return (address(0), RecoverError.InvalidSignatureV);
        }

        // If the signature is valid (and not malleable), return the signer address
        address signer = ecrecover(hash, v, r, s);
        if (signer == address(0)) {
            return (address(0), RecoverError.InvalidSignature);
        }

        return (signer, RecoverError.NoError);
    }

    /**
     * @dev Overload of {ECDSA-recover} that receives the `v`,
     * `r` and `s` signature fields separately.
     */
    function recover(
        bytes32 hash,
        uint8 v,
        bytes32 r,
        bytes32 s
```

```
    ) internal pure returns (address) {
        (address recovered, RecoverError error) = tryRecover(hash, v, r, s);
        _throwError(error);
        return recovered;
    }

    /**
     * @dev Returns an Ethereum Signed Message, created from a `hash`. This
     * produces hash corresponding to the one signed with the
     * https://eth.wiki/json-rpc/API#eth_sign[`eth_sign`]
     * JSON-RPC method as part of EIP-191.
     *
     * See {recover}.
     */
    function toEthSignedMessageHash(bytes32 hash) internal pure returns (bytes32) {
        // 32 is the length in bytes of hash,
        // enforced by the type signature above
        return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", hash));
    }

    /**
     * @dev Returns an Ethereum Signed Message, created from `s`. This
     * produces hash corresponding to the one signed with the
     * https://eth.wiki/json-rpc/API#eth_sign[`eth_sign`]
     * JSON-RPC method as part of EIP-191.
     *
     * See {recover}.
     */
    function toEthSignedMessageHash(bytes memory s) internal pure returns (bytes32) {
        return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n",
Strings.toString(s.length), s));
```

```
    }

    /**
     * @dev Returns an Ethereum Signed Typed Data, created from a
     * `domainSeparator` and a `structHash`. This produces hash corresponding
     * to the one signed with the
     * https://eips.ethereum.org/EIPS/eip-712[`eth_signTypedData`]
     * JSON-RPC method as part of EIP-712.
     *
     * See {recover}.
     */
    function toTypedDataHash(bytes32 domainSeparator, bytes32 structHash) internal pure returns (bytes32) {
        return keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/cryptography/draft-EIP712.sol)

pragma solidity ^0.8.0;

import "./ECDSA.sol";

/**
 * @dev https://eips.ethereum.org/EIPS/eip-712[EIP 712] is a standard for hashing and signing of typed structured data.
 *
 * The encoding specified in the EIP is very generic, and such a generic implementation in Solidity is not feasible,
 * thus this contract does not implement the encoding itself. Protocols need to implement the type-specific encoding
 * they need in their contracts using a combination of `abi.encode` and `keccak256`.
```

```
 *
 * This contract implements the EIP 712 domain separator ({_domainSeparatorV4}) that is used as part of the encoding
 * scheme, and the final step of the encoding to obtain the message digest that is then signed via ECDSA
 * ({_hashTypedDataV4}).
 *
 * The implementation of the domain separator was designed to be as efficient as possible while still properly updating
 * the chain id to protect against replay attacks on an eventual fork of the chain.
 *
 * NOTE: This contract implements the version of the encoding known as "v4", as implemented by the JSON RPC method
 * https://docs.metamask.io/guide/signing-data.html[`eth_signTypedDataV4` in MetaMask].
 *
 * _Available since v3.4._
 */
abstract contract EIP712 {
    /* solhint-disable var-name-mixedcase */
    // Cache the domain separator as an immutable value, but also store the chain id that it corresponds to, in order to
    // invalidate the cached domain separator if the chain id changes.
    bytes32 private immutable _CACHED_DOMAIN_SEPARATOR;
    uint256 private immutable _CACHED_CHAIN_ID;
    address private immutable _CACHED_THIS;

    bytes32 private immutable _HASHED_NAME;
    bytes32 private immutable _HASHED_VERSION;
    bytes32 private immutable _TYPE_HASH;

    /* solhint-enable var-name-mixedcase */

    /**
```

```
    * @dev Initializes the domain separator and parameter caches.
    *
    * The meaning of `name` and `version` is specified in
    * https://eips.ethereum.org/EIPS/eip-712#definition-of-domainseparator[EIP 712]:
    *
    * - `name`: the user readable name of the signing domain, i.e. the name of the DApp or the protocol.
    * - `version`: the current major version of the signing domain.
    *
    * NOTE: These parameters cannot be changed except through a xref:learn::upgrading-smart-contracts.adoc[smart
    * contract upgrade].
    */
   constructor(string memory name, string memory version) {
      bytes32 hashedName = keccak256(bytes(name));
      bytes32 hashedVersion = keccak256(bytes(version));
      bytes32 typeHash = keccak256(
         "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
      );
      _HASHED_NAME = hashedName;
      _HASHED_VERSION = hashedVersion;
      _CACHED_CHAIN_ID = block.chainid;
      _CACHED_DOMAIN_SEPARATOR = _buildDomainSeparator(typeHash, hashedName, hashedVersion);
      _CACHED_THIS = address(this);
      _TYPE_HASH = typeHash;
   }

   /**
    * @dev Returns the domain separator for the current chain.
    */
   function _domainSeparatorV4() internal view returns (bytes32) {
```

```
        if (address(this) == _CACHED_THIS && block.chainid == _CACHED_CHAIN_ID) {
            return _CACHED_DOMAIN_SEPARATOR;
        } else {
            return _buildDomainSeparator(_TYPE_HASH, _HASHED_NAME, _HASHED_VERSION);
        }
    }

    function _buildDomainSeparator(
        bytes32 typeHash,
        bytes32 nameHash,
        bytes32 versionHash
    ) private view returns (bytes32) {
        return keccak256(abi.encode(typeHash, nameHash, versionHash, block.chainid, address(this)));
    }

    /**
     * @dev Given an already https://eips.ethereum.org/EIPS/eip-712#definition-of-hashstruct[hashed struct], this
     * function returns the hash of the fully encoded EIP712 message for this domain.
     *
     * This hash can be used together with {ECDSA-recover} to obtain the signer of a message. For example:
     *
     * ```solidity
     * bytes32 digest = _hashTypedDataV4(keccak256(abi.encode(
     *     keccak256("Mail(address to,string contents)"),
     *     mailTo,
     *     keccak256(bytes(mailContents))
     * )));
     * address signer = ECDSA.recover(digest, signature);
     * ```
     */
```

```
    function _hashTypedDataV4(bytes32 structHash) internal view virtual returns (bytes32) {
        return ECDSA.toTypedDataHash(_domainSeparatorV4(), structHash);
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/draft-IERC20Permit.sol)

pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC20 Permit extension allowing approvals to be made via signatures, as defined in
 * https://eips.ethereum.org/EIPS/eip-2612[EIP-2612].
 *
 * Adds the {permit} method, which can be used to change an account's ERC20 allowance (see {IERC20-allowance}) by
 * presenting a message signed by the account. By not relying on {IERC20-approve}, the token holder account doesn't
 * need to send a transaction, and thus is not required to hold Ether at all.
 */
interface IERC20Permit {
    /**
     * @dev Sets `value` as the allowance of `spender` over ``owner``'s tokens,
     * given ``owner``'s signed approval.
     *
     * IMPORTANT: The same issues {IERC20-approve} has related to transaction
     * ordering also apply here.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
```

```
 * - `spender` cannot be the zero address.
 * - `deadline` must be a timestamp in the future.
 * - `v`, `r` and `s` must be a valid `secp256k1` signature from `owner`
 * over the EIP712-formatted function arguments.
 * - the signature must use ``owner``'s current nonce (see {nonces}).
 *
 * For more information on the signature format, see the
 * https://eips.ethereum.org/EIPS/eip-2612#specification[relevant EIP
 * section].
 */
function permit(
    address owner,
    address spender,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external;

/**
 * @dev Returns the current nonce for `owner`. This value must be
 * included whenever a signature is generated for {permit}.
 *
 * Every successful call to {permit} increases ``owner``'s nonce by one. This
 * prevents a signature from being used multiple times.
 */
function nonces(address owner) external view returns (uint256);

/**
```

```
     * @dev Returns the domain separator used in the encoding of the signature for {permit}, as defined by {EIP712}.
     */
    // solhint-disable-next-line func-name-mixedcase
    function DOMAIN_SEPARATOR() external view returns (bytes32);
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (utils/math/SafeCast.sol)

pragma solidity ^0.8.0;

/**
 * @dev Wrappers over Solidity's uintXX/intXX casting operators with added overflow
 * checks.
 *
 * Downcasting from uint256/int256 in Solidity does not revert on overflow. This can
 * easily result in undesired exploitation or bugs, since developers usually
 * assume that overflows raise errors. `SafeCast` restores this intuition by
 * reverting the transaction when such an operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 *
 * Can be combined with {SafeMath} and {SignedSafeMath} to extend it to smaller types, by performing
 * all math on `uint256` and `int256` and then downcasting.
 */
library SafeCast {
    /**
     * @dev Returns the downcasted uint248 from uint256, reverting on
     * overflow (when the input is greater than largest uint248).
     *
```

```
 * Counterpart to Solidity's `uint248` operator.
 *
 * Requirements:
 *
 * - input must fit into 248 bits
 *
 * _Available since v4.7._
 */
function toUint248(uint256 value) internal pure returns (uint248) {
    require(value <= type(uint248).max, "SafeCast: value doesn't fit in 248 bits");
    return uint248(value);
}

/**
 * @dev Returns the downcasted uint240 from uint256, reverting on
 * overflow (when the input is greater than largest uint240).
 *
 * Counterpart to Solidity's `uint240` operator.
 *
 * Requirements:
 *
 * - input must fit into 240 bits
 *
 * _Available since v4.7._
 */
function toUint240(uint256 value) internal pure returns (uint240) {
    require(value <= type(uint240).max, "SafeCast: value doesn't fit in 240 bits");
    return uint240(value);
}

/**
```

```
 * @dev Returns the downcasted uint232 from uint256, reverting on
 * overflow (when the input is greater than largest uint232).
 *
 * Counterpart to Solidity's `uint232` operator.
 *
 * Requirements:
 *
 * - input must fit into 232 bits
 *
 * _Available since v4.7._
 */
function toUint232(uint256 value) internal pure returns (uint232) {
    require(value <= type(uint232).max, "SafeCast: value doesn't fit in 232 bits");
    return uint232(value);
}

/**
 * @dev Returns the downcasted uint224 from uint256, reverting on
 * overflow (when the input is greater than largest uint224).
 *
 * Counterpart to Solidity's `uint224` operator.
 *
 * Requirements:
 *
 * - input must fit into 224 bits
 *
 * _Available since v4.2._
 */
function toUint224(uint256 value) internal pure returns (uint224) {
    require(value <= type(uint224).max, "SafeCast: value doesn't fit in 224 bits");
    return uint224(value);
```

```
}

/**
 * @dev Returns the downcasted uint216 from uint256, reverting on
 * overflow (when the input is greater than largest uint216).
 *
 * Counterpart to Solidity's `uint216` operator.
 *
 * Requirements:
 *
 * - input must fit into 216 bits
 *
 * _Available since v4.7._
 */
function toUint216(uint256 value) internal pure returns (uint216) {
    require(value <= type(uint216).max, "SafeCast: value doesn't fit in 216 bits");
    return uint216(value);
}

/**
 * @dev Returns the downcasted uint208 from uint256, reverting on
 * overflow (when the input is greater than largest uint208).
 *
 * Counterpart to Solidity's `uint208` operator.
 *
 * Requirements:
 *
 * - input must fit into 208 bits
 *
 * _Available since v4.7._
 */
```

```
function toUint208(uint256 value) internal pure returns (uint208) {
    require(value <= type(uint208).max, "SafeCast: value doesn't fit in 208 bits");
    return uint208(value);
}

/**
 * @dev Returns the downcasted uint200 from uint256, reverting on
 * overflow (when the input is greater than largest uint200).
 *
 * Counterpart to Solidity's `uint200` operator.
 *
 * Requirements:
 *
 * - input must fit into 200 bits
 *
 * _Available since v4.7._
 */
function toUint200(uint256 value) internal pure returns (uint200) {
    require(value <= type(uint200).max, "SafeCast: value doesn't fit in 200 bits");
    return uint200(value);
}

/**
 * @dev Returns the downcasted uint192 from uint256, reverting on
 * overflow (when the input is greater than largest uint192).
 *
 * Counterpart to Solidity's `uint192` operator.
 *
 * Requirements:
 *
 * - input must fit into 192 bits
```

```
 *
 * _Available since v4.7._
 */
function toUint192(uint256 value) internal pure returns (uint192) {
    require(value <= type(uint192).max, "SafeCast: value doesn't fit in 192 bits");
    return uint192(value);
}

/**
 * @dev Returns the downcasted uint184 from uint256, reverting on
 * overflow (when the input is greater than largest uint184).
 *
 * Counterpart to Solidity's `uint184` operator.
 *
 * Requirements:
 *
 * - input must fit into 184 bits
 *
 * _Available since v4.7._
 */
function toUint184(uint256 value) internal pure returns (uint184) {
    require(value <= type(uint184).max, "SafeCast: value doesn't fit in 184 bits");
    return uint184(value);
}

/**
 * @dev Returns the downcasted uint176 from uint256, reverting on
 * overflow (when the input is greater than largest uint176).
 *
 * Counterpart to Solidity's `uint176` operator.
 *
```

```
 * Requirements:
 *
 * - input must fit into 176 bits
 *
 * _Available since v4.7._
 */
function toUint176(uint256 value) internal pure returns (uint176) {
    require(value <= type(uint176).max, "SafeCast: value doesn't fit in 176 bits");
    return uint176(value);
}

/**
 * @dev Returns the downcasted uint168 from uint256, reverting on
 * overflow (when the input is greater than largest uint168).
 *
 * Counterpart to Solidity's `uint168` operator.
 *
 * Requirements:
 *
 * - input must fit into 168 bits
 *
 * _Available since v4.7._
 */
function toUint168(uint256 value) internal pure returns (uint168) {
    require(value <= type(uint168).max, "SafeCast: value doesn't fit in 168 bits");
    return uint168(value);
}

/**
 * @dev Returns the downcasted uint160 from uint256, reverting on
 * overflow (when the input is greater than largest uint160).
```

```
 *
 * Counterpart to Solidity's `uint160` operator.
 *
 * Requirements:
 *
 * - input must fit into 160 bits
 *
 * _Available since v4.7._
 */
function toUint160(uint256 value) internal pure returns (uint160) {
    require(value <= type(uint160).max, "SafeCast: value doesn't fit in 160 bits");
    return uint160(value);
}

/**
 * @dev Returns the downcasted uint152 from uint256, reverting on
 * overflow (when the input is greater than largest uint152).
 *
 * Counterpart to Solidity's `uint152` operator.
 *
 * Requirements:
 *
 * - input must fit into 152 bits
 *
 * _Available since v4.7._
 */
function toUint152(uint256 value) internal pure returns (uint152) {
    require(value <= type(uint152).max, "SafeCast: value doesn't fit in 152 bits");
    return uint152(value);
}
```

```
/**
 * @dev Returns the downcasted uint144 from uint256, reverting on
 * overflow (when the input is greater than largest uint144).
 *
 * Counterpart to Solidity's `uint144` operator.
 *
 * Requirements:
 *
 * - input must fit into 144 bits
 *
 * _Available since v4.7._
 */
function toUint144(uint256 value) internal pure returns (uint144) {
    require(value <= type(uint144).max, "SafeCast: value doesn't fit in 144 bits");
    return uint144(value);
}

/**
 * @dev Returns the downcasted uint136 from uint256, reverting on
 * overflow (when the input is greater than largest uint136).
 *
 * Counterpart to Solidity's `uint136` operator.
 *
 * Requirements:
 *
 * - input must fit into 136 bits
 *
 * _Available since v4.7._
 */
function toUint136(uint256 value) internal pure returns (uint136) {
    require(value <= type(uint136).max, "SafeCast: value doesn't fit in 136 bits");
```

```
    return uint136(value);
}

/**
 * @dev Returns the downcasted uint128 from uint256, reverting on
 * overflow (when the input is greater than largest uint128).
 *
 * Counterpart to Solidity's `uint128` operator.
 *
 * Requirements:
 *
 * - input must fit into 128 bits
 *
 * _Available since v2.5._
 */
function toUint128(uint256 value) internal pure returns (uint128) {
    require(value <= type(uint128).max, "SafeCast: value doesn't fit in 128 bits");
    return uint128(value);
}

/**
 * @dev Returns the downcasted uint120 from uint256, reverting on
 * overflow (when the input is greater than largest uint120).
 *
 * Counterpart to Solidity's `uint120` operator.
 *
 * Requirements:
 *
 * - input must fit into 120 bits
 *
 * _Available since v4.7._
```

```
 */
function toUint120(uint256 value) internal pure returns (uint120) {
    require(value <= type(uint120).max, "SafeCast: value doesn't fit in 120 bits");
    return uint120(value);
}

/**
 * @dev Returns the downcasted uint112 from uint256, reverting on
 * overflow (when the input is greater than largest uint112).
 *
 * Counterpart to Solidity's `uint112` operator.
 *
 * Requirements:
 *
 * - input must fit into 112 bits
 *
 * _Available since v4.7._
 */
function toUint112(uint256 value) internal pure returns (uint112) {
    require(value <= type(uint112).max, "SafeCast: value doesn't fit in 112 bits");
    return uint112(value);
}

/**
 * @dev Returns the downcasted uint104 from uint256, reverting on
 * overflow (when the input is greater than largest uint104).
 *
 * Counterpart to Solidity's `uint104` operator.
 *
 * Requirements:
 *
```

```
 * - input must fit into 104 bits
 *
 * _Available since v4.7._
 */
function toUint104(uint256 value) internal pure returns (uint104) {
    require(value <= type(uint104).max, "SafeCast: value doesn't fit in 104 bits");
    return uint104(value);
}

/**
 * @dev Returns the downcasted uint96 from uint256, reverting on
 * overflow (when the input is greater than largest uint96).
 *
 * Counterpart to Solidity's `uint96` operator.
 *
 * Requirements:
 *
 * - input must fit into 96 bits
 *
 * _Available since v4.2._
 */
function toUint96(uint256 value) internal pure returns (uint96) {
    require(value <= type(uint96).max, "SafeCast: value doesn't fit in 96 bits");
    return uint96(value);
}

/**
 * @dev Returns the downcasted uint88 from uint256, reverting on
 * overflow (when the input is greater than largest uint88).
 *
 * Counterpart to Solidity's `uint88` operator.
```

```
 *
 * Requirements:
 *
 * - input must fit into 88 bits
 *
 * _Available since v4.7._
 */
function toUint88(uint256 value) internal pure returns (uint88) {
    require(value <= type(uint88).max, "SafeCast: value doesn't fit in 88 bits");
    return uint88(value);
}

/**
 * @dev Returns the downcasted uint80 from uint256, reverting on
 * overflow (when the input is greater than largest uint80).
 *
 * Counterpart to Solidity's `uint80` operator.
 *
 * Requirements:
 *
 * - input must fit into 80 bits
 *
 * _Available since v4.7._
 */
function toUint80(uint256 value) internal pure returns (uint80) {
    require(value <= type(uint80).max, "SafeCast: value doesn't fit in 80 bits");
    return uint80(value);
}

/**
 * @dev Returns the downcasted uint72 from uint256, reverting on
```

```
 * overflow (when the input is greater than largest uint72).
 *
 * Counterpart to Solidity's `uint72` operator.
 *
 * Requirements:
 *
 * - input must fit into 72 bits
 *
 * _Available since v4.7._
 */
function toUint72(uint256 value) internal pure returns (uint72) {
    require(value <= type(uint72).max, "SafeCast: value doesn't fit in 72 bits");
    return uint72(value);
}

/**
 * @dev Returns the downcasted uint64 from uint256, reverting on
 * overflow (when the input is greater than largest uint64).
 *
 * Counterpart to Solidity's `uint64` operator.
 *
 * Requirements:
 *
 * - input must fit into 64 bits
 *
 * _Available since v2.5._
 */
function toUint64(uint256 value) internal pure returns (uint64) {
    require(value <= type(uint64).max, "SafeCast: value doesn't fit in 64 bits");
    return uint64(value);
}
```

```
/**
 * @dev Returns the downcasted uint56 from uint256, reverting on
 * overflow (when the input is greater than largest uint56).
 *
 * Counterpart to Solidity's `uint56` operator.
 *
 * Requirements:
 *
 * - input must fit into 56 bits
 *
 * _Available since v4.7._
 */
function toUint56(uint256 value) internal pure returns (uint56) {
    require(value <= type(uint56).max, "SafeCast: value doesn't fit in 56 bits");
    return uint56(value);
}

/**
 * @dev Returns the downcasted uint48 from uint256, reverting on
 * overflow (when the input is greater than largest uint48).
 *
 * Counterpart to Solidity's `uint48` operator.
 *
 * Requirements:
 *
 * - input must fit into 48 bits
 *
 * _Available since v4.7._
 */
function toUint48(uint256 value) internal pure returns (uint48) {
```

```
    require(value <= type(uint48).max, "SafeCast: value doesn't fit in 48 bits");
    return uint48(value);
}

/**
 * @dev Returns the downcasted uint40 from uint256, reverting on
 * overflow (when the input is greater than largest uint40).
 *
 * Counterpart to Solidity's `uint40` operator.
 *
 * Requirements:
 *
 * - input must fit into 40 bits
 *
 * _Available since v4.7._
 */
function toUint40(uint256 value) internal pure returns (uint40) {
    require(value <= type(uint40).max, "SafeCast: value doesn't fit in 40 bits");
    return uint40(value);
}

/**
 * @dev Returns the downcasted uint32 from uint256, reverting on
 * overflow (when the input is greater than largest uint32).
 *
 * Counterpart to Solidity's `uint32` operator.
 *
 * Requirements:
 *
 * - input must fit into 32 bits
 *
```

```
 * _Available since v2.5._
 */
function toUint32(uint256 value) internal pure returns (uint32) {
    require(value <= type(uint32).max, "SafeCast: value doesn't fit in 32 bits");
    return uint32(value);
}

/**
 * @dev Returns the downcasted uint24 from uint256, reverting on
 * overflow (when the input is greater than largest uint24).
 *
 * Counterpart to Solidity's `uint24` operator.
 *
 * Requirements:
 *
 * - input must fit into 24 bits
 *
 * _Available since v4.7._
 */
function toUint24(uint256 value) internal pure returns (uint24) {
    require(value <= type(uint24).max, "SafeCast: value doesn't fit in 24 bits");
    return uint24(value);
}

/**
 * @dev Returns the downcasted uint16 from uint256, reverting on
 * overflow (when the input is greater than largest uint16).
 *
 * Counterpart to Solidity's `uint16` operator.
 *
 * Requirements:
```

```
 *
 * - input must fit into 16 bits
 *
 * _Available since v2.5._
 */
function toUint16(uint256 value) internal pure returns (uint16) {
    require(value <= type(uint16).max, "SafeCast: value doesn't fit in 16 bits");
    return uint16(value);
}

/**
 * @dev Returns the downcasted uint8 from uint256, reverting on
 * overflow (when the input is greater than largest uint8).
 *
 * Counterpart to Solidity's `uint8` operator.
 *
 * Requirements:
 *
 * - input must fit into 8 bits
 *
 * _Available since v2.5._
 */
function toUint8(uint256 value) internal pure returns (uint8) {
    require(value <= type(uint8).max, "SafeCast: value doesn't fit in 8 bits");
    return uint8(value);
}

/**
 * @dev Converts a signed int256 into an unsigned uint256.
 *
 * Requirements:
```

```
     *
     * - input must be greater than or equal to 0.
     *
     * _Available since v3.0._
     */
    function toUint256(int256 value) internal pure returns (uint256) {
        require(value >= 0, "SafeCast: value must be positive");
        return uint256(value);
    }

    /**
     * @dev Returns the downcasted int248 from int256, reverting on
     * overflow (when the input is less than smallest int248 or
     * greater than largest int248).
     *
     * Counterpart to Solidity's `int248` operator.
     *
     * Requirements:
     *
     * - input must fit into 248 bits
     *
     * _Available since v4.7._
     */
    function toInt248(int256 value) internal pure returns (int248) {
        require(value >= type(int248).min && value <= type(int248).max, "SafeCast: value doesn't fit in
248 bits");
        return int248(value);
    }

    /**
     * @dev Returns the downcasted int240 from int256, reverting on
```

```
     * overflow (when the input is less than smallest int240 or
     * greater than largest int240).
     *
     * Counterpart to Solidity's `int240` operator.
     *
     * Requirements:
     *
     * - input must fit into 240 bits
     *
     * _Available since v4.7._
     */
    function toInt240(int256 value) internal pure returns (int240) {
        require(value >= type(int240).min && value <= type(int240).max, "SafeCast: value doesn't fit in 240 bits");
        return int240(value);
    }

    /**
     * @dev Returns the downcasted int232 from int256, reverting on
     * overflow (when the input is less than smallest int232 or
     * greater than largest int232).
     *
     * Counterpart to Solidity's `int232` operator.
     *
     * Requirements:
     *
     * - input must fit into 232 bits
     *
     * _Available since v4.7._
     */
    function toInt232(int256 value) internal pure returns (int232) {
```

```
        require(value >= type(int232).min && value <= type(int232).max, "SafeCast: value doesn't fit in 232 bits");
        return int232(value);
    }

    /**
     * @dev Returns the downcasted int224 from int256, reverting on
     * overflow (when the input is less than smallest int224 or
     * greater than largest int224).
     *
     * Counterpart to Solidity's `int224` operator.
     *
     * Requirements:
     *
     * - input must fit into 224 bits
     *
     * _Available since v4.7._
     */
    function toInt224(int256 value) internal pure returns (int224) {
        require(value >= type(int224).min && value <= type(int224).max, "SafeCast: value doesn't fit in 224 bits");
        return int224(value);
    }

    /**
     * @dev Returns the downcasted int216 from int256, reverting on
     * overflow (when the input is less than smallest int216 or
     * greater than largest int216).
     *
     * Counterpart to Solidity's `int216` operator.
     *
     * Requirements:
```

```
     *
     * - input must fit into 216 bits
     *
     * _Available since v4.7._
     */
    function toInt216(int256 value) internal pure returns (int216) {
        require(value >= type(int216).min && value <= type(int216).max, "SafeCast: value doesn't fit in 216 bits");
        return int216(value);
    }

    /**
     * @dev Returns the downcasted int208 from int256, reverting on
     * overflow (when the input is less than smallest int208 or
     * greater than largest int208).
     *
     * Counterpart to Solidity's `int208` operator.
     *
     * Requirements:
     *
     * - input must fit into 208 bits
     *
     * _Available since v4.7._
     */
    function toInt208(int256 value) internal pure returns (int208) {
        require(value >= type(int208).min && value <= type(int208).max, "SafeCast: value doesn't fit in 208 bits");
        return int208(value);
    }

    /**
     * @dev Returns the downcasted int200 from int256, reverting on
```

```
     * overflow (when the input is less than smallest int200 or
     * greater than largest int200).
     *
     * Counterpart to Solidity's `int200` operator.
     *
     * Requirements:
     *
     * - input must fit into 200 bits
     *
     * _Available since v4.7._
     */
    function toInt200(int256 value) internal pure returns (int200) {
        require(value >= type(int200).min && value <= type(int200).max, "SafeCast: value doesn't fit in 200 bits");
        return int200(value);
    }

    /**
     * @dev Returns the downcasted int192 from int256, reverting on
     * overflow (when the input is less than smallest int192 or
     * greater than largest int192).
     *
     * Counterpart to Solidity's `int192` operator.
     *
     * Requirements:
     *
     * - input must fit into 192 bits
     *
     * _Available since v4.7._
     */
    function toInt192(int256 value) internal pure returns (int192) {
```

```
        require(value >= type(int192).min && value <= type(int192).max, "SafeCast: value doesn't fit in 192 bits");
        return int192(value);
    }

    /**
     * @dev Returns the downcasted int184 from int256, reverting on
     * overflow (when the input is less than smallest int184 or
     * greater than largest int184).
     *
     * Counterpart to Solidity's `int184` operator.
     *
     * Requirements:
     *
     * - input must fit into 184 bits
     *
     * _Available since v4.7._
     */
    function toInt184(int256 value) internal pure returns (int184) {
        require(value >= type(int184).min && value <= type(int184).max, "SafeCast: value doesn't fit in 184 bits");
        return int184(value);
    }

    /**
     * @dev Returns the downcasted int176 from int256, reverting on
     * overflow (when the input is less than smallest int176 or
     * greater than largest int176).
     *
     * Counterpart to Solidity's `int176` operator.
     *
     * Requirements:
```

```
     *
     * - input must fit into 176 bits
     *
     * _Available since v4.7._
     */
    function toInt176(int256 value) internal pure returns (int176) {
        require(value >= type(int176).min && value <= type(int176).max, "SafeCast: value doesn't fit in 176 bits");
        return int176(value);
    }

    /**
     * @dev Returns the downcasted int168 from int256, reverting on
     * overflow (when the input is less than smallest int168 or
     * greater than largest int168).
     *
     * Counterpart to Solidity's `int168` operator.
     *
     * Requirements:
     *
     * - input must fit into 168 bits
     *
     * _Available since v4.7._
     */
    function toInt168(int256 value) internal pure returns (int168) {
        require(value >= type(int168).min && value <= type(int168).max, "SafeCast: value doesn't fit in 168 bits");
        return int168(value);
    }

    /**
     * @dev Returns the downcasted int160 from int256, reverting on
```

```
     * overflow (when the input is less than smallest int160 or
     * greater than largest int160).
     *
     * Counterpart to Solidity's `int160` operator.
     *
     * Requirements:
     *
     * - input must fit into 160 bits
     *
     * _Available since v4.7._
     */
    function toInt160(int256 value) internal pure returns (int160) {
        require(value >= type(int160).min && value <= type(int160).max, "SafeCast: value doesn't fit in 160 bits");
        return int160(value);
    }

    /**
     * @dev Returns the downcasted int152 from int256, reverting on
     * overflow (when the input is less than smallest int152 or
     * greater than largest int152).
     *
     * Counterpart to Solidity's `int152` operator.
     *
     * Requirements:
     *
     * - input must fit into 152 bits
     *
     * _Available since v4.7._
     */
    function toInt152(int256 value) internal pure returns (int152) {
```

```
        require(value >= type(int152).min && value <= type(int152).max, "SafeCast: value doesn't fit in 152 bits");
        return int152(value);
    }

    /**
     * @dev Returns the downcasted int144 from int256, reverting on
     * overflow (when the input is less than smallest int144 or
     * greater than largest int144).
     *
     * Counterpart to Solidity's `int144` operator.
     *
     * Requirements:
     *
     * - input must fit into 144 bits
     *
     * _Available since v4.7._
     */
    function toInt144(int256 value) internal pure returns (int144) {
        require(value >= type(int144).min && value <= type(int144).max, "SafeCast: value doesn't fit in 144 bits");
        return int144(value);
    }

    /**
     * @dev Returns the downcasted int136 from int256, reverting on
     * overflow (when the input is less than smallest int136 or
     * greater than largest int136).
     *
     * Counterpart to Solidity's `int136` operator.
     *
     * Requirements:
```

```
     *
     * - input must fit into 136 bits
     *
     * _Available since v4.7._
     */
    function toInt136(int256 value) internal pure returns (int136) {
        require(value >= type(int136).min && value <= type(int136).max, "SafeCast: value doesn't fit in 136 bits");
        return int136(value);
    }

    /**
     * @dev Returns the downcasted int128 from int256, reverting on
     * overflow (when the input is less than smallest int128 or
     * greater than largest int128).
     *
     * Counterpart to Solidity's `int128` operator.
     *
     * Requirements:
     *
     * - input must fit into 128 bits
     *
     * _Available since v3.1._
     */
    function toInt128(int256 value) internal pure returns (int128) {
        require(value >= type(int128).min && value <= type(int128).max, "SafeCast: value doesn't fit in 128 bits");
        return int128(value);
    }

    /**
     * @dev Returns the downcasted int120 from int256, reverting on
```

```
     * overflow (when the input is less than smallest int120 or
     * greater than largest int120).
     *
     * Counterpart to Solidity's `int120` operator.
     *
     * Requirements:
     *
     * - input must fit into 120 bits
     *
     * _Available since v4.7._
     */
    function toInt120(int256 value) internal pure returns (int120) {
        require(value >= type(int120).min && value <= type(int120).max, "SafeCast: value doesn't fit in 120 bits");
        return int120(value);
    }

    /**
     * @dev Returns the downcasted int112 from int256, reverting on
     * overflow (when the input is less than smallest int112 or
     * greater than largest int112).
     *
     * Counterpart to Solidity's `int112` operator.
     *
     * Requirements:
     *
     * - input must fit into 112 bits
     *
     * _Available since v4.7._
     */
    function toInt112(int256 value) internal pure returns (int112) {
```

```
        require(value >= type(int112).min && value <= type(int112).max, "SafeCast: value doesn't fit in 112 bits");

        return int112(value);
    }

    /**
     * @dev Returns the downcasted int104 from int256, reverting on
     * overflow (when the input is less than smallest int104 or
     * greater than largest int104).
     *
     * Counterpart to Solidity's `int104` operator.
     *
     * Requirements:
     *
     * - input must fit into 104 bits
     *
     * _Available since v4.7._
     */
    function toInt104(int256 value) internal pure returns (int104) {
        require(value >= type(int104).min && value <= type(int104).max, "SafeCast: value doesn't fit in 104 bits");

        return int104(value);
    }

    /**
     * @dev Returns the downcasted int96 from int256, reverting on
     * overflow (when the input is less than smallest int96 or
     * greater than largest int96).
     *
     * Counterpart to Solidity's `int96` operator.
     *
     * Requirements:
```

```
     *
     * - input must fit into 96 bits
     *
     * _Available since v4.7._
     */
    function toInt96(int256 value) internal pure returns (int96) {
        require(value >= type(int96).min && value <= type(int96).max, "SafeCast: value doesn't fit in 96 bits");
        return int96(value);
    }

    /**
     * @dev Returns the downcasted int88 from int256, reverting on
     * overflow (when the input is less than smallest int88 or
     * greater than largest int88).
     *
     * Counterpart to Solidity's `int88` operator.
     *
     * Requirements:
     *
     * - input must fit into 88 bits
     *
     * _Available since v4.7._
     */
    function toInt88(int256 value) internal pure returns (int88) {
        require(value >= type(int88).min && value <= type(int88).max, "SafeCast: value doesn't fit in 88 bits");
        return int88(value);
    }

    /**
     * @dev Returns the downcasted int80 from int256, reverting on
```

```
     * overflow (when the input is less than smallest int80 or
     * greater than largest int80).
     *
     * Counterpart to Solidity's `int80` operator.
     *
     * Requirements:
     *
     * - input must fit into 80 bits
     *
     * _Available since v4.7._
     */
    function toInt80(int256 value) internal pure returns (int80) {
        require(value >= type(int80).min && value <= type(int80).max, "SafeCast: value doesn't fit in 80 bits");
        return int80(value);
    }

    /**
     * @dev Returns the downcasted int72 from int256, reverting on
     * overflow (when the input is less than smallest int72 or
     * greater than largest int72).
     *
     * Counterpart to Solidity's `int72` operator.
     *
     * Requirements:
     *
     * - input must fit into 72 bits
     *
     * _Available since v4.7._
     */
    function toInt72(int256 value) internal pure returns (int72) {
```

```
        require(value >= type(int72).min && value <= type(int72).max, "SafeCast: value doesn't fit in 72 bits");
        return int72(value);
    }

    /**
     * @dev Returns the downcasted int64 from int256, reverting on
     * overflow (when the input is less than smallest int64 or
     * greater than largest int64).
     *
     * Counterpart to Solidity's `int64` operator.
     *
     * Requirements:
     *
     * - input must fit into 64 bits
     *
     * _Available since v3.1._
     */
    function toInt64(int256 value) internal pure returns (int64) {
        require(value >= type(int64).min && value <= type(int64).max, "SafeCast: value doesn't fit in 64 bits");
        return int64(value);
    }

    /**
     * @dev Returns the downcasted int56 from int256, reverting on
     * overflow (when the input is less than smallest int56 or
     * greater than largest int56).
     *
     * Counterpart to Solidity's `int56` operator.
     *
     * Requirements:
```

```
     *
     * - input must fit into 56 bits
     *
     * _Available since v4.7._
     */
    function toInt56(int256 value) internal pure returns (int56) {
        require(value >= type(int56).min && value <= type(int56).max, "SafeCast: value doesn't fit in 56 bits");
        return int56(value);
    }

    /**
     * @dev Returns the downcasted int48 from int256, reverting on
     * overflow (when the input is less than smallest int48 or
     * greater than largest int48).
     *
     * Counterpart to Solidity's `int48` operator.
     *
     * Requirements:
     *
     * - input must fit into 48 bits
     *
     * _Available since v4.7._
     */
    function toInt48(int256 value) internal pure returns (int48) {
        require(value >= type(int48).min && value <= type(int48).max, "SafeCast: value doesn't fit in 48 bits");
        return int48(value);
    }

    /**
     * @dev Returns the downcasted int40 from int256, reverting on
```

```
     * overflow (when the input is less than smallest int40 or
     * greater than largest int40).
     *
     * Counterpart to Solidity's `int40` operator.
     *
     * Requirements:
     *
     * - input must fit into 40 bits
     *
     * _Available since v4.7._
     */
    function toInt40(int256 value) internal pure returns (int40) {
        require(value >= type(int40).min && value <= type(int40).max, "SafeCast: value doesn't fit in 40 bits");
        return int40(value);
    }

    /**
     * @dev Returns the downcasted int32 from int256, reverting on
     * overflow (when the input is less than smallest int32 or
     * greater than largest int32).
     *
     * Counterpart to Solidity's `int32` operator.
     *
     * Requirements:
     *
     * - input must fit into 32 bits
     *
     * _Available since v3.1._
     */
    function toInt32(int256 value) internal pure returns (int32) {
```

```
        require(value >= type(int32).min && value <= type(int32).max, "SafeCast: value doesn't fit in 32 bits");
        return int32(value);
    }

    /**
     * @dev Returns the downcasted int24 from int256, reverting on
     * overflow (when the input is less than smallest int24 or
     * greater than largest int24).
     *
     * Counterpart to Solidity's `int24` operator.
     *
     * Requirements:
     *
     * - input must fit into 24 bits
     *
     * _Available since v4.7._
     */
    function toInt24(int256 value) internal pure returns (int24) {
        require(value >= type(int24).min && value <= type(int24).max, "SafeCast: value doesn't fit in 24 bits");
        return int24(value);
    }

    /**
     * @dev Returns the downcasted int16 from int256, reverting on
     * overflow (when the input is less than smallest int16 or
     * greater than largest int16).
     *
     * Counterpart to Solidity's `int16` operator.
     *
     * Requirements:
```

```
     *
     * - input must fit into 16 bits
     *
     * _Available since v3.1._
     */
    function toInt16(int256 value) internal pure returns (int16) {
        require(value >= type(int16).min && value <= type(int16).max, "SafeCast: value doesn't fit in 16 bits");
        return int16(value);
    }

    /**
     * @dev Returns the downcasted int8 from int256, reverting on
     * overflow (when the input is less than smallest int8 or
     * greater than largest int8).
     *
     * Counterpart to Solidity's `int8` operator.
     *
     * Requirements:
     *
     * - input must fit into 8 bits
     *
     * _Available since v3.1._
     */
    function toInt8(int256 value) internal pure returns (int8) {
        require(value >= type(int8).min && value <= type(int8).max, "SafeCast: value doesn't fit in 8 bits");
        return int8(value);
    }

    /**
     * @dev Converts an unsigned uint256 into a signed int256.
```

```
     *
     * Requirements:
     *
     * - input must be less than or equal to maxInt256.
     *
     * _Available since v3.0._
     */
    function toInt256(uint256 value) internal pure returns (int256) {
        // Note: Unsafe cast below is okay because `type(int256).max` is guaranteed to be positive
        require(value <= uint256(type(int256).max), "SafeCast: value doesn't fit in an int256");
        return int256(value);
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.5.0) (governance/utils/IVotes.sol)
pragma solidity ^0.8.0;

/**
 * @dev Common interface for {ERC20Votes}, {ERC721Votes}, and other {Votes}-enabled contracts.
 *
 * _Available since v4.5._
 */
interface IVotes {
    /**
     * @dev Emitted when an account changes their delegate.
     */
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address
indexed toDelegate);

    /**
     * @dev Emitted when a token transfer or delegate change results in changes to a delegate's
number of votes.
```

```
    */
    event DelegateVotesChanged(address indexed delegate, uint256 previousBalance, uint256 newBalance);

    /**
     * @dev Returns the current amount of votes that `account` has.
     */
    function getVotes(address account) external view returns (uint256);

    /**
     * @dev Returns the amount of votes that `account` had at the end of a past block (`blockNumber`).
     */
    function getPastVotes(address account, uint256 blockNumber) external view returns (uint256);

    /**
     * @dev Returns the total supply of votes available at the end of a past block (`blockNumber`).
     *
     * NOTE: This value is the sum of all available votes, which is not necessarily the sum of all delegated votes.
     * Votes that have not been delegated are still part of total supply, even though they would not participate in a
     * vote.
     */
    function getPastTotalSupply(uint256 blockNumber) external view returns (uint256);

    /**
     * @dev Returns the delegate that `account` has chosen.
     */
    function delegates(address account) external view returns (address);

    /**
```

```
     * @dev Delegates votes from the sender to `delegatee`.
     */
    function delegate(address delegatee) external;

    /**
     * @dev Delegates votes from signer to `delegatee`.
     */
    function delegateBySig(
        address delegatee,
        uint256 nonce,
        uint256 expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (utils/math/Math.sol)

pragma solidity ^0.8.0;

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
    enum Rounding {
        Down, // Toward negative infinity
        Up, // Toward infinity
        Zero // Toward zero
    }
```

```
/**
 * @dev Returns the largest of two numbers.
 */
function max(uint256 a, uint256 b) internal pure returns (uint256) {
    return a >= b ? a : b;
}

/**
 * @dev Returns the smallest of two numbers.
 */
function min(uint256 a, uint256 b) internal pure returns (uint256) {
    return a < b ? a : b;
}

/**
 * @dev Returns the average of two numbers. The result is rounded towards
 * zero.
 */
function average(uint256 a, uint256 b) internal pure returns (uint256) {
    // (a + b) / 2 can overflow.
    return (a & b) + (a ^ b) / 2;
}

/**
 * @dev Returns the ceiling of the division of two numbers.
 *
 * This differs from standard division with `/` in that it rounds up instead
 * of rounding down.
 */
function ceilDiv(uint256 a, uint256 b) internal pure returns (uint256) {
    // (a + b - 1) / b can overflow on addition, so we distribute.
```

```
        return a == 0 ? 0 : (a - 1) / b + 1;
    }

    /**
     * @notice Calculates floor(x * y / denominator) with full precision. Throws if result overflows a uint256 or denominator == 0
     * @dev Original credit to Remco Bloemen under MIT license (https://xn--2-umb.com/21/muldiv)
     * with further edits by Uniswap Labs also under MIT license.
     */
    function mulDiv(
        uint256 x,
        uint256 y,
        uint256 denominator
    ) internal pure returns (uint256 result) {
        unchecked {
            // 512-bit multiply [prod1 prod0] = x * y. Compute the product mod 2^256 and mod 2^256 - 1, then use
            // use the Chinese Remainder Theorem to reconstruct the 512 bit result. The result is stored in two 256
            // variables such that product = prod1 * 2^256 + prod0.
            uint256 prod0; // Least significant 256 bits of the product
            uint256 prod1; // Most significant 256 bits of the product
            assembly {
                let mm := mulmod(x, y, not(0))
                prod0 := mul(x, y)
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

            // Handle non-overflow cases, 256 by 256 division.
            if (prod1 == 0) {
                return prod0 / denominator;
            }
```

```
        // Make sure the result is less than 2^256. Also prevents denominator == 0.
        require(denominator > prod1);

        ///////////////////////////////////////////////
        // 512 by 256 division.
        ///////////////////////////////////////////////

        // Make division exact by subtracting the remainder from [prod1 prod0].
        uint256 remainder;
        assembly {
            // Compute remainder using mulmod.
            remainder := mulmod(x, y, denominator)

            // Subtract 256 bit number from 512 bit number.
            prod1 := sub(prod1, gt(remainder, prod0))
            prod0 := sub(prod0, remainder)
        }

        // Factor powers of two out of denominator and compute largest power of two divisor of
denominator. Always >= 1.
        // See https://cs.stackexchange.com/q/138556/92363.

        // Does not overflow because the denominator cannot be zero at this stage in the function.
        uint256 twos = denominator & (~denominator + 1);
        assembly {
            // Divide denominator by twos.
            denominator := div(denominator, twos)

            // Divide [prod1 prod0] by twos.
            prod0 := div(prod0, twos)
```

```
            // Flip twos such that it is 2^256 / twos. If twos is zero, then it becomes one.
            twos := add(div(sub(0, twos), twos), 1)
        }

        // Shift in bits from prod1 into prod0.
        prod0 |= prod1 * twos;

        // Invert denominator mod 2^256. Now that denominator is an odd number, it has an inverse modulo 2^256 such
        // that denominator * inv = 1 mod 2^256. Compute the inverse by starting with a seed that is correct for
        // four bits. That is, denominator * inv = 1 mod 2^4.
        uint256 inverse = (3 * denominator) ^ 2;

        // Use the Newton-Raphson iteration to improve the precision. Thanks to Hensel's lifting lemma, this also works
        // in modular arithmetic, doubling the correct bits in each step.
        inverse *= 2 - denominator * inverse; // inverse mod 2^8
        inverse *= 2 - denominator * inverse; // inverse mod 2^16
        inverse *= 2 - denominator * inverse; // inverse mod 2^32
        inverse *= 2 - denominator * inverse; // inverse mod 2^64
        inverse *= 2 - denominator * inverse; // inverse mod 2^128
        inverse *= 2 - denominator * inverse; // inverse mod 2^256

        // Because the division is now exact we can divide by multiplying with the modular inverse of denominator.
        // This will give us the correct result modulo 2^256. Since the preconditions guarantee that the outcome is
        // less than 2^256, this is the final result. We don't need to compute the high bits of the result and prod1
        // is no longer required.
        result = prod0 * inverse;
```

```
            return result;
        }
    }

    /**
     * @notice Calculates x * y / denominator with full precision, following the selected rounding direction.
     */
    function mulDiv(
        uint256 x,
        uint256 y,
        uint256 denominator,
        Rounding rounding
    ) internal pure returns (uint256) {
        uint256 result = mulDiv(x, y, denominator);
        if (rounding == Rounding.Up && mulmod(x, y, denominator) > 0) {
            result += 1;
        }
        return result;
    }

    /**
     * @dev Returns the square root of a number. It the number is not a perfect square, the value is rounded down.
     *
     * Inspired by Henry S. Warren, Jr.'s "Hacker's Delight" (Chapter 11).
     */
    function sqrt(uint256 a) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
```

```
        // For our first guess, we get the biggest power of 2 which is smaller than the square root of the target.
        // We know that the "msb" (most significant bit) of our target number `a` is a power of 2 such that we have
        // `msb(a) <= a < 2*msb(a)`.
        // We also know that `k`, the position of the most significant bit, is such that `msb(a) = 2**k`.
        // This gives `2**k < a <= 2**(k+1)` → `2**(k/2) <= sqrt(a) < 2 ** (k/2+1)`.
        // Using an algorithm similar to the msb conmputation, we are able to compute `result = 2**(k/2)` which is a
        // good first aproximation of `sqrt(a)` with at least 1 correct bit.
        uint256 result = 1;
        uint256 x = a;
        if (x >> 128 > 0) {
            x >>= 128;
            result <<= 64;
        }
        if (x >> 64 > 0) {
            x >>= 64;
            result <<= 32;
        }
        if (x >> 32 > 0) {
            x >>= 32;
            result <<= 16;
        }
        if (x >> 16 > 0) {
            x >>= 16;
            result <<= 8;
        }
        if (x >> 8 > 0) {
            x >>= 8;
            result <<= 4;
        }
```

```
        if (x >> 4 > 0) {
            x >>= 4;
            result <<= 2;
        }
        if (x >> 2 > 0) {
            result <<= 1;
        }

        // At this point `result` is an estimation with one bit of precision. We know the true value is a uint128,
        // since it is the square root of a uint256. Newton's method converges quadratically (precision doubles at
        // every iteration). We thus need at most 7 iteration to turn our partial result with one bit of precision
        // into the expected uint128 result.
        unchecked {
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            return min(result, a / result);
        }
    }

    /**
     * @notice Calculates sqrt(a), following the selected rounding direction.
     */
    function sqrt(uint256 a, Rounding rounding) internal pure returns (uint256) {
        uint256 result = sqrt(a);
```

```
        if (rounding == Rounding.Up && result * result < a) {
            result += 1;
        }
        return result;
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/Arrays.sol)

pragma solidity ^0.8.0;

import "./math/Math.sol";

/**
 * @dev Collection of functions related to array types.
 */
library Arrays {
    /**
     * @dev Searches a sorted `array` and returns the first index that contains
     * a value greater or equal to `element`. If no such index exists (i.e. all
     * values in the array are strictly less than `element`), the array length is
     * returned. Time complexity O(log n).
     *
     * `array` is expected to be sorted in ascending order, and to contain no
     * repeated elements.
     */
    function findUpperBound(uint256[] storage array, uint256 element) internal view returns (uint256)
{
        if (array.length == 0) {
            return 0;
        }
```

```
        uint256 low = 0;
        uint256 high = array.length;

        while (low < high) {
            uint256 mid = Math.average(low, high);

            // Note that mid will always be strictly less than high (i.e. it will be a valid array index)
            // because Math.average rounds down (it does integer division with truncation).
            if (array[mid] > element) {
                high = mid;
            } else {
                low = mid + 1;
            }
        }

        // At this point `low` is the exclusive upper bound. We will return the inclusive upper bound.
        if (low > 0 && array[low - 1] == element) {
            return low - 1;
        } else {
            return low;
        }
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)

pragma solidity ^0.8.0;

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

```
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/IERC20Metadata.sol)

pragma solidity ^0.8.0;

import "../IERC20.sol";

/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 *
 * _Available since v4.1._
 */
interface IERC20Metadata is IERC20 {
    /**
```

```
    * @dev Returns the name of the token.
    */
   function name() external view returns (string memory);

   /**
    * @dev Returns the symbol of the token.
    */
   function symbol() external view returns (string memory);

   /**
    * @dev Returns the decimals places of the token.
    */
   function decimals() external view returns (uint8);
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.6.0) (token/ERC20/IERC20.sol)

pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
   /**
    * @dev Emitted when `value` tokens are moved from one account (`from`) to
    * another (`to`).
    *
    * Note that `value` may be zero.
    */
   event Transfer(address indexed from, address indexed to, uint256 value);
```

```
/**
 * @dev Emitted when the allowance of a `spender` for an `owner` is set by
 * a call to {approve}. `value` is the new allowance.
 */
event Approval(address indexed owner, address indexed spender, uint256 value);

/**
 * @dev Returns the amount of tokens in existence.
 */
function totalSupply() external view returns (uint256);

/**
 * @dev Returns the amount of tokens owned by `account`.
 */
function balanceOf(address account) external view returns (uint256);

/**
 * @dev Moves `amount` tokens from the caller's account to `to`.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
function transfer(address to, uint256 amount) external returns (bool);

/**
 * @dev Returns the remaining number of tokens that `spender` will be
 * allowed to spend on behalf of `owner` through {transferFrom}. This is
 * zero by default.
 *
 * This value changes when {approve} or {transferFrom} are called.
```

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

/**
 * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * IMPORTANT: Beware that changing an allowance with this method brings the risk
 * that someone may use both the old and the new allowance by unfortunate
 * transaction ordering. One possible solution to mitigate this race
 * condition is to first reduce the spender's allowance to 0 and set the
 * desired value afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 *
 * Emits an {Approval} event.
 */
function approve(address spender, uint256 amount) external returns (bool);

/**
 * @dev Moves `amount` tokens from `from` to `to` using the
 * allowance mechanism. `amount` is then deducted from the caller's
 * allowance.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
function transferFrom(
    address from,
    address to,
```

```
        uint256 amount
    ) external returns (bool);
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (utils/Strings.sol)

pragma solidity ^0.8.0;

/**
 * @dev String operations.
 */
library Strings {
    bytes16 private constant _HEX_SYMBOLS = "0123456789abcdef";
    uint8 private constant _ADDRESS_LENGTH = 20;

    /**
     * @dev Converts a `uint256` to its ASCII `string` decimal representation.
     */
    function toString(uint256 value) internal pure returns (string memory) {
        // Inspired by OraclizeAPI's implementation - MIT licence
        // https://github.com/oraclize/ethereum-
api/blob/b42146b063c7d6ee1358846c198246239e9360e8/oraclizeAPI_0.4.25.sol

        if (value == 0) {
            return "0";
        }
        uint256 temp = value;
        uint256 digits;
        while (temp != 0) {
            digits++;
            temp /= 10;
```

```
    }
    bytes memory buffer = new bytes(digits);
    while (value != 0) {
        digits -= 1;
        buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
        value /= 10;
    }
    return string(buffer);
}

/**
 * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation.
 */
function toHexString(uint256 value) internal pure returns (string memory) {
    if (value == 0) {
        return "0x00";
    }
    uint256 temp = value;
    uint256 length = 0;
    while (temp != 0) {
        length++;
        temp >>= 8;
    }
    return toHexString(value, length);
}

/**
 * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation with fixed length.
 */
function toHexString(uint256 value, uint256 length) internal pure returns (string memory) {
    bytes memory buffer = new bytes(2 * length + 2);
```

```
        buffer[0] = "0";
        buffer[1] = "x";
        for (uint256 i = 2 * length + 1; i > 1; --i) {
            buffer[i] = _HEX_SYMBOLS[value & 0xf];
            value >>= 4;
        }
        require(value == 0, "Strings: hex length insufficient");
        return string(buffer);
    }

    /**
     * @dev Converts an `address` with fixed length of 20 bytes to its not checksummed ASCII `string` hexadecimal representation.
     */
    function toHexString(address addr) internal pure returns (string memory) {
        return toHexString(uint256(uint160(addr)), _ADDRESS_LENGTH);
    }
}
{
  "optimizer": {
    "enabled": false,
    "runs": 200
  },
  "outputSelection": {
    "*": {
      "*": [
        "evm.bytecode",
        "evm.deployedBytecode",
        "devdoc",
        "userdoc",
        "metadata",
```

```
      "abi"
    ]
  }
 }
}
```

Contract Security Audit

[{"inputs":[],"stateMutability":"nonpayable","type":"constructor"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"owner","type":"address"},{"indexed":true,"internalType":"address","name":"spender","type":"address"},{"indexed":false,"internalType":"uint256","name":"value","type":"uint256"}],"name":"Approval","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"delegator","type":"address"},{"indexed":true,"internalType":"address","name":"fromDelegate","type":"address"},{"indexed":true,"internalType":"address","name":"toDelegate","type":"address"}],"name":"DelegateChanged","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"delegate","type":"address"},{"indexed":false,"internalType":"uint256","name":"previousBalance","type":"uint256"},{"indexed":false,"internalType":"uint256","name":"newBalance","type":"uint256"}],"name":"DelegateVotesChanged","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"previousOwner","type":"address"},{"indexed":true,"internalType":"address","name":"newOwner","type":"address"}],"name":"OwnershipTransferred","type":"event"},{"anonymous":false,"inputs":[{"indexed":false,"internalType":"uint256","name":"id","type":"uint256"}],"name":"Snapshot","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"from","type":"address"},{"indexed":true,"internalType":"address","name":"to","type":"address"},{"indexed":false,"internalType":"uint256","name":"value","type":"uint256"}],"name":"Transfer","type":"event"},{"inputs":[],"name":"DOMAIN_SEPARATOR","outputs":[{"internalType":"bytes32","name":"","type":"bytes32"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"owner","type":"address"},{"internalType":"address","name":"spender","type":"address"}],"name":"allowance","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"spender","type":"address"},{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"approve","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"}],"name":"balanceOf","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"},{"internalType":"uint256","name":"snapshotId","type":"uint256"}],"name":"balanceOfAt","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"burn","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"},{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"burnFrom","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"},{"internalType":"uint32","name":"pos","type":"uint32"}],"name":"checkpoints","outputs":[{"components":[{"internalType":"uint32","name":"fromBlock","type":"uint32"},{"internalType":"uint224","name":"votes","type":"uint224"}],"internalType":"struct ERC20Votes.Checkpoint","name":"","type":"tuple"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"decimals","outputs":[{"internalType":"uint8","name":"","type":"uint8"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"spender","type":"address"},{"internalType":"uint256","name":"subtractedValue","type":"uint256"}],"name":"decreaseAllowance","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"delegatee","type":"address"}],"name":"delegate","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"delegatee","type":"address"},{"internalType":"uint256","name":"nonce","type":"uint256"},{"internalType":"uint256","name":"expiry","type":"uint256"},{"internalType":"uint8","name":"v","type":"uint8"},{"internalType":"bytes32","name":"r","type":"bytes32"},{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"delegateBySig","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"}],"name":"delegates","outputs":[{"internalType":"address","name"

:"","type":"address"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"blockNumber","type":"uint256"}],"name":"getPastTotalSupply","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"},{"internalType":"uint256","name":"blockNumber","type":"uint256"}],"name":"getPastVotes","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"}],"name":"getVotes","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"spender","type":"address"},{"internalType":"uint256","name":"addedValue","type":"uint256"}],"name":"increaseAllowance","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"name","outputs":[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"owner","type":"address"}],"name":"nonces","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"}],"name":"numCheckpoints","outputs":[{"internalType":"uint32","name":"","type":"uint32"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"owner","outputs":[{"internalType":"address","name":"","type":"address"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"owner","type":"address"},{"internalType":"address","name":"spender","type":"address"},{"internalType":"uint256","name":"value","type":"uint256"},{"internalType":"uint256","name":"deadline","type":"uint256"},{"internalType":"uint8","name":"v","type":"uint8"},{"internalType":"bytes32","name":"r","type":"bytes32"},{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"permit","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"renounceOwnership","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"snapshot","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"symbol","outputs":[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"totalSupply","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"snapshotId","type":"uint256"}],"name":"totalSupplyAt","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"transfer","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"from","type":"address"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"transferFrom","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"newOwner","type":"address"}],"name":"transferOwnership","outputs":[],"stateMutability":"nonpayable","type":"function"}]

Contract Creation Code

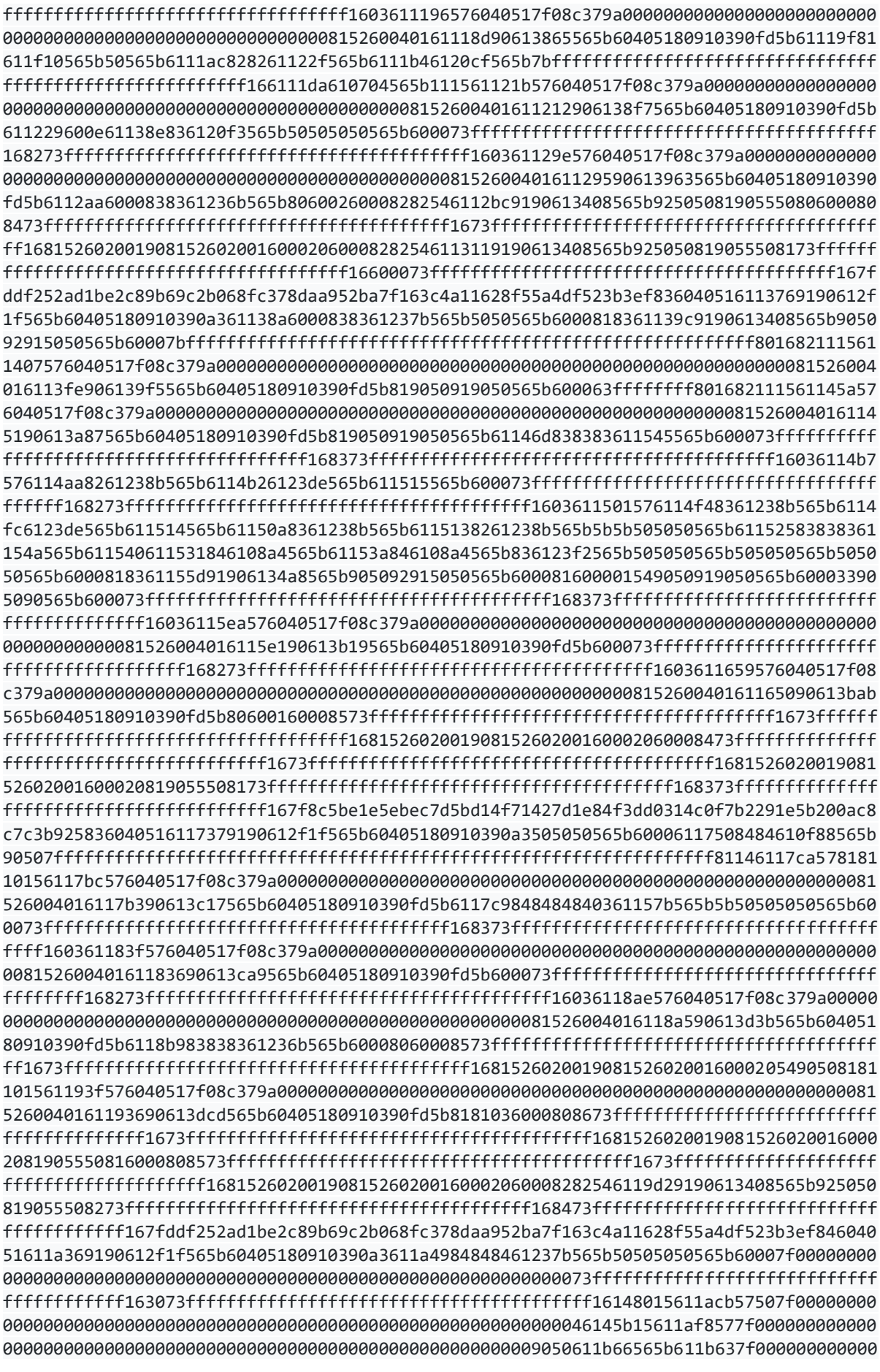

```
610140604052348015620000125760008fd5b5060405180604001604052806010815260200017f4e656f20
45636f6e6f6d7920436f696e0000000000000000000000000000000000815250806040518060400160405280
600181526020017f3100000000000000000000000000000000000000000000000000000000000000815250
604051806040016040528060108152602001 7f4e656f2045636f6e6f6d7920436f696e0000000000000000
0000000000000000815250604051806040016040528060038152602001 7f4e4543000000000000000000000
0000000000000000000000000000000000000000815250816003908162000fd9190620011cf565b50806004
90816200010f9190620011cf565b505050620001326200012662002 2b60201b60201c565b620002336020
1b60201c565b600082805190602001209050600082805190602001209050600007f8b73c3c69bb8fe3d512e
cc4cf759cc79239f7b179b0ffacaa9a75d522b39400f90508260e0818152505081610100818152505046 60
a0818152505062000 19b818484620002f960201b60201c565b60808181525050307 3ffffffffffffffffff
ffffffffffffffffffff1660c08173ffffffffffffffffffffffffffffffffffffffff16815250508061
012081815250505050505050506200022533620001fa620003356020 1b60201c565b600a62000208919062
001446565b63fa56ea00620002199190620014975 65b6200033e60201b60201c565b620018ea565b600033
905090565b60006009600090549061010 00a900473ffffffffffffffffffffffffffffffffffffffff1690
508160096000610100 0a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffff
ffffffffffffffffffffffffffffff1602179055508173ffffffffffffffffffffffffffffffffffffffff
ffff168173ffffffffffffffffffffffffffffffffffffffff167f8be0079c531659141344cd1fd0a4f284
19497f9722a3daafe3b4186f6b6457e0604051604051809103 90a35050565b60008383846306040516020
016200031695949392919062001553565b604051602081830303815290604052805190602001209050939 2
505050565b60006012905090565b6200035582826200035960201b620011a21760201c565b5050565b6200
037082826200041760201b6200122f1760201c565b62000380620005 8f60201b60201c565b7bffffffffff
```

ffffffffffffffffffffffffffffffffffffffffffff16620003ae620005b360201b60201c565b111562
0003f2576040517f08c379a0000000000000000000000000000000000000000000000000000000815260
0401620003e99062001637565b60405180910390fd5b62000411600e620005bd60201b6200138e17836200
05d560201b60201c565b50505050565b600073ffffffffffffffffffffffffffffffffffffffff168273ff
ffffffffffffffffffffffffffffffffffffff160362000489576040517f08c379a000000000000000000000
0000000000000000000000000000000000000000815260040162000480906200016a9565b60405180910390fd
5b6200049d6000838362000886602018b60201c565b806002600082825462000508919062001 6cb565b9250
508190555080600080847 3ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffff
ffffffffffffffffff168152602001908152602001600020600082825462000508919062001 6cb565b92
5050819055508173ffffffffffffffffffffffffffffffffffffffff16600073ffffffffffffffffffff
ffffffffffffffff167fddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef
836040516200056f919062001706565b60405180910390a36200058b60008383620008a360201b60201c56
5b5050565b60007bffffffffffffffffffffffffffffffffffffffffffffffffffffffff905090565b6000
6002549050905 65b60008183620005cd9190620016cb565b905092915050565b60008060008580549050 90
506000811462000 64a5785600182620005fa919062001723565b815481106200060e576200060d6200175e
565b5b9060005260206000200160000160049054906101000a90047bffffffffffffffffffffffffffff
ffffffffffffffffffffffff166200064d565b60005b7bffffffffffffffffffffffffffffffffffffffff
ffffffffffffffff16925062000679838587602 01c565b915060008111801562000 6d257504386600183
62000698919062001723565b81548110620006ac57620006ab6200175e565b5b9060005260206000200160
0001600090549061010 00a900463ffffffff1663ffffffff16145b1562000773576200 06ee82620008c060
201b620013a41760201c565b866001836200 06fe919062001723565b8154811062000712576200071162 00
175e565b5b9060005260206000200160000160046101000a8154817bffffffffffffffffffffffffffff
ffffffffffffffffffffffff02191690837bffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffff160217905550620008 7d565b856040518060400160405280620007954362000 92e60201b620014
0f1760201c565b63ffffffff16815260200162000 7b685620008c060201b620013a41760201c565b7bffff
ffffffffffffffffffffffffffffffffffffffffffffffffffff168152509080600181540180825580915 0
506001900390600052602060002001600090919091909150600082015181600001600061010 00a81548163
ffffffff021916908363ffffffff160217905550602082015181600001600046101000a8154817bffffffff
ffffffffffffffffffffffffffffffffffffffffffff02191690837bffffffffffffffffffffffffffff
ffffffffffffffffffffffffffff160217905550505 05b5093509391505 0565b6200089e838383620009
8460201b620014621760201c565b505050565b620008bb83838362000a7d60201b6200151a1760201c565b
505050565b60007bffffffffffffffffffffffffffffffffffffffffffffffffffffffff80168211156200
0926576040517f08c379a0000000000000000000000000000000000000000000000000000000081526004
016200091d9062001803565b60405180910390fd5b81905091905 0565b600063ffffffff80168211156200
097c576040517f08c379a0000000000000000000000000000000000000000000000000000000081526004
016200097390620018 9b565b60405180910390fd5b81905091905 0565b6200099c83838362000acd60201b
620015451760201c565b600073ffffffffffffffffffffffffffffffffffffffff168373ffffffffffffff
ffffffffffffffffffff1603620009f857620009e28262000ad260201b60201c565b620009f26200
0b3560201b60201c565b62000a78565b600073ffffffffffffffffffffffffffffffffffffffff168273ff
ffffffffffffffffffffffffffffffffffff160362000a545762000a3e8362000ad260201b60201c565b
62000a4e62000b3560201b60201c565b62000a77565b62000a658362000ad260201b60201c565b62000a76
8262000ad260201b60201c565b5b5b505050565b62000a9583838362000b5960201b6200154a1760201c56
5b62000ac862000aaa8462000b5e60201b60201c565b62000abb8462000b5e60201b60201c565b8362000b
c760201b60201c565b505050565b505050565b62000b32600560008373ffffffffffffffffffffffffff
ffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002 0
62000b268362000dea60201b60201c565b62000e3260201b60201c565b50565b62000b57600662000b4b62
0005b360201b60201c565b62000e3260201b60201c565b565b505050565b6000600c60008373ffffffffff
ffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001
908152602001600020600090549061010 00a900473ffffffffffffffffffffffffffffffffffffffff1690
50919050565b8173ffffffffffffffffffffffffffffffffffffffff168373ffffffffffffffffffffffff
ffffffffffffffff161415801562000c045750600081115b1562000de557600073ffffffffffffffffffff
ffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffff161462000cf757600080
62000c9e600d60008773ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffff
ffffffffffffffff168152602001908152602001600020620 00ebe60201b6200154f1785620005d56020
1b60201c565b915091508473ffffffffffffffffffffffffffffffffffffffff167fdec2bacdd2f05b59de
34da9b523dff8be42e5e38e818c82fdb0bae774387a72483836040516 2000cec929190620018bd565b6040
5180910390a250505b600073ffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffff
ffffffffffffffffffff161462000de4576000806 2000d8b600d60008673ffffffffffffffffffff

ffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001
600020620005bd60201b6200138e1785620005d560201b60201c565b915091508373ffffffffffffffff
ffffffffffffffffffff167fdec2bacdd2f05b59de34da9b523dff8be42e5e38e818c82fdb0bae774387
a724838360405162000dd9929190620018bd565b60405180910390a250505b5b505050565b600080600083
73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff
168152602001908152602001600020549050919050565b600062000e4462000ed660201b60201c565b9050
8062000e5b8460000162000ef460201b60201c565b101562000eb95782600001819080600181540180825 5
809150506001900390600052602060002001600090919091909150558260010182908060018154018082 55
809150506001900390600052602060002001600090919091909150555b505050565b6000818362000ece91
9062001723565b90509291505056 5b600062000eef600862000f4760201b620015651760201c565b905090
565b60008082805490500362000f0c5760009050620 00f42565b8160018380549050620 00f209190620017
23565b8154811062000f345762000f336200175e565b5b9060005260206000200154905 05b919050565b60
0081600001549050919050565b600081519050919050565b7f4e487b7100000000000000000000000000000
0000000000000000000000000000600052604160045260246000fd5b7f4e487b7100000000000000000000
0000000000000000000000000000000000600052602260045260246000fd5b60006002820490506001 82
168062000fd757607f821691505b60208210810362000fed5762000fec62000f8f565b5b50919050565b60
0081905081600052602060002090509190505 65b60006020601f8301049050919050565b600082821b9050
92915050565b60006008830262001057 7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffff8262001018565b6200106386836 2001018565b9550801984169350808616841792505050 93
92505050565b600081905091905056 5b600081905091905056 5b600081905091905056 5b6000620010b0620010aa620010a4846200
107b565b62001085565b6200107b565b9050919050565b600081905091905056 5b620010cc836200108f56
5b620010e4620010db82620010b7565b84845462001025565b825550505050565b600090565b620010fb62
0010ec565b62001108818484620010c1565b505050565b5b818110156200113057620011246000826200 10
f1565b60018101905062001 10e565b5050565b601f8211156200117f57620011498162000ff3565b620011
548462001008565b810160208510156200116457 8190505b6200117c6200117385620010085 65b83018262
00110d565b50505b505050565b600082821c90509291505056 5b6000620011a46000198460080262001184
565b19808316915050929150505 65b6000620011bf838362001191565b9150826002028217905092915050
565b620011da8262000f55565b67ffffffffffffffff811115620011f657620011f562000f60565b5b6200
1202825462000fbe565b6200120f828285620011345 65b600060209050601f831160018114620012475760
0084156200123257828701519050 5b6200123e8582620011b1565b865550620012ae565b601f1984166200
12578662000ff3565b60005b828110156200128157848901518255600182019150602085019450602081 01
90506200125a565b86831015620012a157848901516200129d601f8916826200119156 5b8355505b600160
028802018855505050 5b505050505050565b7f4e487b7100000000000000000000000000000000000000000000
0000000000000000600052601160045260246000fd5b60008160011c9050919050565b6000808291508390
505b60018511156200134457808604811115620013 1c576200131b620012b6565b5b60018516156200132c
5780820291505b80810290506200133c85620012e5565b9450620012fc565b945094925050505 65b600082
6200135f576001905062001432565b816200136f576000905062001432565b816001811462001388576002
811462001393576 20013c9565b60019150506200143256 5b60ff841115620013a857620013a7620012b656
5b5b8360020a915084821115620013c257 5620013c1620012b6565b5b5062001432565b5060208310610133
831016604e8410600b8410161715620014035782820a9050838111115620013fd57620013fc620012b6565b
5b62001432565b6200141284848460016200 12f2565b92509050818404811115620014 2c576200142b6200
12b6565b5b81810290505b939250505056 5b600060ff8216905091905056 5b6000620014538262001 07b56
5b9150620014608362001439565b92506200148f7fffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffff84846200134d565b90509291505056 5b6000620014a4826200107b565b91506200
14b1836200107b565b9250828202620014c1816200107b565b915082820484148315176200 14db57620014
da620012b6565b5b509291505056 5b60008190509190505 65b620014f781620014e2565b82525050565b62
001508816200107b565b82525050565b600073ffffffffffffffffffffffffffffffffffffffffffffffff82169050
919050565b60006200153b826200150e565b9050919050565b6200154d816200152e565b82525050565b60
0060a0820190506200156a6000830188620014ec565b6200157960208301876200 14ec565b620015886040
830186620014ec565b6200159760608301856200 14fd565b620015a660808301846200 1542565b96955050
50505050565b60008282526020820190509291505056 5b7f4552433230566f7465733a20746f74616c2073
7570706c79207269736b73206f60008201527f766572666c6f77696e6720766f7465730000000000000000
0000000000000000602082015250565b60006200161f603083620015b0565b91506200162c82620015c156
5b6040820190509190505 65b60006020820190508181036000830152620016528162001610565b90509190
50565b7f45524332303a206d696e7420746f20746865207a65726f206164647265737300600082015250 56
5b60006200169 1601f83620015b0565b91506200169e8262001659565b6020820190509190505 65b600060
2082019050818103600083015262001 6c48162001682565b9050919050565b6000620016d8826200107b56
5b9150620016e5836200107b565b92508282019050808211156200170057620016ff620012b6565b5b9291

5050565b6000602082019050620017 1d6000830184620014fd565b92915050565b6000620017308262001 0
7b565b91506200173d836200107b565b9250828203905081811115620017585762001757620012b6565b5b
92915050565b7f4e487b7100000000000000000000000000000000000000000000000000000000000060005260
3260045260246000fd5b7f53616665436173743a2076616c756520646f65736e27742066697420696e2032
60008201527f32342062697473000000000000000000000000000000000000000000000000006020820152
50565b6000620017eb602783620015b0565b9150620017f8826200178d565b604082019050919050565b60
0060208201905081810360008301526200181e81620017dc565b9050919050565b7f536166654361737 43a
2076616c756520646f65736e27742066697420696e203360008201527f32206269747300000000000000000
0000000000000000000000000000000000000060208201525056 5b6000620018836026836200 15b0565b9150
6200189082620018255656 5b604082019050919050565b6000602082019050818103600083015262001 8b681
62001874565b9050919050565b6000604082019050620018d4600083018562001 4fd565b620018e3602083
0184620014fd565b9392505050565b60805160a05160c05160e05161010051610120516144866200193a60
00396000611afd01526000611b3f01526000611b1e01526000611a5301526000611aa901526000611ad201
526144866000f3fe6080604052348015610010576000 80fd5b50600436106101da5760003560e01c806379
cc6790116101045780639ab24eb0116100a2578063d505accf11610071578063d505accf146105b7578063
dd62ed3e146105d3578063f1127ed814610603578063f2fde38b14610633576101da565b80639ab24eb014
61050b578063a457c2d71461053b578063a9059cbb1461056b578063c3cda5201461059b576101da565b80
638e539e8c116100de5780638e539e8c146104835780639d89b41146104b35780639711715a146104d157
8063981b24d0146104db576101da565b806379cc6790146104195780637ecebe0014610435578063 8da5cb
5b14610465576101da565b80633a46b1a81161017c5780635c19a95c1161014b5780635c19a95c14610393
5780636fcfff45146103af57806370a08231146103df578063715018a61461040f576101da565b80633a46
b1a8146102e757806342966c68146103175780634ee2cd7e14610333578063587cde1e14610363576101da
565b806323b872dd116101b857806323b872dd1461024b578063313ce567146102 7b5780633644e5151461
02995780633395093511461026b7576101da565b806306fdde03146101df578063095ea7b3146101fd578063
18160ddd1461022d575b600080fd5b6101e761064f565b6040516101f49190612ddf565b60405180910390
f35b610217600480360381019061021291906 12e9a565b6106e1565b60405161022491906 12ef5565b6040
5180910390f35b610235610704565b60405161024291906 12f1f565b60405180910390f35b610265600480
360381019061026091906 12f3a565b61070e565b60405161027291906 12ef5565b60405180910390f35b61
028361073d565b60405161029091906 12fa9565b60405180910390f35b6102a1610746565b6040516102ae
9190612fdd565b60405180910390f35b6102d160048036038101906102cc9190612e9a565b610755565b60
40516102de9190612ef5565b60405180910390f35b6103016004803603810190610 2fc9190612e9a565b61
078c565b60405161030e9190612f1f565b60405180910390f35b6103316004803603810190610 32c919061
2ff8565b610820565b005b61034d60048036038101906103489190612e9a565b610834565b60405161035a
9190612f1f565b60405180910390f35b61037d60048036038101906103789190613025565b6108a4565b60
405161038a9190613061565b60405180910390f35b6103ad60048036038101906103a89190613025565b61
090d565b005b6103c960048036038101906103c49190613025565b610921565b6040516103d6919061309b
565b60405180910390f35b6103f960048036038101906103f49190613025565b610975565b604051610406
9190612f1f565b60405180910390f35b6104176109bd565b005b6104336004803603810190610 42e919061
2e9a565b6109d1565b005b61044f60048036038101906104a9190613025565b6109f1565b60405161045c
9190612f1f565b60405180910390f35b61046d610a41565b60405161047a9190613061565b604051809103
90f35b61049d60048036038101906104989190612ff8565b610a6b565b6040516104aa9190612f1f565b60
405180910390f35b6104bb610ac1565b6040516104c89190612ddf565b60405180910390f35b6104d9610b
53565b005b6104f5600480360381019061 04f09190612ff8565b610b66565b60405161050291906 12f1f56
5b60405180910390f35b61052560048036038101906105209190613025565b610b97565b60405161053291
90612f1f565b60405180910390f35b61055560048036038101906105509190612e9a565b610ca8565b6040
516105629190612ef5565b60405180910390f35b61058560048036038101906105809190612e9a565b610d
1f565b60405161059291906 12ef5565b60405180910390f35b6105b5600480360381019061 05b0919061 31
0e565b610d42565b005b6105d1600480360381019061 05cc919061319b565b610e46565b005b6105ed6004
8036038101906105e8919061323d565b610f88565b6040516105fa9190612f1f565b60405180910390f35b
61061d600480360381019061061891906132a9565b61100f565b60405161062a919061335e565b60405180
910390f35b61064d60048036038101906106489190613025565b61111f565b005b6060600380546106 5e90
6133a8565b80601f0160208091040260200160405190810160405280929190818152602001828054610 68a
906133a8565b80156106d75780601f106106ac57610100808354040283529160200191610 6d7565b820191
9060005260206000020905b8154815290600101906020018083116106ba57829003601f168201915b505050
5050905090565b6000806106ec611573565b90506106f981858561157b565b600191505092915050565b60
0060025490509056 5b60008061071961157356 5b905061072685828561174456 5b6107318585611 7d056
5b60019150509392505050565b60006012905090565b6000610750611a4f565b905090565b600080610760
611573565b905061078181858561077285896 10f88565b61077c9190613408565b61157b565b6001915050

92915050565b60004382106107d0576040517f08c379a0000000000000000000000000000000000000000000
0000000000000000815260040161 07c790613488565b60405180910390fd5b610818600d60008573ffffff
ffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260
2001908152602001600020836 11b69565b90509291505056 5b61083161082b611573565b82611c75565b50
565b60008060006108818460056000887 3ffffffffffffffffffffffffffffffffffffffff1673ffffffff
ffffffffffffffffffffffffffff1681526020019081526020016000206 11c83565b91509150816108
985761089385610975565b61089a565b805b92505050929150505 65b6000600c60008373ffffffffffffff
ffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081
526020016000206000905490610 1000a900473ffffffffffffffffffffffffffffffffffffffff16905091
9050565b61091e610918611573565b82611d78565b50565b600061096e600d60008473ffffffffffffffff
ffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152
602001600020805490506 1140f565b9050919050565b60008060008373ffffffffffffffffffffffff
fffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002 0
549050919050565b6109c5611e92565b6109cf6000611f10565b565b6109e3826109dd611573565b836117
44565b6109ed8282611c75565b5050565b6000610a3a600a60008473ffffffffffffffffffffffffff
ffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002061
1565565b9050919050565b60006009600090549061010 00a900473ffffffffffffffffffffffffffffff
ffffffff16905090565b6000438210610aaf576040517f08c379a0000000000000000000000000000000
0000000000000000000000815260040161 0aa690613488565b60405180910390fd5b610aba600e83611b
69565b9050919050565b606060048054610ad0906133a8565b80601f0160208091040260200160405190 81
016040528092919081815260200182805461 0afc906133a8565b8015610b495780601f10610b1e57610100
808354040283529160200191610b49565b8201919060005260206000209 05b81548152906001019060200 1
808311610b2c57829003601f168201915b5050505050905090565b610b5b611e92565b610b63611fd6565b
50565b6000806000610b76846006611c83565b9150915081610b8c57610b87610704565b610b8e565b805b
9250505091905056 5b600080600d60008473ffffffffffffffffffffffffffffffffffffffff1673ffffff
ffffffffffffffffffffffffffffffff16815260200190815260200160002080549050905060008114 61
0c7f57600d60008473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffff
ffffffffffffff16815260200190815260200160002060018 2610c3391906134a8565b81548110610c44
57610c436134dc565b5b9060005260206000200160000160049054906101000a90047bffffffffffffffff
ffffffffffffffffffffffffffffffff16610c82565b60005b7bffffffffffffffffffffffff
ffffffffffffffffffffffffffff1691505091905056 5b600080610cb3611573565b90506000610cc182
86610f88565b905083811015610d06576040517f08c379a0000000000000000000000000000000000000
0000000000000815260040161 0cfd9061357d565b60405180910390fd5b610d138286840361157b
565b600192505050929150505 65b600080610d2a611573565b9050610d378185856117d0565b6001915050
92915050565b83421115610d85576040517f08c379a0000000000000000000000000000000000000000000
0000000000000815260040161 0d7c906135e9565b60405180910390fd5b6000610de7610ddf7fe4832905
7bfd03d55e49b547132e39cffd9c1820ad7b9d4c5307691425d15adf898989604051602001610dc4949392
9190613609565b60405160208183030381529060405280519060200 12061202c565b858585612046565b90
50610df281612071565b8614610e33576040517f08c379a0000000000000000000000000000000000000
0000000000000815260040161 0e2a9061369a565b60405180910390fd5b610e3d8188611d78565b50
505050505050565b83421115610e89576040517f08c379a0000000000000000000000000000000000000
0000000000000815260040161 0e809061370656 5b60405180910390fd5b60007f6e71edae12b1b97f
4d1f60370fef10105fa2faae0126114a169c64845d6126c9888888610eb88c612071565b89604051602001
610ece969594939291906 13726565b604051602081830303815290604052805190602001209050600061 0e
f18261202c565b90506000610f018287878 7612046565b90508973ffffffffffffffffffffffffffffff
ffffffff168173ffffffffffffffffffffffffffffffffffffffff1614610f71576040517f08c379a00000
0000000000000000000000000000000000000000000000000000815260040161 0f68906137d3565b604051
80910390fd5b610f7c8a8a8a61157b565b5050505050505050505050565b60006001600008473ffffffffffff
ffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190
8152602001600020600083 73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffff
ffffffffffffffff16815260200190815260200160002054905092915050565b611017612d11565b
600d60008473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffff
ffffffffff16815260200190815260200160002082 63ffffffff168154811061106e5761106d6134dc565b
5b9060005260206000200160405180604001604052908160008201600090549061010 0a900463ffffffff
1663ffffffff1663ffffffff16815260200160008201600490549061010 0a90047bffffffffffffffffff
ffffffffffffffffffffffffffffffffff167bffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffff167bffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff16815250509050
92915050565b611127611e92565b600073ffffffffffffffffffffffffffffffffffffffff168173ffffff



0000000000000000000000000000000000000000000000007f0000000000000000000000000000000000000000000000007f000000000000000000000000000000000000000000000000000000000006125eb565b90505b90565b60008083805490509050600005b81811015611be8576000611b8a8284612625565b905084868281548110611ba057611b9f6134dc565b5b90600052602060002001600001600090054906101000a900463ffffffff1663ffffffff161115611bd257809250611be2565b600181611bdf9190613408565b91505b50611b76565b60008214611c4a5784600183611bfe91906134a8565b81548110611c0f57611c0e6134dc565b5b9060005260206000200160000160049054906101000a90047bffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff16611c4d565b60005b7bffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff1692505050929150505650565b611c7f828261264b565b5050565b60008060008411611cc9576040517f08c379a000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000815260040161611d0a90613ea5565b60405180910390fd5b6000611d2b858560000161267a90919063ffffffff16565b905083600001805490508103611d4857600080925092505061d71565b60018460010182815481106611d6057611d5f6134dc565b5b906000526020600020015492509250505b9250929050565b6000611d83836108a4565b90506000611d9084610975565b905082600c60008673ffffffffffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020600061010000a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffffffff1602179055508273ffffffffffffffffffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffffffffffffffffffffffffffffff168573ffffffffffffffffffffffffffffffffffffffffffffffffffffffff167f3134e8a2e6d97e929a7e54011ea5485d7d196dd5f0ba4d4ef95803e8e3fc257f6040516040518091039 0a4611e8c8284836123f2565b50505050565b611e9a611573565b73ffffffffffffffffffffffffffffffffffffffffffffffffffffffff16611eb8610a41565b73ffffffffffffffffffffffffffffffffffffffffffffffffffffffff1614611f0e576040517f08c379a000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000815260040161f0590613f11565b60405180910390fd5b565b60006009600090549061010000a900473ffffffffffffffffffffffffffffffffffffffffffffffffffffffff1690508160096000610100a81548173ffffffffffffffffffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffffffff1602179055508173ffffffffffffffffffffffffffffffffffffffffffffffffffffffff168173ffffffffffffffffffffffffffffffffffffffffffffffffffffffff167f8be0079c531659141344cd1fd0a4f28419497f9722a3daafe3b4186f6b6457e0604051604051809103 90a35050565b6000611fe260086127535655b6000611fec612669565b90507f8030e83b04d87bef53480e26263266d6ca66863aa8506aca6f2559d18aa1cb678160405161201d9190612f1f565b60405180910390a1809150509056 5b600061203f612039611a4f565b83612769565b9050919050565b6000806000612057878787612 79c565b91509150612064816128a8565b819250505094935050505056 5b600080600a60008473ffffffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020905061 20be81611565565b91506120c981612753565b50919050565b60007bffffffffffffffffffffffffffffffffffffffffffffffffffffffff905090565b60008060008580549050905060008114612161578560018261211591906134a8565b8154811061212657612125 6134dc565b5b9060005260206000200160000160049054906101000a900463ffffffff1663ffffffff16145b15612272576121f3826113a4565b866001836122019190 6134a8565b81548110612212576122116134dc565b5b9060005260206000200160000160046101000a8154817bffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff02191690837bffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff160217905550612362565b856040518060400160405280612287436114 0f565b63ffffffff1681526020016122 9b856113a4565b7bffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff16815250908060018154018082558091505060019003906000526020600020016000909190919091506000820151816000160006101000a81548163ffffffff02191690836 3ffffffff160217905550602082015181600001600061010000a8154817bffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff02191690837bffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff1602179055505050 5b509350939150505650565b6123768383836114625 65b505050565b6123683838361151a565b505050565b6123db6005600083731ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020612 3d683610975565b612a74565b50565b6123f0600661 23eb610704565b612a74565b565b8173ffffffffffffffffffffffffffffffffffffffffffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffffffff161415801561242e5750600081115b156125e657600073ffffffffffffffffffffffffffffffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffffffffffffffffffff161461250c576000806124b5600d60008773ffffffffffffffffffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020611 54f856120f3565b915091508473ffffffffff

ffffffffffffffffffffffffffff167fdec2bacdd2f05b59de34da9b523dff8be42e5e38e818c82fdb0b
ae774387a72483836040516125019291906l3f31565b60405180910390a250505b600073ffffffffffffff
ffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffffffffffffffff16146125e55760
008061258e600d60008673ffffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffff
ffffffffffffffffff16815260200190815260200160002061138e856120f3565b915091508373ffffff
ffffffffffffffffffffffffffffff167fdec2bacdd2f05b59de34da9b523dff8be42e5e38e818c82f
db0bae774387a72483836040516125da929190613f31565b60405180910390a250505b5b505050565b6000
838383463060405160200161260695949392919061 3f5a565b60405160208183030381529060405280519
602001209050939250505056 5b60006002828418612636919061 3fdc565b828416612643919061 3408565b
90509291505056 5b6126558282612aef565b612663600e61154f836120f3565b50505050565b6000612675
6008611565565b905090565b60008083805490500361269057600090506 1274d565b600080848054905090
505b808210156126f45760006126af8383612625565b90508486828154811061 26c5576126c46134dc565b
5b906000526020600020015411156126de578091506126ee565b6001816126eb9190613408565b92505b50
61269b565b6000821180156127 2c57508385600184612 70e91906134a8565b8154811061271f5761271e61
34dc565b5b906000526020600020015414 5b15612747576001826 1273e91906134a8565b9250505061274d
565b81925050505b92915050565b60018160000160008282540192505081905550505 65b60008282604051
60200161277e929190614085565b60405160208183030381529060405280519060200120905092915050 56
5b6000807f7fffffffffffffffffffffffffffff5d576e7357a4501ddfe92f46681b20a08360001c1115
6127d75760006003915091506 1289f565b601b8560ff16141580156127ef5750601c8560ff1614155b1561
280157600060049150915061289f565b60006001878787876040516000815260200160405260405161282 6
94939291906140bc565b60206040516020810390808403908 55afa158015612848573d6000803e3d6000fd
5b505050602060405103519050600073ffffffffffffffffffffffffffffffffffffffff168173ffffffff
ffffffffffffffffffffffffffff160361289657600060019250925050612 89f565b80600092509250
505b945094925050505 65b60006004811115612 8bc576128bb614101565b5b8160048111156128cf576128
ce614101565b5b0315612a715760016004811115612 8e9576128e8614101565b5b8160048111156128fc57
6128fb614101565b5b0361293c576040517f08c379a000000000000000000000000000000000000000000000
0000000000000815260040161293906141 7c565b60405180910390fd5b60026004811115612950576129
4f614101565b5b8160048111156129635761296261410 1565b5b036129a3576040517f08c379a000000000
0000000000000000000000000000000000000000000815260040161299a906141e8565b6040518091
0390fd5b60036004811115612 9b7576129b6614101565b5b8160048111156129ca576129c9614101565b5b
03612a0a576040517f08c379a000000000000000000000000000000000000000000000000000000000008152
600401612a019061427a565b60405180910390fd5b600480811115612a1d57612a1c614101565b5b816004
811115612a3057612a2f614101565b5b03612a70576040517f08c379a000000000000000000000000000000000
0000000000000000000000000008152600401612a679061430c565b60405180910390fd5b5b50565b6000
612a7e612669565b905080612a8d84600001612cc5565b1015612aea578260000181908060018154018082
558091505060019003906000526020600020016000909190919091505582600101829080600181540180825
558091505060019003906000526020600020016000909190919091505555b505050565b600073ffffffffff
ffffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffffffffffffffff1603612b5e
576040517f08c379a000000000000000000000000000000000000000000000000000000000008152600401612
2b559061439e565b60405180910390fd5b612b6a8260008361236b565b60008060008473ffffffffffffff
ffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffffff1681526020019081
526020016000205490508181101 5612bf0576040517f08c379a000000000000000000000000000000000000
0000000000000000000008152600401612be790614430565b60405180910390fd5b8181036000808573ff
ffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffffff1681
526020019081526020016000208190555081600260008282546 12c4791906134a8565b9250508190555060
0073ffffffffffffffffffffffffffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffff
ffff167fddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef84604051612cac
9190612f1f565b60405180910390a3612cc08360008461237b565b505050565b60008082805490500 3612c
db5760009050612d0c565b816001838054905061 2ced91906134a8565b81548110612cfe57612cfd6134dc
565b5b906000526020600020015490505b919050565b60405180604001604052806000 63ffffffff168152
60200160007bffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff1681525090565b6000
8151905091905056 5b60008282526020820190509291505056 5b60005b83811015612d8957808201518184
01526020819050612d6e565b60008484015250505050565b6000601f19601f830116905091905056 5b60
00612db182612d4f565b612dbb8185612d5a565b9350612dcb818560208601612d6b565b612dd481612d95
565b840191505092915050565b60006020820190508181036000830152612df98184612da6565b90509291
5050565b600080fd5b600073ffffffffffffffffffffffffffffffffffffffff82169050919050565b6000
612e3182612e06565b9050919050565b612e4181612e26565b8114612e4c57600080fd5b50565b60008135
9050612e5e81612e38565b92915050565b60008190509190 50565b612e7781612e64565b8114612e825760

0080fd5b50565b60008135905061 2e9481612e6e565b92915050565b6000806040838503121561 2eb15761
2eb0612e01565b5b6000612ebf85828601612e4f565b9250506020612ed085828601612e85565b91505092
50929050565b60008115159050919050565b612eef81612eda565b82525050565b6000602082019050612f
0a6000830184612ee6565b92915050565b612f1981612e64565b82525050565b6000602082019050612f34
6000830184612f10565b92915050565b60008060006060848603121561 2f5357612f52612e01565b5b6000
612f6186828701612e4f565b9350506020612f7286828701612e4f565b9250506040612f83868287016 12e
85565b915050925092509256 5b600060ff8216905091905056 5b612fa381612f8d565b82525050565b6000
602082019050612fbe6000830184612f9a565b92915050565b60008190509190505 65b612fd781612fc456
5b82525050565b6000602082019050612ff26000830184612fce565b92915050565b600602082842031215
61300e5761300d612e01565b5b600061301c84828501612e85565b9150509291505056 5b60006020828403
121561303b5761303a612e01565b5b60006130498428501612e4f565b9150509291505056 5b61305b8161
2e26565b82525050565b60006020820190506130766000830184613052565b92915050565b600063ffffff
ff8216905091905056 5b6130958161307c565b82525050565b60006020820190506130b060008301846130
8c565b92915050565b6130bf81612f8d565b81146130ca57600080fd5b50565b60008135905061 30dc8161
30b6565b92915050565b6130eb81612fc4565b81146130f657600080fd5b50565b60008135905061 310881
6130e2565b92915050565b60008060008060008060c087890312156 1312b5761312a612e01565b5b600061
313989828a01612e4f565b965050602061314a89828a01612e85565b955050604061315b89828a01612e85
565b945050606061316c89828a016130cd565b935050608061317d89828a016130f9565b92505060a06131
8e89828a016130f9565b9150509295509295509295565b60008060008060008060060e0888a0312156131
ba576131b9612e01565b5b60006131c88a828b01612e4f565b975050602061 31d98a828b01612e4f565b96
505060406131ea8a828b01612e85565b955050606061 31fb8a828b01612e85565b945050608061320c8a82
8b016130cd565b93505060a061321d8a828b016130f9565b92505060c061322e8a828b016130f9565b9150
5092959891949750929550565b60008060408385031215613254576132536 12e01565b5b60006132628582
8601612e4f565b9250506020613273828601612e4f565b9150509250929050565b6132868161307c565b
81146132915760080fd5b50565b60008135905061 32a38161327d565b92915050565b60008060408385 03
12156132c0576132bf612e01565b5b60006132ce85828601612e4f565b92505060206132df858286016132
94565b9150509250929050565b6132f28161307c565b82525050565b60007bffffffffffffffffffffffff
ffffffffffffffffffffffffffffffff8216905091905056 5b613329816132f8565b82525050565b604082
0160008201516133456000850182613 2e9565b5060208201516133586020850182613320565b5050505056
5b60006040820190506133736000830184613 32f565b92915050565b7f4e487b7100000000000000000000
0000000000000000000000000000000000000000600052602260045260246000fd5b60006002820490506001 82
16806133c057607f821691505b6020821081036133d3576133d2613379565b5b50919050565b7f4e487b71
0000000000000000000000000000000000000000000000000000000000000000600052601160045260246000fd5b60
0061341382612e64565b915061341e83612e64565b92508282019050808211156134365761343 56133d956
5b5b92915050565b7f4552433230566f7465733a20626c6f636b206e6f7420796574206d696e6564006000
82015250565b60006134726 01f83612d5a565b915061347d8261343c565b6020820190509190505 65b6000
6020820190508181036000830 1526134a181613465565b9050919050565b60006134b382612e64565b9150
6134be83612e64565b92508282039050818111156134d6576134d56133d9565b5b92915050565b7f4e487b
7100000000000000000000000000000000000000000000000000000000000000600052603260045260246000fd5b
7f45524332303a20646563726561736420616c6c6f77616e63652062656c6f7760008201527f207a6572
6f0000000000000000000000000000000000000000000000000000000000006020820152505 65b60006135676025
83612d5a565b9150613572826 1350b565b604082019050919050565b60006020820190508181036000830 1
526135968161355a565b9050919050565b7f4552433230566f7465733a207369676e617475726520657870
6972656400000060008201525 0565b60006135d3601d83612d5a565b91506135de8261359d565b60208201
9050919050565b60006020820190508181036000830152613602816135c6565b9050919050565b60006080
8201905061361e6000830187612fce565b61362b6020830186613052565b6136386040830185612f10565b
6136456060830184612f10565b9594505050505056 5b7f4552433230566f7465733a20696e76616c696420
6e6f6e6365000000000000060008201525 0565b60006136846019836 12d5a565b915061368f8261364e56
5b6020820190509190505 65b60006020820190508181036000830 1526136b3816136775 65b905091905056
5b7f4552433230506572 6d69743a20657870697265642064656 1646c696e650000006000820152505 65b60
006136f0601d83612d5a565b91506136fb826136ba565b6020820190509190505 65b60006020820190508 1
8103600083015261371f816136e3565b9050919050565b600060c08201905061373b6000830189612fce56
5b6137486020830188613052565b6137556040830187613052565b6137626060830186612f10565b61376f
6080830185612f10565b61377c60a0830184612f10565b9796505050505050565b7f4552433230506572
6d69743a20696e76616c6964207369676e617475726500006000820152505 65b60006137bd601e83612d5a
565b91506137c882613787565b6020820190509190505 65b60006020820190508181036000830 1526137ec
816137b0565b9050919050565b7f4f776e61626c653a206e6577206f776e657220697320746865207a6572
6f206160008201527f6464726573730000000000000000000000000000000000000000000000000000006020

82015250565b600061384f602683612d5a565b915061385a826137f3565b6040820190509190505650565b6000
60208201905081810360008301526138 7e81613842565b9050919050565b7f4552433230566f7465733a20
746f74616c20737570706c79207269736b73206f60008201527f766572666c6f77696e6720766f74657300
0000000000000000000000000000000060208201525056 5b60006138e1603083612d5a565b91506138ec8261
3885565b6040820190509190505650565b60006020820190508181036000830152613910816138d4565b905091
9050565b7f45524332303a206d696e7420746f20746865207a65726f206164647265737300600082015250
565b600061394d601f83612d5a565b9150613958826139179 17565b602082019050919050565b600060208201
905081810360008301526139 7c81613940565b9050919050565b7f5361666543617374 3a2076616c756520
646f65736e2774206669742069 6e20326000820152 7f32342062697473000000000000000000000000000000
0000000000000000000000060208201525056 5b60006139df602783612d5a565b91506139ea82613983565b
6040820190509190505650 5b60006020820190508181036000830152613a0e816139d2565b9050919050565b
7f5361666543617374 3a2076616c756520646f65736e2774206669742069 6e203360008201527f32206269
747300000000000000000000000000000000000000000000000000000000602082015250565b6000613a716026
83612d5a565b9150613a7c82613a15565b6040820190509190505650 5b60006020820190508181036000830 1
52613aa081613a64565b9050919050565b7f45524332303a20617070726f76652066726f6d20746865207a
65726f206164646000820 1527f726573730000000000000000000000000000000000000000000000000000
0000602082015250565b6000613b03602483612d5a565b9150613b0e82613aa7565b6040820190509190 50
565b60006020820190508181036000830152613b3281613af6565b9050919050565b7f45524332303a2061
7070726f766520746f20746865207a65726f206164647265600082015 27f737300000000000000000000000
0000000000000000000000000000000000000000602082015250565b6000613b95602283612d5a565b915061
3ba082613b39565b6040820190509190505650 5b60006020820190508181036000830152613bc481613b8856
5b9050919050565b7f45524332303a20696e737566666963696 56e7420616c6c6f77616e6365000000600 0
82015250565b6000613c01601d83612d5a565b9150613c0c82613bcb565b6020820190509190505650 5b6000
60208201905081810360008301526138 13c3081613bf4565b9050919050565b7f45524332303a207472616e73
6665722066726f6d20746865207a65726f206164000820 1527f647265737300000000000000000000000000
0000000000000000000000000000000060208201525056 5b6000613c93602583612d5a565b9150613c9e8261
3c37565b6040820190509190505650 5b60006020820190508181036000830152613cc281613c86565b905091
9050565b7f45524332303a207472616e7366657220746f20746865207a65726f206164647260008201527f
6573730000000000000000000000000000000000000000000000000000000000000000602082015250565b600061
3d25602383612d5a565b9150613d3082613cc9565b6040820190509190505650 5b60006020820190508181103
6000830152613d5481613d18565b9050919050565b7f45524332303a207472616e7366657220616d6f756e
7420657863656564732062600082 01527f616c616e6365000000000000000000000000000000000000000000
0000000000006020820152 15250565b6000613db7602683612d5a565b9150613dc282613d5b565b6040820190
50919050565b6000602082019050818103600083015261 3de681613daa565b9050919050565b7f45524332
30536e617073686f743a206964206973203000000000000000000000600082015250565b6000613e236016
83612d5a565b9150613e2e82613ded565b6020820190509190505650 5b60006020820190508181036000830 1
52613e5281613e16565b9050919050565b7f4552433230536e617073686f743a206e6f6e6578697374656e
7420696400000060008201525056 5b6000613e8f601d83612d5a565b9150613e9a82613e59565b60208201
9050919050565b6000602082019050818103600083015261 3ebe81613e82565b9050919050565b7f4f776e
61626c653a2063616c6c6572206973206e6f7420746865206f776e657260008201525056 5b6000613efb60
2083612d5a565b9150613f0682613ec5565b6020820190509190505650 5b60006020820190508181036000830
0152613f2a81613eee565b9050919050565b6000604082019050613f466000830185612f10565b613f5360
20830184612f10565b9392505050565b600060a082019050613f6f6000830188612fce565b613f7c602083
0187612fce565b613f896040830186612fce565b613f966060830185612f10565b613fa360808301846130
52565b969550505050505050565b7f4e487b7100000000000000000000000000000000000000000000000000000000
0000006000526012600452602460 00fd5b6000613fe782612e64565b9150613ff283612e64565b92508261
400257614001613fad565b5b828204905092915050565b60008190509291505056 5b7f1901000000000000
0000000000000000000000000000000000000000000000000000600082015250565b600061404e60028361400d
565b9150614059826140185656 5b60028201905091905056 5b60008190509190505650 5b61407f61407a82612f
c4565b614064565b82525050565b60006140908261404156 5b915061409c828561406e565b602082019150
6140ac828461406e565b6020820191508190509392505050565b60006080820190506140d1600083018761
2fce565b6140de6020830186612f9a565b6140eb6040830185612fce565b6140f8606083018461 2fce565b
9594505050505050565b7f4e487b7100000000000000000000000000000000000000000000000000000000000060
0052602160045260246000fd5b7f45434453413a20696e76616c6964207369676e617457265000000000000
0000006000820152 15250565b60006141666601883612d5a565b9150614171826141305656 5b6020820190509190
50565b60006020820190508181036000830152614195816141595656 5b9050919050565b7f45434453413a20
696e76616c6964207369676e6174757265206c656e6774680060008201525056 5b60006141d2601f83612d
5a565b91506141dd8261419c565b6020820190509190505650 5b60006020820190508181036000830152614 2

01816141c5565b9050919050565b7f45434453413a20696e76616c6964207369676e617475726520277327
2076616c60008201527f7565000000000000000000000000000000000000000000000000000000000060
2082015250565b60006142646022836 12d5a565b915061426f82614208565b6040820190509190505 65b60
0060208201905081810360008301526142938161425 7565b9050919050565b7f45434453413a20696e7661
6c6964207369676e617475726520277627207661 6c60008201527f7565000000000000000000000000000000000
0000000000000000000000000000000060208201525 0565b60006142f6602283612d5a565b915061430182
61429a565b6040820190509190505 65b60006020820190508181036000830152614325816142e9565b9050
919050565b7f45524332303a206275726e2066726f6d20746865207a65726f206164647265736000820152
7f730000000000000000000000000000000000000000000000000000000000000000000000000000602082015250565b6000
614388602183612d5a565b91506143938261432c565b6040820190509190505 65b60006020820190508181
036000830152614 3b78161437b565b9050919050565b7f45524332303a206275726e20616d6f756e742065
7863656564732062616c616e60008201527f6365000000000000000000000000000000000000000000000000000000000000
0000000000000060208201525 0565b600061441a602283612d5a565b9150614425826143be565b60408201
9050919050565b60006020820190508181036000830152614449816144 0d565b905091905056fea2646970
6673582212200944f10e3c6ec0635bc1ad514b0309313f69c144087096af3d845ef6d0ff57e264736f6c63
430008120033