
Armors Labs
DeFiFox
Smart Contract Audit

DeFiFox Audit Summary

Project name : DeFiFox Contract

Project address: None

Code URL : https://www.oklink.com/okexchain/address/0xecfbe19cff5d63331f4eb5c5a05d0ec136c424c3

Code URL : https://www.oklink.com/okexchain/address/0xb9913548f780308bc95d30b035b894fee8c98e3d

Code URL : https://www.oklink.com/okexchain/address/0x4d2a3e33e9f6140eabf7d5723ce7aa4636be7370

Code URL : https://www.oklink.com/okexchain/address/0x77e13d541b27bb7773e6f9a82535b8b700851bfc

Code URL : https://www.oklink.com/okexchain/address/0xc455663f576667e588a1100ad40c160aae530a91

Code URL : https://www.oklink.com/okexchain/address/0x398a200cc7cd39a27954d7eed23eba74e1330112

Code URL : https://www.oklink.com/okexchain/address/0x79e04db01b1ff7c35769cce80f69841c1fec37e8

Code URL : https://www.oklink.com/okexchain/address/0x50e7a850da06a7fe98a084b64b32d27e360bda3e

Code URL : https://www.oklink.com/okexchain/address/0x059eae59d17666539de04ddb7bf4345f4a55de27

Code URL : https://www.oklink.com/okexchain/address/0x53788793bd37e12bced07f846e47d7e2d32ef3fe

Code URL : https://www.oklink.com/okexchain/address/0xa8f5266dede5bdd97ef2eeb764b8ef72e824ea58

Code URL : https://www.oklink.com/okexchain/address/0x94b70bff70eca942df6e5a4c44ae9c2e35c777ec

Code URL : https://www.oklink.com/okexchain/address/0x47403265ebd48c7c03cfe4fc6f9cf5dce9ef2a6e

Code URL : https://www.oklink.com/okexchain/address/0x0d61205585dbd825f24d70103f55152abc0d15aa

Code URL : https://www.oklink.com/okexchain/address/0x8aa638834c40ea3d77161a98f7a0864437086789

Code URL : https://www.oklink.com/okexchain/address/0x9a9d4c36efbd79358448d24a5ddc565907744a21

Code URL : https://www.oklink.com/okexchain/address/0x56f3d88353e44ddba196522e15263d7543743273

Code URL : https://www.oklink.com/okexchain/address/0x2eb3253e5c189aedc94fc73e58d80809e1ae2345

Commit : None

Project target : DeFiFox Contract Audit

Blockchain : OKExChain

Test result : PASSED

Audit Info

Audit NO : 0X202108060026

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

DeFiFox Audit

The DeFiFox team asked us to review and audit their DeFiFox contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
DeFiFox Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-08-06

Audit results

Notice:

1.StrategyV2Pair::_withdraw, StrategyV2Pair::_deposit Contract has whiteList.

2.DFOXPools::deposit,TenBankHall::depositLPToken,TenBankHall::deposit,SafeBoxFoxCToken::_deposit,Contract has Blacklist.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the DeFiFox contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
DFOXPools.sol	3e25fd50d73d373007b6a432fe9070cb
MultiSourceOracle::TransparentUpgradeableProxy.sol	5a1650b709c912c97614039c9d08b193

file	md5
SafeBoxFox::TransparentUpgradeableProxy.sol	5a1650b709c912c97614039c9d08b193
MultiSourceOracle.sol	8337e167c59906cdccd4e498cb8b70fb
StrategyV2Pair::TransparentUpgradeableProxy.sol	5a1650b709c912c97614039c9d08b193
SafeBoxFox.sol	3df3529da5f6d2f20d88b6ba86c16dbf
Timelock.sol	d8802ec36bcca7cac23ec9de6767447c
StrategyV2PairHelper.sol	46623a1e3bd3838a9e0b5cd0931d52c6
StrategyV2Pair.sol	5573ba86619a394b39e594fe5e1152da
ActionCompPools.sol	f22d9b1ef27eae976a9a0eea2e068042
TenBankHall.sol	a812e542a8a425f1fe8107306e0d1dba
StrategyConfig.sol	d023c99fa5e1376b9b85f249a62fc751
PriceCheckerLPToken::TransparentUpgradeableProxy.sol	5a1650b709c912c97614039c9d08b193
TenBankHall::TransparentUpgradeableProxy.sol	5a1650b709c912c97614039c9d08b193
PriceCheckerLPToken.sol	7a397e730808e8556d119c7ba3d5e7a5
ActionCompPools::TransparentUpgradeableProxy.sol	5a1650b709c912c97614039c9d08b193
DFOXPools::TransparentUpgradeableProxy.sol	5a1650b709c912c97614039c9d08b193
StrategyV2CherrySwapPool.sol	72749d6192cf0b9beea561a671922af2

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe

Vulnerability	status
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

DFOXPools.sol

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20Upgradeable {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
```

```
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMathUpgradeable {
    /**
```

```
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
```

```
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}


pragma solidity >=0.6.2 <0.8.0;
```

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

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
```

```
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

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

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
```

```
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) intern
        require(isContract(target), "Address: static call to non-contract");

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

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

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


pragma solidity >=0.6.0 <0.8.0;


/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20Upgradeable {
    using SafeMathUpgradeable for uint256;
    using AddressUpgradeable for address;

    function safeTransfer(IERC20Upgradeable token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20Upgradeable token, address from, address to, uint256 value) inter
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20Upgradeable token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
```

```
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20Upgradeable token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}


// solhint-disable-next-line compiler-version
pragma solidity >=0.4.24 <0.8.0;


/**
 * @dev This is a base contract to aid in writing upgradeable contracts, or any kind of contract that
 * behind a proxy. Since a proxied contract can't have a constructor, it's common to move constructor
 * external initializer function, usually called `initialize`. It then becomes necessary to protect t
 * function so it can only be called once. The {initializer} modifier provided by this contract will
 *
 * TIP: To avoid leaving the proxy in an uninitialized state, the initializer function should be call
 * possible by providing the encoded function call as the `_data` argument to {UpgradeableProxy-const
 *
 * CAUTION: When used with inheritance, manual care must be taken to not invoke a parent initializer
 * that all initializers are idempotent. This is not verified automatically as constructors are by So
 */
abstract contract Initializable {

    /**
     * @dev Indicates that the contract has been initialized.
     */
    bool private _initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private _initializing;
```

```
    /**
     * @dev Modifier to protect an initializer function from being invoked twice.
     */
    modifier initializer() {
        require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is alrea

        bool isTopLevelCall = !_initializing;
        if (isTopLevelCall) {
            _initializing = true;
            _initialized = true;
        }

        _;

        if (isTopLevelCall) {
            _initializing = false;
        }
    }

    /// @dev Returns true if and only if the function is running in the constructor
    function _isConstructor() private view returns (bool) {
        return !AddressUpgradeable.isContract(address(this));
    }
}


pragma solidity >=0.6.0 <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 GSN 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 ContextUpgradeable is Initializable {
    function __Context_init() internal initializer {
        __Context_init_unchained();
    }

    function __Context_init_unchained() internal initializer {
    }
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
    uint256[50] private __gap;
}


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
```

```
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract OwnableUpgradeable is Initializable, ContextUpgradeable {
    address private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    function __Ownable_init() internal initializer {
        __Context_init_unchained();
        __Ownable_init_unchained();
    }

    function __Ownable_init_unchained() internal initializer {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
    uint256[49] private __gap;
}


pragma solidity 0.6.12;
```

```
interface IActionTrigger {
    function getATPoolInfo(uint256 _pid) external view
        returns (address lpToken, uint256 allocRate, uint256 totalAmount);
    function getATUserAmount(uint256 _pid, address _account) external view
        returns (uint256 acctAmount);
}

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

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

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

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

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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


pragma solidity 0.6.12;


interface IActionPools {

    function getPoolInfo(uint256 _pid) external view
        returns (address callFrom, uint256 callId, address rewardToken);
    function mintRewards(uint256 _callId) external;
    function getPoolIndex(address _callFrom, uint256 _callId) external view returns (uint256[] memory

    function onAcionIn(uint256 _callId, address _account, uint256 _fromAmount, uint256 _toAmount) ext
    function onAcionOut(uint256 _callId, address _account, uint256 _fromAmount, uint256 _toAmount) ex
    function onAcionClaim(uint256 _callId, address _account) external;
    function onAcionEmergency(uint256 _callId, address _account) external;
    function onAcionUpdate(uint256 _callId) external;
}

pragma solidity 0.6.12;

interface IClaimFromBank {
    function claimFromBank(address _account, uint256[] memory _pidlist) external returns (uint256 val
}

pragma solidity >=0.6.0 <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 GSN 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 payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
}


pragma solidity >=0.6.0 <0.8.0;
```

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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

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

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

```
pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
```

```
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
```

```
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
```

```
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;

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

```
        _name = name_;
        _symbol = symbol_;
        _decimals = 18;
    }

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

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

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual returns (uint8) {
        return _decimals;
    }

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

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

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

    /**
     * @dev See {IERC20-allowance}.
```

```
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256)
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
```

```
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC2
        return true;
    }

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

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

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

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

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

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

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

```
        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

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

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

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal virtual {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks]
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev Extension of {ERC20} that adds a cap to the supply of tokens.
 */
abstract contract ERC20Capped is ERC20 {
    using SafeMath for uint256;

    uint256 private _cap;
```

```
    /**
     * @dev Sets the value of the `cap`. This value is immutable, it can only be
     * set once during construction.
     */
    constructor (uint256 cap_) internal {
        require(cap_ > 0, "ERC20Capped: cap is 0");
        _cap = cap_;
    }

    /**
     * @dev Returns the cap on the token's total supply.
     */
    function cap() public view virtual returns (uint256) {
        return _cap;
    }

    /**
     * @dev See {ERC20-_beforeTokenTransfer}.
     *
     * Requirements:
     *
     * - minted tokens must not cause the total supply to go over the cap.
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual override
        super._beforeTokenTransfer(from, to, amount);

        if (from == address(0)) { // When minting tokens
            require(totalSupply().add(amount) <= cap(), "ERC20Capped: cap exceeded");
        }
    }
}


pragma solidity 0.6.12;

interface IRewardsToken {
    function mint(address _account, uint256 _amount) external;
    function burn(uint256 _amount) external;

    function setMintWhitelist(address _account, bool _enabled) external;
    function checkWhitelist(address _account) external view returns (bool);
}

pragma solidity 0.6.12;



contract DFOXToken is ERC20Capped, Ownable, IRewardsToken {
    constructor (
            string memory _name,
            string memory _symbol,
            uint256 _totalSupply
        ) public ERC20Capped(_totalSupply) ERC20(_name, _symbol) {
    }

    mapping(address => bool) public mintWhitelist;

    function setMintWhitelist(address _account, bool _enabled) external override onlyOwner {
        mintWhitelist[_account] = _enabled;
    }

    function checkWhitelist(address _account) external override view returns (bool) {
        return mintWhitelist[_account];
    }

    function mint(address _account, uint256 _amount) external override {
```

```
        require(mintWhitelist[msg.sender], 'not allow');
        _mint(_account, _amount);
    }

    function burn(uint256 _amount) external override onlyOwner {
        _burn(msg.sender, _amount);
    }
}



pragma solidity 0.6.12;



// Note that it's ownable and the owner wields tremendous power. The ownership
// will be transferred to a governance smart contract once Token is sufficiently
// distributed and the community can show to govern itself.
//
// Have fun reading it. Hopefully it's bug-free. God bless.
contract DFOXPools is OwnableUpgradeable, IActionTrigger, IClaimFromBank {
    using SafeMathUpgradeable for uint256;
    using SafeERC20Upgradeable for IERC20Upgradeable;

    // Info of each user.
    struct UserInfo {
        uint256 amount;         // How many LP tokens the user has provided.
        uint256 rewardDebt;     // Reward debt. See explanation below.
        uint256 rewardRemain;   // Remain rewards
        //
        // We do some fancy math here. Basically, any point in time, the amount of Token
        // entitled to a user but is pending to be distributed is:
        //
        //   pending reward = (user.amount * pool.accRewardPerShare) + user.rewardRemain - user.rewar
        //
        // Whenever a user deposits or withdraws LP tokens to a pool. Here's what happens:
        //   1. The pool's `accRewardPerShare` (and `lastRewardBlock`) gets updated.
        //   2. User calc the pending rewards and record at rewardRemain.
        //   3. User's `amount` gets updated.
        //   4. User's `rewardDebt` gets updated.
    }

    // Info of each pool.
    struct PoolInfo {
        address lpToken;             // Address of LP token contract.
        uint256 allocPoint;         // How many allocation points assigned to this pool. Token to dis
        uint256 lastRewardBlock;    // Last block number that Token distribution occurs.
        uint256 accRewardPerShare;  // Accumulated Token per share, times 1e18. See below.
        uint256 totalAmount;        // Total amount of current pool deposit.
    }

    // The FOX TOKEN!
    DFOXToken public rewardToken;
    // FOX tokens created per block.
    uint256 public rewardPerBlock;
    address public devaddr;

    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping (uint256 => mapping (address => UserInfo)) public userInfo;
    // Total allocation points. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when reward Token mining starts.
    uint256 public startBlock;
    // The extend Pool
```

```
    address public extendPool;

    // block hacker user to deposit
    mapping (address => bool) public depositBlacklist;
    // block hacker user to restricted reward
    mapping (address => uint256) public rewardRestricted;
    // enable pool emergency withdraw
    mapping (uint256 => bool) public emergencyWithdrawEnabled;
    // allow bank proxy claim
    address public bank;

    event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
    event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
    event Claim(address indexed user, uint256 indexed pid, uint256 amount);
    event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

    function initialize(
        address _bank,
        address _rewardToken,
        uint256 _rewardPerBlock,
        address _devaddr,
        uint256 _startBlock) public initializer {
        __Ownable_init();

        bank = _bank;
        rewardToken = DFOXToken(_rewardToken);
        startBlock = _startBlock;
        devaddr = _devaddr;
        rewardPerBlock = _rewardPerBlock;
    }

    function poolLength() external view returns (uint256) {
        return poolInfo.length;
    }

    function getATPoolInfo(uint256 _pid) external override view
        returns (address lpToken, uint256 allocRate, uint256 totalAmount) {
        lpToken = poolInfo[_pid].lpToken;
        totalAmount = poolInfo[_pid].totalAmount;
        if(totalAllocPoint > 0) {
            allocRate = poolInfo[_pid].allocPoint.mul(1e9).div(totalAllocPoint);
        }else{
            allocRate = 1e9;
        }
    }

    function getATUserAmount(uint256 _pid, address _account) external override view
        returns (uint256 acctAmount) {
        acctAmount = userInfo[_pid][_account].amount;
    }

    // Add a new lp to the pool. Can only be called by the owner.
    function add(uint256 _allocPoint, address _lpToken) public onlyOwner {
        massUpdatePools();
        require(IERC20Upgradeable(_lpToken).totalSupply() >= 0, 'error lptoken address');
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        poolInfo.push(PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accRewardPerShare: 0,
            totalAmount: 0
        }));
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
    }
```

```
    // Set the number of reward produced by each block
    function setRewardPerBlock(uint256 _rewardPerBlock) external onlyOwner {
        massUpdatePools();
        rewardPerBlock = _rewardPerBlock;
    }

    function setExtendPool(address _extendPool) external onlyOwner {
        extendPool = _extendPool;
    }

    // Update the given pool's Token allocation point. Can only be called by the owner.
    function setAllocPoint(uint256 _pid, uint256 _allocPoint) external onlyOwner {
        massUpdatePools();
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // Update dev address by the previous one
    function setDevAddress(address _devaddr) external {
        require(msg.sender == devaddr, "only dev caller");
        devaddr = _devaddr;
    }

    function setPoolStartBlock(uint256 _pid, uint256 _startBlock) external onlyOwner {
        PoolInfo storage pool = poolInfo[_pid];
        require(block.number < pool.lastRewardBlock, 'set too late');
        require(_startBlock  > block.number, 'startBlock error');
        poolInfo[_pid].lastRewardBlock = _startBlock;
    }

    // block hacker user to deposit
    function setBlacklist(address _hacker, bool _set) external onlyOwner {
        depositBlacklist[_hacker] = _set;
    }

    function setEmergencyWithdrawEnabled(uint256 _pid, bool _set) external onlyOwner {
        emergencyWithdrawEnabled[_pid] = _set;
    }

    // block hacker user to restricted reward
    function setRewardRestricted(address _hacker, uint256 _rate) external onlyOwner {
        require(_rate <= 1e9, 'max is 1e9');
        rewardRestricted[_hacker] = _rate;
    }

    // Return reward multiplier over the given _from to _to block.
    function getBlocksReward(uint256 _from, uint256 _to) public view returns (uint256 value) {
        require(_from <= _to, 'getBlocksReward error');
        if (_to < startBlock) {
            return 0;
        }
        if (_from < startBlock && _to >= startBlock) {
            value = getBlocksReward(startBlock, _to);
        } else {
            value = _to.sub(_from).mul(rewardPerBlock);
        }
    }

    // View function to see pending Tokens on frontend.
    function pendingRewards(uint256 _pid, address _user) public view returns (uint256 value) {
        value = totalRewards(_pid, _user)
                    .add(userInfo[_pid][_user].rewardRemain)
                    .sub(userInfo[_pid][_user].rewardDebt);
    }

    function totalRewards(uint256 _pid, address _user) public view returns (uint256 value) {
```

```
        PoolInfo storage pool = poolInfo[_pid];
        uint256 accRewardPerShare = pool.accRewardPerShare;
        if (block.number > pool.lastRewardBlock && pool.totalAmount != 0) {
            uint256 poolReward = getBlocksReward(pool.lastRewardBlock, block.number)
                                    .mul(pool.allocPoint).div(totalAllocPoint);
            accRewardPerShare = accRewardPerShare.add(poolReward.mul(1e18).div(pool.totalAmount));
        }
        value = userInfo[_pid][_user].amount.mul(accRewardPerShare).div(1e18);
    }

    // Update reward variables for all pools. Be careful of gas spending!
    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    // Update reward variables of the given pool to be up-to-date.
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (pool.allocPoint == 0 || pool.totalAmount == 0) {
            pool.lastRewardBlock = block.number;
            return;
        }
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        uint256 poolReward = getBlocksReward(pool.lastRewardBlock, block.number)
                                .mul(pool.allocPoint).div(totalAllocPoint);
        if (poolReward > 0) {
            rewardToken.mint(address(this), poolReward);
            rewardToken.mint(devaddr, poolReward.mul(100).div(750));
            pool.accRewardPerShare = pool.accRewardPerShare.add(poolReward.mul(1e18).div(pool.totalAm
        }
        pool.lastRewardBlock = block.number;

        if(extendPool != address(0)) {
            IActionPools(extendPool).onAcionUpdate(_pid);
        }
    }

    // Deposit LP tokens to MasterChef for Token allocation.
    function deposit(uint256 _pid, uint256 _amount) external {
        require(!depositBlacklist[msg.sender], 'user in blacklist');
        updatePool(_pid);
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        if (user.amount > 0) {
            user.rewardRemain = pendingRewards(_pid, msg.sender);
        }
        uint256 amountOld = user.amount;
        if(_amount > 0) {
            IERC20Upgradeable(pool.lpToken).safeTransferFrom(address(msg.sender), address(this), _amo
            user.amount = user.amount.add(_amount);
            pool.totalAmount = pool.totalAmount.add(_amount);
        }
        user.rewardDebt = totalRewards(_pid, msg.sender);
        emit Deposit(msg.sender, _pid, _amount);

        if(extendPool != address(0)) {
            IActionPools(extendPool).onAcionIn(_pid, msg.sender, amountOld, user.amount);
        }
    }

    // Withdraw LP tokens from StarPool.
```

```
    function withdraw(uint256 _pid, uint256 _amount) external {
        updatePool(_pid);
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");
        user.rewardRemain = pendingRewards(_pid, msg.sender);
        uint256 amountOld = user.amount;
        if(_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            IERC20Upgradeable(pool.lpToken).safeTransfer(address(msg.sender), _amount);
        }
        user.rewardDebt = totalRewards(_pid, msg.sender);
        emit Withdraw(msg.sender, _pid, _amount);

        if(extendPool != address(0)) {
            IActionPools(extendPool).onAcionOut(_pid, msg.sender, amountOld, user.amount);
        }
    }

    function claim(uint256 _pid) public returns (uint256 value) {
        _claim(_pid, msg.sender);
    }

    function _claim(uint256 _pid, address _account) internal returns (uint256 value) {
        updatePool(_pid);
        value = pendingRewards(_pid, _account);
        if (value > 0) {
            userInfo[_pid][_account].rewardRemain = 0;
            if(rewardRestricted[_account] > 0) {
                value = value.sub(value.mul(rewardRestricted[_account]).div(1e9));
            }
            value = safeTokenTransfer(_account, value);
            userInfo[_pid][_account].rewardDebt = totalRewards(_pid, _account);
        }

        emit Claim(_account, _pid, value);

        if(extendPool != address(0)) {
            IActionPools(extendPool).onAcionClaim(_pid, _account);
        }
    }

    function claimFromBank(address _account, uint256[] memory _pidlist)
        external override returns (uint256 value) {
        require(bank==msg.sender, 'only call from bank');
        for (uint256 piid = 0; piid < _pidlist.length; ++piid) {
            value = value.add(_claim(_pidlist[piid], _account));
        }
    }

    function claimAll() external returns (uint256 value) {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            claim(pid);
        }
    }

    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) external {
        require(emergencyWithdrawEnabled[_pid], 'not allowed');
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
```

```
        user.rewardRemain = 0;
        pool.totalAmount = pool.totalAmount.sub(amount);
        IERC20Upgradeable(pool.lpToken).safeTransfer(address(msg.sender), amount);
        emit EmergencyWithdraw(msg.sender, _pid, amount);

        if(extendPool != address(0)) {
            IActionPools(extendPool).onAcionEmergency(_pid, msg.sender);
        }
    }

    // Safe Token transfer function, just in case if rounding error causes pool to not have enough To
    function safeTokenTransfer(address _to, uint256 _amount) internal returns (uint256 value) {
        uint256 balance = rewardToken.balanceOf(address(this));
        value = _amount > balance ? balance : _amount;
        if ( value > 0 ) {
            rewardToken.transfer(_to, value);
        }
    }

    // If the user transfers TH to contract, it will revert
    receive() external payable {
        revert();
    }
}
```

MultiSourceOracle::TransparentUpgradeableProxy.sol

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


pragma solidity >=0.6.0 <0.8.0;

            /**
 * @dev This abstract contract provides a fallback function that delegates all calls to an
 * instruction `delegatecall`. We refer to the second contract as the
 * be specified by overriding the virtual {_implementation} function.
 *
 * Additionally, delegation to the implementation can be triggered manually through
 * different contract through the {_delegate} function.
 *
 * The success and return data of the delegated call will b
 */
abstract contract Proxy {
                /**
   * @dev Delegates the current call to `implementation`.
   *
   * This function does not return to its internall call site, it will return directly to
   */
    function _delegate(address implementation) internal virtual {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)
```

```
            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
            // delegatecall returns 0 on error.
            case 0 { revert(0, returndatasize()) }
            default { return(0, returndatasize()) }
        }
    }

    /**
     * @dev This is a virtual function that should be override
     * and {_fallback} should delegate.
     */
    function _implementation() internal view virtual returns (address);

    /**
     * @dev Delegates the current call to the address return
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _fallback() internal virtual {
        _beforeFallback();
        _delegate(_implementation());
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_impleme
     * function in the contract matches the call data.
     */
    fallback () external payable virtual {
        _fallback();
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_impleme
     * is empty.
     */
    receive () external payable virtual {
        _fallback();
    }

    /**
     * @dev Hook that is called before falling back to the implementation. Can happen a
     * call, or as part of the Solidity `fallback` or `receive` functions.
     *
     * If overriden should call `super._beforeFallback()`.
     */
    function _beforeFallback() internal virtual {
    }
}


pragma solidity >=0.6.2 <0.8.0;

/**
 * @dev Collection of functions related to the address type
```

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

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

                /**
   * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
   * `recipient`, forwarding all available gas and reverting on errors.
   *
   * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
   * of certain opcodes, possibly making contracts go over the 2300 gas limit
   * imposed by `transfer`, making them unable to receive funds via
   * `transfer`. {sendValue} removes this limitation.
   *
   * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more
   *
   * IMPORTANT: because control is transferred to `recipient`, care must be
   * taken to not create reentrancy vulnerabilities. Consider using
   * {ReentrancyGuard} or the
   * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use- the -che
   */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

                /**
```

```
 * @dev Performs a Solidity function call using a low lev
 * plain`call` is an unsafe replacement for a function call:
 * function instead.
 *
 * If `target` reverts with a revert reason, it is bubbled up by this
 * function ( like regular Solidity function calls).
 *
 * Returns the raw returned data. To convert to the expe
 * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding
 *
 * Requirements:
 *
 * - `target` must be a contract.
 * - calling `target` with `data` must not revert.
 *
 * _Available since v3.1._
 */
  function functionCall(address target, bytes memory data) internal returns (bytes memory) {
    return functionCall(target, data, "Address: low-level call failed");
  }

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

                /**
 * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
 * but also transferring `value` wei to `target`.
 *
 * Requirements:
 *
 * - the calling contract must have an ETH balance of at
 * - the called Solidity function must be `payable`.
 *
 * _Available since v3.1._
 */
  function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
      return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
  }

                /**
 * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`],
 * with `errorMessage` as a fallback revert reason when `target` reverts.
 *
 * _Available since v3.1._
 */
  function functionCallWithValue(address target, bytes memory data, uint256 value, string memory er
```

```
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

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

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

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

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

    function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) pri
        if (success) {
```

```
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

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


pragma solidity >=0.6.0 <0.8.0;


            /**
 * @dev This contract implements an upgradeable proxy. It is upgradeable because cal
 * implementation address that can be changed. This address is stored in storage in the
 * https://eips.ethereum.org/EIPS/eip-1967[EIP1967], so that it doesn't
 * implementation behind the proxy.
 *
 * Upgradeability is only provided internally through {_upgradeTo}. For an externally up
 * {TransparentUpgradeableProxy}.
 */
contract UpgradeableProxy is Proxy {
                /**
   * @dev Initializes the upgradeable proxy with an initial
   *
   * If `_data` is nonempty, it's used as data in a delegate call to `_logic`. This
   * function call, and allows initializating the storage of the
   */
    constructor(address _logic, bytes memory _data) public payable {
        assert(_IMPLEMENTATION_SLOT == bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1
        _setImplementation(_logic);
        if(_data.length > 0) {
            Address.functionDelegateCall(_logic, _data);
        }
    }

                /**
   * @dev Emitted when the implementation is upgraded.
   */
    event Upgraded(address indexed implementation);

                /**
   * @dev Storage slot with the address of the current imp
   * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, ar
   * validated in the constructor.
   */
    bytes32 private constant _IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a
```

```
    /**
     * @dev Returns the current implementation address.
     */
    function _implementation() internal view virtual override returns (address impl) {
        bytes32 slot = _IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            impl := sload(slot)
        }
    }

    /**
     * @dev Upgrades the proxy to a new implementation.
     *
     * Emits an {Upgraded} event.
     */
    function _upgradeTo(address newImplementation) internal virtual {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

    /**
     * @dev Stores a new address in the EIP1967 implemer
     */
    function _setImplementation(address newImplementation) private {
        require(Address.isContract(newImplementation), "UpgradeableProxy: new implementation is not a

        bytes32 slot = _IMPLEMENTATION_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newImplementation)
        }
    }
}


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev This contract implements a proxy that is upgradeable by an
 *
 * To avoid https://medium.com/nomic-labs-blog/malicious-backdoors-in-ethereum-proxies-62629adf3357[proxy sel
 * clashing], which can potentially be used in an attack, this contract uses
 * https://blog.openzeppelin.com/ the -transparent-proxy-pattern/[transparent proxy patte
 * things that go hand in hand:
 *
 * 1. If any account other than the admin calls the proxy,
 * that call matches one of the admin functions exposed by the
 * 2. If the admin calls the proxy, it can access
 * implementation. If the admin tries to call a function on
 * "admin cannot fallback to proxy target".
 *
 * These properties mean that the admin account can only be used for admin actions
 * the admin, so it's best if it's a
 * to sudden errors when trying to call a function from the p
```

```
 *
 * Our recommendation is for the dedicated account to be an
 *           you should think of the `Pro
 */
contract TransparentUpgradeableProxy is UpgradeableProxy {
                /**
   * @dev Initializes an upgradeable proxy managed by `_admin`, backed by
   * optionally initialized with `_data` as explained in {UpgradeableProxy-constructor}.
   */
    constructor(address _logic, address admin_, bytes memory _data) public payable UpgradeableProxy(_
        assert(_ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
        _setAdmin(admin_);
    }

                /**
   * @dev Emitted when the admin account has changed.
   */
    event AdminChanged(address previousAdmin, address newAdmin);

                /**
   * @dev Storage slot with the admin of the contract.
   * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
   * validated in the constructor.
   */
    bytes32 private constant _ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a71785

                /**
   * @dev Modifier used internally that will delegate the
   */
    modifier ifAdmin() {
        if (msg.sender == _admin()) {
            _;
        } else {
            _fallback();
        }
    }

                /**
   * @dev Returns the current admin.
   *
   *           NOTE: Only the admin can call this function. See {Prox
   *
   * TIP: To get this value clients can read directly from the storage slot shown below (
   * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
   * `0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103`
   */
    function admin() external ifAdmin returns (address admin_) {
        admin_ = _admin();
    }

                /**
   * @dev Returns the current implementation.
   *
   *           NOTE: Only the admin can call this function. See {Prox
   *
```

```
 * TIP: To get this value clients can read directly from the storage slot shown below (
 * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
 * `0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`
 */
  function implementation() external ifAdmin returns (address implementation_) {
      implementation_ = _implementation();
  }

              /**
 * @dev Changes the admin of the proxy.
 *
 * Emits an {AdminChanged} event.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 */
  function changeAdmin(address newAdmin) external virtual ifAdmin {
      require(newAdmin != address(0), "TransparentUpgradeableProxy: new admin is the zero address")
      emit AdminChanged(_admin(), newAdmin);
      _setAdmin(newAdmin);
  }

              /**
 * @dev Upgrade the implementation of the proxy.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 */
  function upgradeTo(address newImplementation) external virtual ifAdmin {
      _upgradeTo(newImplementation);
  }

              /**
 * @dev Upgrade the implementation of the proxy, and t
 * by `data`, which should be an encoded function call. T
 * proxied contract.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 */
  function upgradeToAndCall(address newImplementation, bytes calldata data) external payable virtua
      _upgradeTo(newImplementation);
      Address.functionDelegateCall(newImplementation, data);
  }

              /**
 * @dev Returns the current admin.
 */
  function _admin() internal view virtual returns (address adm) {
      bytes32 slot = _ADMIN_SLOT;
      // solhint-disable-next-line no-inline-assembly
      assembly {
          adm := sload(slot)
      }
  }

              /**
 * @dev Stores a new address in the EIP1967 admin slo
 */
  function _setAdmin(address newAdmin) private {
```

```
        bytes32 slot = _ADMIN_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newAdmin)
        }
    }

                /**
   * @dev Makes sure the admin cannot access the fallba
   */
    function _beforeFallback() internal virtual override {
        require(msg.sender != _admin(), "TransparentUpgradeableProxy: admin cannot fallback to proxy
        super._beforeFallback();
    }
}
```

SafeBoxFox::TransparentUpgradeableProxy.sol

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


pragma solidity >=0.6.0 <0.8.0;

            /**
 * @dev This abstract contract provides a fallback function that delegates all calls to an
 * instruction `delegatecall`. We refer to the second contract as the
 * be specified by overriding the virtual {_implementation} function.
 *
 * Additionally, delegation to the implementation can be triggered manually through
 * different contract through the {_delegate} function.
 *
 * The success and return data of the delegated call will b
 */
abstract contract Proxy {
                /**
   * @dev Delegates the current call to `implementation`.
   *
   * This function does not return to its internall call site, it will return directly to
   */
    function _delegate(address implementation) internal virtual {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
            // delegatecall returns 0 on error.
            case 0 { revert(0, returndatasize()) }
```

```
            default { return(0, returndatasize()) }
        }
    }

    /**
     * @dev This is a virtual function that should be override
     * and {_fallback} should delegate.
     */
    function _implementation() internal view virtual returns (address);

    /**
     * @dev Delegates the current call to the address return
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _fallback() internal virtual {
        _beforeFallback();
        _delegate(_implementation());
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_impleme
     * function in the contract matches the call data.
     */
    fallback () external payable virtual {
        _fallback();
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_impleme
     * is empty.
     */
    receive () external payable virtual {
        _fallback();
    }

    /**
     * @dev Hook that is called before falling back to the implementation. Can happen a
     * call, or as part of the Solidity `fallback` or `receive` functions.
     *
     * If overriden should call `super._beforeFallback()`.
     */
    function _beforeFallback() internal virtual {
    }
}


pragma solidity >=0.6.2 <0.8.0;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
```

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

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

    /**
 * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
 * `recipient`, forwarding all available gas and reverting on errors.
 *
 * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
 * of certain opcodes, possibly making contracts go over the 2300 gas limit
 * imposed by `transfer`, making them unable to receive funds via
 * `transfer`. {sendValue} removes this limitation.
 *
 * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more
 *
 * IMPORTANT: because control is transferred to `recipient`, care must be
 * taken to not create reentrancy vulnerabilities. Consider using
 * {ReentrancyGuard} or the
 * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use- the -che
 */
function sendValue(address payable recipient, uint256 amount) internal {
    require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
 * @dev Performs a Solidity function call using a low lev
 * plain`call` is an unsafe replacement for a function call:
 * function instead.
 *
 * If `target` reverts with a revert reason, it is bubbled up by this
```

```
 * function (like regular Solidity function calls).
 *
 * Returns the raw returned data. To convert to the expe
 * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding
 *
 * Requirements:
 *
 * - `target` must be a contract.
 * - calling `target` with `data` must not revert.
 *
 * _Available since v3.1._
 */
  function functionCall(address target, bytes memory data) internal returns (bytes memory) {
    return functionCall(target, data, "Address: low-level call failed");
  }

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

                /**
 * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
 * but also transferring `value` wei to `target`.
 *
 * Requirements:
 *
 * - the calling contract must have an ETH balance of at
 * - the called Solidity function must be `payable`.
 *
 * _Available since v3.1._
 */
  function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
      return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
  }

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

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

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

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

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

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

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

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


pragma solidity >=0.6.0 <0.8.0;


            /**
 * @dev This contract implements an upgradeable proxy. It is upgradeable because cal
 * implementation address that can be changed. This address is stored in storage in the
 * https://eips.ethereum.org/EIPS/eip-1967[EIP1967], so that it doesn't
 * implementation behind the proxy.
 *
 * Upgradeability is only provided internally through {_upgradeTo}. For an externally up
 * {TransparentUpgradeableProxy}.
 */
contract UpgradeableProxy is Proxy {
            /**
   * @dev Initializes the upgradeable proxy with an initial
   *
   * If `_data` is nonempty, it's used as data in a delegate call to `_logic`. This
   * function call, and allows initializating the storage of the
   */
    constructor(address _logic, bytes memory _data) public payable {
        assert(_IMPLEMENTATION_SLOT == bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1
        _setImplementation(_logic);
        if(_data.length > 0) {
            Address.functionDelegateCall(_logic, _data);
        }
    }

            /**
   * @dev Emitted when the implementation is upgraded.
   */
    event Upgraded(address indexed implementation);

            /**
   * @dev Storage slot with the address of the current imp
   * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, ar
   * validated in the constructor.
   */
    bytes32 private constant _IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a

            /**
   * @dev Returns the current implementation address.
   */
    function _implementation() internal view virtual override returns (address impl) {
        bytes32 slot = _IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
```

```
            impl := sload(slot)
        }
    }

    /**
     * @dev Upgrades the proxy to a new implementation.
     *
     * Emits an {Upgraded} event.
     */
    function _upgradeTo(address newImplementation) internal virtual {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

    /**
     * @dev Stores a new address in the EIP1967 implemer
     */
    function _setImplementation(address newImplementation) private {
        require(Address.isContract(newImplementation), "UpgradeableProxy: new implementation is not a

        bytes32 slot = _IMPLEMENTATION_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newImplementation)
        }
    }
}


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev This contract implements a proxy that is upgradeable by an
 *
 * To avoid https://medium.com/nomic-labs-blog/malicious-backdoors-in-ethereum-proxies-62629adf3357[proxy sel
 * clashing], which can potentially be used in an attack, this contract uses
 * https://blog.openzeppelin.com/ the -transparent-proxy-pattern/[transparent proxy patte
 * things that go hand in hand:
 *
 * 1. If any account other than the admin calls the proxy,
 * that call matches one of the admin functions exposed by the
 * 2. If the admin calls the proxy, it can access
 * implementation. If the admin tries to call a function on
 * "admin cannot fallback to proxy target".
 *
 * These properties mean that the admin account can only be used for admin actions
 * the admin, so it's best if it's a
 * to sudden errors when trying to call a function from the p
 *
 * Our recommendation is for the dedicated account to be an
 * you should think of the `Pro
 */
contract TransparentUpgradeableProxy is UpgradeableProxy {
    /**
```

```
 * @dev Initializes an upgradeable proxy managed by `_admin`, backed by
 * optionally initialized with `_data` as explained in {UpgradeableProxy-constructor}.
 */
  constructor(address _logic, address admin_, bytes memory _data) public payable UpgradeableProxy(_
      assert(_ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
      _setAdmin(admin_);
  }

              /**
 * @dev Emitted when the admin account has changed.
 */
  event AdminChanged(address previousAdmin, address newAdmin);

              /**
 * @dev Storage slot with the admin of the contract.
 * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
 * validated in the constructor.
 */
  bytes32 private constant _ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a71785

              /**
 * @dev Modifier used internally that will delegate the
 */
  modifier ifAdmin() {
      if (msg.sender == _admin()) {
          _;
      } else {
          _fallback();
      }
  }

              /**
 * @dev Returns the current admin.
 *
 *              NOTE: Only the admin can call this function. See {Prox
 *
 * TIP: To get this value clients can read directly from the storage slot shown below (
 * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
 * `0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103`
 */
  function admin() external ifAdmin returns (address admin_) {
      admin_ = _admin();
  }

              /**
 * @dev Returns the current implementation.
 *
 *              NOTE: Only the admin can call this function. See {Prox
 *
 * TIP: To get this value clients can read directly from the storage slot shown below (
 * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
 * `0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`
 */
  function implementation() external ifAdmin returns (address implementation_) {
      implementation_ = _implementation();
  }
```

```
    /**
     * @dev Changes the admin of the proxy.
     *
     * Emits an {AdminChanged} event.
     *
     * NOTE: Only the admin can call this function. See {Prox
     */
    function changeAdmin(address newAdmin) external virtual ifAdmin {
        require(newAdmin != address(0), "TransparentUpgradeableProxy: new admin is the zero address")
        emit AdminChanged(_admin(), newAdmin);
        _setAdmin(newAdmin);
    }

    /**
     * @dev Upgrade the implementation of the proxy.
     *
     * NOTE: Only the admin can call this function. See {Prox
     */
    function upgradeTo(address newImplementation) external virtual ifAdmin {
        _upgradeTo(newImplementation);
    }

    /**
     * @dev Upgrade the implementation of the proxy, and t
     * by `data`, which should be an encoded function call. T
     * proxied contract.
     *
     * NOTE: Only the admin can call this function. See {Prox
     */
    function upgradeToAndCall(address newImplementation, bytes calldata data) external payable virtua
        _upgradeTo(newImplementation);
        Address.functionDelegateCall(newImplementation, data);
    }

    /**
     * @dev Returns the current admin.
     */
    function _admin() internal view virtual returns (address adm) {
        bytes32 slot = _ADMIN_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            adm := sload(slot)
        }
    }

    /**
     * @dev Stores a new address in the EIP1967 admin slo
     */
    function _setAdmin(address newAdmin) private {
        bytes32 slot = _ADMIN_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newAdmin)
        }
    }
```

```
                /**
   * @dev Makes sure                the          admin cannot access          the          fallba
   */
    function _beforeFallback() internal virtual override {
        require(msg.sender != _admin(), "TransparentUpgradeableProxy: admin cannot fallback to proxy
        super._beforeFallback();
    }
}
```

MultiSourceOracle.sol

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMathUpgradeable {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
```

```
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
```

```
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
```

```
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}


pragma solidity >=0.6.2 <0.8.0;

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

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

```
    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
```

```
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

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

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

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

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

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

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

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

```
// solhint-disable-next-line compiler-version
pragma solidity >=0.4.24 <0.8.0;


/**
 * @dev This is a base contract to aid in writing upgradeable contracts, or any kind of contract that
 * behind a proxy. Since a proxied contract can't have a constructor, it's common to move constructor
 * external initializer function, usually called `initialize`. It then becomes necessary to protect t
 * function so it can only be called once. The {initializer} modifier provided by this contract will
 *
 * TIP: To avoid leaving the proxy in an uninitialized state, the initializer function should be call
 * possible by providing the encoded function call as the `_data` argument to {UpgradeableProxy-const
 *
 * CAUTION: When used with inheritance, manual care must be taken to not invoke a parent initializer
 * that all initializers are idempotent. This is not verified automatically as constructors are by So
 */
abstract contract Initializable {

    /**
     * @dev Indicates that the contract has been initialized.
     */
    bool private _initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private _initializing;

    /**
     * @dev Modifier to protect an initializer function from being invoked twice.
     */
    modifier initializer() {
        require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is alrea

        bool isTopLevelCall = !_initializing;
        if (isTopLevelCall) {
            _initializing = true;
            _initialized = true;
        }

        _;

        if (isTopLevelCall) {
            _initializing = false;
        }
    }

    /// @dev Returns true if and only if the function is running in the constructor
    function _isConstructor() private view returns (bool) {
        return !AddressUpgradeable.isContract(address(this));
    }
}


pragma solidity >=0.6.0 <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 GSN 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 ContextUpgradeable is Initializable {
    function __Context_init() internal initializer {
        __Context_init_unchained();
    }

    function __Context_init_unchained() internal initializer {
    }
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
    uint256[50] private __gap;
}


pragma solidity >=0.6.0 <0.8.0;

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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    function __Ownable_init() internal initializer {
        __Context_init_unchained();
        __Ownable_init_unchained();
    }

    function __Ownable_init_unchained() internal initializer {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }
```

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
    uint256[49] private __gap;
}

pragma solidity 0.6.12;

interface ICTokenInterface {

  function isCToken() external view returns (bool);

  // function decimals() external returns (uint8);

  function underlying() external view returns (address);

  // function mint(uint mintAmount) external returns (uint);

  // function redeem(uint redeemTokens) external returns (uint);

  // function balanceOf(address user) external view returns (uint);

  // function borrowBalanceCurrent(address account) external returns (uint);

  // function borrowBalanceStored(address account) external view returns (uint);

  // function borrow(uint borrowAmount) external returns (uint);

  // function repayBorrow(uint repayAmount) external returns (uint);
  // function transfer(address dst, uint amount) external returns (bool);
  // function transferFrom(address src, address dst, uint amount) external returns (bool);
  // function approve(address spender, uint amount) external returns (bool);
  // function allowance(address owner, address spender) external view returns (uint);
  // function balanceOf(address owner) external view returns (uint);
  function balanceOfUnderlying(address owner) external view returns (uint);
  function getAccountSnapshot(address account) external view returns (uint, uint, uint, uint);
  function borrowRatePerBlock() external view returns (uint);
  function supplyRatePerBlock() external view returns (uint);
  function totalBorrowsCurrent() external returns (uint);
  function borrowBalanceCurrent(address account) external returns (uint);
  function borrowBalanceStored(address account) external view returns (uint);
  function exchangeRateCurrent() external returns (uint);
  function exchangeRateStored() external view returns (uint);
  function getCash() external view returns (uint);
  function accrueInterest() external returns (uint);
  function seize(address liquidator, address borrower, uint seizeTokens) external returns (uint);
}
```

```
pragma solidity 0.6.12;

interface ITokenOracle {
    function getPrice(address _token) external view returns (int);
}


pragma solidity 0.6.12;



contract MultiSourceOracle is OwnableUpgradeable, ITokenOracle {
    using SafeMathUpgradeable for uint256;

    struct PriceData {
        uint price;
        uint lastUpdate;
    }

    bool public constant isPriceOracle = true;
    mapping(address => bool) public opers;
    mapping(address => address) public priceFeeds;
    mapping(address => PriceData) public store;

    event PriceUpdate(address indexed _token, uint price);
    event PriceFeed(address indexed _token, address _feed);

    constructor() public {
    }

    function initialize() public initializer {
        __Ownable_init();
        opers[msg.sender] = true;
    }

    function setPriceOperator(address _oper, bool _enable) public onlyOwner {
        opers[_oper] = _enable;
    }

    function setFeeds(address[] memory _tokens, address[] memory _feeds) public onlyOwner {
        require(_tokens.length == _feeds.length, 'bad token length');
        for (uint idx = 0; idx < _tokens.length; idx++) {
            address token0 = _tokens[idx];
            address feed = _feeds[idx];
            emit PriceFeed(token0, feed);
            require(ITokenOracle(feed).getPrice(token0) > 0, 'token no price');
            priceFeeds[token0] = feed;
        }
    }

    /// @dev Set the prices of the token token pairs. Must be called by the oper.
    // price (scaled by 1e18).
    function setPrices(
        address[] memory tokens,
        uint[] memory prices
    ) external {
        require(opers[msg.sender], 'only oper');
        require(tokens.length == prices.length, 'bad token length');
        for (uint idx = 0; idx < tokens.length; idx++) {
            address token0 = tokens[idx];
            uint price = prices[idx];
            store[token0] = PriceData({price: price, lastUpdate: now});
            emit PriceUpdate(token0, price);
        }
```

```
    }

    function getPrice(address _token) public override view returns (int) {
        address feed = priceFeeds[_token];
        if(feed != address(0)) {
            return ITokenOracle(feed).getPrice(_token);
        }
        require(store[_token].price >= 0, 'price to lower');
        return int(store[_token].price);
    }


    /**
      * @notice Get the underlying price of a cToken asset
      * @param cToken The cToken to get the underlying price of
      * @return The underlying asset price mantissa (scaled by 1e18).
      *  Zero means the price is unavailable.
      */
    function getUnderlyingPrice(address cToken) external view returns (uint) {
        address token = ICTokenInterface(cToken).underlying();
        int price = getPrice(token);
        require(price >= 0, 'price to lower');
        return uint(price).mul(uint(1e18).div(1e8));
    }
}
```

StrategyV2Pair::TransparentUpgradeableProxy.sol

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

pragma solidity >=0.6.0 <0.8.0;

            /**
 * @dev This abstract contract provides a fallback function that delegates all calls to an
 * instruction `delegatecall`. We refer to the second contract as the
 * be specified by overriding the virtual {_implementation} function.
 *
 * Additionally, delegation to the implementation can be triggered manually through
 * different contract through the {_delegate} function.
 *
 * The success and return data of the delegated call will b
 */
abstract contract Proxy {
                /**
   * @dev Delegates the current call to `implementation`.
   *
   * This function does not return to its internall call site, it will return directly to
   */
    function _delegate(address implementation) internal virtual {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
```

```
            let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
            // delegatecall returns 0 on error.
            case 0 { revert(0, returndatasize()) }
            default { return(0, returndatasize()) }
        }
    }

    /**
     * @dev This is a virtual function that should be override
     * and {_fallback} should delegate.
     */
    function _implementation() internal view virtual returns (address);

    /**
     * @dev Delegates the current call to the address return
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _fallback() internal virtual {
        _beforeFallback();
        _delegate(_implementation());
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_impleme
     * function in the contract matches the call data.
     */
    fallback () external payable virtual {
        _fallback();
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_impleme
     * is empty.
     */
    receive () external payable virtual {
        _fallback();
    }

    /**
     * @dev Hook that is called before falling back to the implementation. Can happen a
     * call, or as part of the Solidity `fallback` or `receive` functions.
     *
     * If overriden should call `super._beforeFallback()`.
     */
    function _beforeFallback() internal virtual {
    }
}


pragma solidity >=0.6.2 <0.8.0;

            /**
```

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

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use- the -che
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

```
    /**
     * @dev Performs a Solidity function call using a low lev
     * plain`call` is an unsafe replacement for a function call:
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function ( like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expe
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`],
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
```

```
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory er
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

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

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

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

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

    function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) pri
```

```
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

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


pragma solidity >=0.6.0 <0.8.0;


            /**
 * @dev This contract implements an upgradeable proxy. It is upgradeable because cal
 * implementation address that can be changed. This address is stored in storage in the
 * https://eips.ethereum.org/EIPS/eip-1967[EIP1967], so that it doesn't
 * implementation behind the proxy.
 *
 * Upgradeability is only provided internally through {_upgradeTo}. For an externally up
 * {TransparentUpgradeableProxy}.
 */
contract UpgradeableProxy is Proxy {
                /**
   * @dev Initializes the upgradeable proxy with an initial
   *
   * If `_data` is nonempty, it's used as data in a delegate call to `_logic`. This
   * function call, and allows initializating the storage of the
   */
    constructor(address _logic, bytes memory _data) public payable {
        assert(_IMPLEMENTATION_SLOT == bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1
        _setImplementation(_logic);
        if(_data.length > 0) {
            Address.functionDelegateCall(_logic, _data);
        }
    }

                /**
   * @dev Emitted when the implementation is upgraded.
   */
    event Upgraded(address indexed implementation);

                /**
   * @dev Storage slot with the address of the current imp
   * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, ar
   * validated in the constructor.
   */
    bytes32 private constant _IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a
```

```
    /**
     * @dev Returns the current implementation address.
     */
    function _implementation() internal view virtual override returns (address impl) {
        bytes32 slot = _IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            impl := sload(slot)
        }
    }

    /**
     * @dev Upgrades the proxy to a new implementation.
     *
     * Emits an {Upgraded} event.
     */
    function _upgradeTo(address newImplementation) internal virtual {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

    /**
     * @dev Stores a new address in the EIP1967 implemer
     */
    function _setImplementation(address newImplementation) private {
        require(Address.isContract(newImplementation), "UpgradeableProxy: new implementation is not a

        bytes32 slot = _IMPLEMENTATION_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newImplementation)
        }
    }
}


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev This contract implements a proxy that is upgradeable by an
 *
 * To avoid https://medium.com/nomic-labs-blog/malicious-backdoors-in-ethereum-proxies-62629adf3357[proxy sel
 * clashing], which can potentially be used in an attack, this contract uses
 * https://blog.openzeppelin.com/ the -transparent-proxy-pattern/[transparent proxy patte
 * things that go hand in hand:
 *
 * 1. If any account other than the admin calls the proxy,
 * that call matches one of the admin functions exposed by the
 * 2. If the admin calls the proxy, it can access
 * implementation. If the admin tries to call a function on
 * "admin cannot fallback to proxy target".
 *
 * These properties mean that the admin account can only be used for admin actions
 * the admin, so it's best if it's a
 * to sudden errors when trying to call a function from the p
```

```
 *
 * Our recommendation is for the dedicated account to be an
 *            you should think of the `Pro
 */
contract TransparentUpgradeableProxy is UpgradeableProxy {
                /**
   * @dev Initializes an upgradeable proxy managed by `_admin`, backed by
   * optionally initialized with `_data` as explained in {UpgradeableProxy-constructor}.
   */
    constructor(address _logic, address admin_, bytes memory _data) public payable UpgradeableProxy(_
        assert(_ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
        _setAdmin(admin_);
    }

                /**
   * @dev Emitted when the admin account has changed.
   */
    event AdminChanged(address previousAdmin, address newAdmin);

                /**
   * @dev Storage slot with the admin of the contract.
   * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
   * validated in the constructor.
   */
    bytes32 private constant _ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a71785

                /**
   * @dev Modifier used internally that will delegate the
   */
    modifier ifAdmin() {
        if (msg.sender == _admin()) {
            _;
        } else {
            _fallback();
        }
    }

                /**
   * @dev Returns the current admin.
   *
   *            NOTE: Only the admin can call this function. See {Prox
   *
   * TIP: To get this value clients can read directly from the storage slot shown below (
   * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
   * `0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103`
   */
    function admin() external ifAdmin returns (address admin_) {
        admin_ = _admin();
    }

                /**
   * @dev Returns the current implementation.
   *
   *            NOTE: Only the admin can call this function. See {Prox
   *
```

```
 * TIP: To get this value clients can read directly from the storage slot shown below (
 * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
 * `0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`
 */
  function implementation() external ifAdmin returns (address implementation_) {
      implementation_ = _implementation();
  }

                /**
 * @dev Changes the admin of the proxy.
 *
 * Emits an {AdminChanged} event.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 */
  function changeAdmin(address newAdmin) external virtual ifAdmin {
      require(newAdmin != address(0), "TransparentUpgradeableProxy: new admin is the zero address")
      emit AdminChanged(_admin(), newAdmin);
      _setAdmin(newAdmin);
  }

                /**
 * @dev Upgrade the implementation of the proxy.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 */
  function upgradeTo(address newImplementation) external virtual ifAdmin {
      _upgradeTo(newImplementation);
  }

                /**
 * @dev Upgrade the implementation of the proxy, and t
 * by `data`, which should be an encoded function call. T
 * proxied contract.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 */
  function upgradeToAndCall(address newImplementation, bytes calldata data) external payable virtua
      _upgradeTo(newImplementation);
      Address.functionDelegateCall(newImplementation, data);
  }

                /**
 * @dev Returns the current admin.
 */
  function _admin() internal view virtual returns (address adm) {
      bytes32 slot = _ADMIN_SLOT;
      // solhint-disable-next-line no-inline-assembly
      assembly {
          adm := sload(slot)
      }
  }

                /**
 * @dev Stores a new address in the EIP1967 admin slo
 */
  function _setAdmin(address newAdmin) private {
```

```
        bytes32 slot = _ADMIN_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newAdmin)
        }
    }

    /**
     * @dev Makes sure the admin cannot access the fallba
     */
    function _beforeFallback() internal virtual override {
        require(msg.sender != _admin(), "TransparentUpgradeableProxy: admin cannot fallback to proxy
        super._beforeFallback();
    }
}
```

SafeBoxFox.sol

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20Upgradeable {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
```

```
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMathUpgradeable {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
```

```
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
```

```
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
```

```
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}


pragma solidity >=0.6.2 <0.8.0;

/**
 * @dev Collection of functions related to the address type
 */
library AddressUpgradeable {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
```

```
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }
```

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

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

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

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

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

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

    function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) pri
        if (success) {
```

```
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

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


pragma solidity >=0.6.0 <0.8.0;


/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20Upgradeable {
    using SafeMathUpgradeable for uint256;
    using AddressUpgradeable for address;

    function safeTransfer(IERC20Upgradeable token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20Upgradeable token, address from, address to, uint256 value) inter
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20Upgradeable token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }
```

```
    function safeDecreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20Upgradeable token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}


// solhint-disable-next-line compiler-version
pragma solidity >=0.4.24 <0.8.0;


/**
 * @dev This is a base contract to aid in writing upgradeable contracts, or any kind of contract that
 * behind a proxy. Since a proxied contract can't have a constructor, it's common to move constructor
 * external initializer function, usually called `initialize`. It then becomes necessary to protect t
 * function so it can only be called once. The {initializer} modifier provided by this contract will
 *
 * TIP: To avoid leaving the proxy in an uninitialized state, the initializer function should be call
 * possible by providing the encoded function call as the `_data` argument to {UpgradeableProxy-const
 *
 * CAUTION: When used with inheritance, manual care must be taken to not invoke a parent initializer
 * that all initializers are idempotent. This is not verified automatically as constructors are by So
 */
abstract contract Initializable {

    /**
     * @dev Indicates that the contract has been initialized.
     */
    bool private _initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private _initializing;

    /**
     * @dev Modifier to protect an initializer function from being invoked twice.
     */
    modifier initializer() {
        require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is alrea

        bool isTopLevelCall = !_initializing;
        if (isTopLevelCall) {
            _initializing = true;
            _initialized = true;
        }

        _;
```

```
        if (isTopLevelCall) {
            _initializing = false;
        }
    }

    /// @dev Returns true if and only if the function is running in the constructor
    function _isConstructor() private view returns (bool) {
        return !AddressUpgradeable.isContract(address(this));
    }
}


pragma solidity >=0.6.0 <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 GSN 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 ContextUpgradeable is Initializable {
    function __Context_init() internal initializer {
        __Context_init_unchained();
    }

    function __Context_init_unchained() internal initializer {
    }
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
    uint256[50] private __gap;
}


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
```

```
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20Upgradeable is Initializable, ContextUpgradeable, IERC20Upgradeable {
    using SafeMathUpgradeable for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
     * a default value of 18.
     *
     * To select a different value for {decimals}, use {_setupDecimals}.
     *
     * All three of these values are immutable: they can only be set once during
     * construction.
     */
    function __ERC20_init(string memory name_, string memory symbol_) internal initializer {
        __Context_init_unchained();
        __ERC20_init_unchained(name_, symbol_);
    }

    function __ERC20_init_unchained(string memory name_, string memory symbol_) internal initializer
        _name = name_;
        _symbol = symbol_;
        _decimals = 18;
    }

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

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

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
```

```
    function decimals() public view virtual returns (uint8) {
        return _decimals;
    }

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

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

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

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

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
```

```
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC2
        return true;
    }

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

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
```

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

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

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

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

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

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

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

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

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
```

```
    function _setupDecimals(uint8 decimals_) internal virtual {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks]
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
    uint256[44] private __gap;
}


pragma solidity >=0.6.0 <0.8.0;

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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    function __Ownable_init() internal initializer {
        __Context_init_unchained();
        __Ownable_init_unchained();
    }

    function __Ownable_init_unchained() internal initializer {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

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

```
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
    uint256[49] private __gap;
}


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Contract module that helps prevent reentrant calls to a function.
 *
 * Inheriting from `ReentrancyGuard` will make the {nonReentrant} modifier
 * available, which can be applied to functions to make sure there are no nested
 * (reentrant) calls to them.
 *
 * Note that because there is a single `nonReentrant` guard, functions marked as
 * `nonReentrant` may not call one another. This can be worked around by making
 * those functions `private`, and then adding `external` `nonReentrant` entry
 * points to them.
 *
 * TIP: If you would like to learn more about reentrancy and alternative ways
 * to protect against it, check out our blog post
 * https://blog.openzeppelin.com/reentrancy-after-istanbul/[Reentrancy After Istanbul].
 */
abstract contract ReentrancyGuardUpgradeable is Initializable {
    // Booleans are more expensive than uint256 or any type that takes up a full
    // word because each write operation emits an extra SLOAD to first read the
    // slot's contents, replace the bits taken up by the boolean, and then write
    // back. This is the compiler's defense against contract upgrades and
    // pointer aliasing, and it cannot be disabled.

    // The values being non-zero value makes deployment a bit more expensive,
    // but in exchange the refund on every call to nonReentrant will be lower in
    // amount. Since refunds are capped to a percentage of the total
    // transaction's gas, it is best to keep them low in cases like this one, to
    // increase the likelihood of the full refund coming into effect.
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    function __ReentrancyGuard_init() internal initializer {
```

```
        __ReentrancyGuard_init_unchained();
    }

    function __ReentrancyGuard_init_unchained() internal initializer {
        _status = _NOT_ENTERED;
    }

    /**
     * @dev Prevents a contract from calling itself, directly or indirectly.
     * Calling a `nonReentrant` function from another `nonReentrant`
     * function is not supported. It is possible to prevent this from happening
     * by making the `nonReentrant` function external, and make it call a
     * `private` function that does the actual work.
     */
    modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _status = _ENTERED;

        _;

        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
    uint256[49] private __gap;
}


pragma solidity 0.6.12;

interface ISafeBox {

    function bank() external view returns(address);

    function token() external view returns(address);

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

    function supplyRatePerBlock() external view returns (uint256);
    function borrowRatePerBlock() external view returns (uint256);

    function getBorrowInfo(uint256 _bid) external view
            returns (address owner, uint256 amount, address strategy, uint256 pid);
    function getBorrowId(address _strategy, uint256 _pid, address _account) external view returns (ui
    function getBorrowId(address _strategy, uint256 _pid, address _account, bool _add) external retur
    function getDepositTotal() external view returns (uint256);
    function getBorrowTotal() external view returns (uint256);
    // function getBorrowAmount(address _account) external view returns (uint256 value);
    function getBaseTokenPerLPToken() external view returns (uint256);

    function deposit(uint256 _value) external;
    function withdraw(uint256 _value) external;

    function emergencyWithdraw() external;
    function emergencyRepay(uint256 _bid) external;

    function borrowInfoLength() external view returns (uint256);

    function borrow(uint256 _bid, uint256 _value, address _to) external;
    function repay(uint256 _bid, uint256 _value) external;
    function claim(uint256 _tTokenAmount) external;

    function update() external;
```

```
    function mintDonate(uint256 _value) external;

    function pendingSupplyAmount(address _account) external view returns (uint256 value);
    function pendingBorrowAmount(uint256 _bid) external view returns (uint256 value);
    function pendingBorrowRewards(uint256 _bid) external view returns (uint256 value);
}

pragma solidity 0.6.12;

interface ICompActionTrigger {
    function getCATPoolInfo(uint256 _pid) external view
        returns (address lpToken, uint256 allocRate, uint256 totalPoints, uint256 totalAmount);
    function getCATUserAmount(uint256 _pid, address _account) external view
        returns (uint256 points);
}

pragma solidity 0.6.12;


interface IActionPools {

    function getPoolInfo(uint256 _pid) external view
        returns (address callFrom, uint256 callId, address rewardToken);
    function mintRewards(uint256 _callId) external;
    function getPoolIndex(address _callFrom, uint256 _callId) external view returns (uint256[] memory

    function onAcionIn(uint256 _callId, address _account, uint256 _fromAmount, uint256 _toAmount) ext
    function onAcionOut(uint256 _callId, address _account, uint256 _fromAmount, uint256 _toAmount) ex
    function onAcionClaim(uint256 _callId, address _account) external;
    function onAcionEmergency(uint256 _callId, address _account) external;
    function onAcionUpdate(uint256 _callId) external;
}
pragma solidity 0.6.12;


library TenMath {
  using SafeMathUpgradeable for uint256;

  function min(uint256 v1, uint256 v2) public pure returns (uint256 vr) {
    vr = v1 > v2 ? v2 : v1;
  }

  function safeSub(uint256 v1, uint256 v2) internal pure returns (uint256 vr) {
    vr = v1 > v2 ? v1.sub(v2) : 0;
  }

  // implementation from https://github.com/Uniswap/uniswap-lib/commit/99f3f28770640ba1bb1ff460ac7c52
  // original implementation: https://github.com/abdk-consulting/abdk-libraries-solidity/blob/master/
  function sqrt(uint256 x) internal pure returns (uint256) {
    if (x == 0) return 0;
    uint256 xx = x;
    uint256 r = 1;

    if (xx >= 0x100000000000000000000000000000000) {
      xx >>= 128;
      r <<= 64;
    }

    if (xx >= 0x10000000000000000) {
      xx >>= 64;
      r <<= 32;
    }
    if (xx >= 0x100000000) {
      xx >>= 32;
      r <<= 16;
    }
```

```
    if (xx >= 0x10000) {
      xx >>= 16;
      r <<= 8;
    }
    if (xx >= 0x100) {
      xx >>= 8;
      r <<= 4;
    }
    if (xx >= 0x10) {
      xx >>= 4;
      r <<= 2;
    }
    if (xx >= 0x8) {
      r <<= 1;
    }

    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1; // Seven iterations should be enough
    uint256 r1 = x / r;
    return (r < r1 ? r : r1);
  }
}


pragma solidity 0.6.12;

interface IErc20Interface {

    /*** User Interface ***/
    function underlying() external view returns (address);

    function mint(uint mintAmount) external returns (uint);  //
    function redeem(uint redeemTokens) external returns (uint);
    function redeemUnderlying(uint redeemAmount) external returns (uint);
    function borrow(uint borrowAmount) external returns (uint);
    function repayBorrow(uint repayAmount) external returns (uint);
    // function repayBorrowBehalf(address borrower, uint repayAmount) external returns (uint);
    // function liquidateBorrow(address borrower, uint repayAmount, CTokenInterface cTokenCollateral)

}
pragma solidity 0.6.12;

interface ICTokenInterface {

  function isCToken() external view returns (bool);

  // function decimals() external returns (uint8);

  function underlying() external view returns (address);

  // function mint(uint mintAmount) external returns (uint);

  // function redeem(uint redeemTokens) external returns (uint);

  // function balanceOf(address user) external view returns (uint);

  // function borrowBalanceCurrent(address account) external returns (uint);

  // function borrowBalanceStored(address account) external view returns (uint);

  // function borrow(uint borrowAmount) external returns (uint);
```

```
  // function repayBorrow(uint repayAmount) external returns (uint);
  // function transfer(address dst, uint amount) external returns (bool);
  // function transferFrom(address src, address dst, uint amount) external returns (bool);
  // function approve(address spender, uint amount) external returns (bool);
  // function allowance(address owner, address spender) external view returns (uint);
  // function balanceOf(address owner) external view returns (uint);
  function balanceOfUnderlying(address owner) external view returns (uint);
  function getAccountSnapshot(address account) external view returns (uint, uint, uint, uint);
  function borrowRatePerBlock() external view returns (uint);
  function supplyRatePerBlock() external view returns (uint);
  function totalBorrowsCurrent() external returns (uint);
  function borrowBalanceCurrent(address account) external returns (uint);
  function borrowBalanceStored(address account) external view returns (uint);
  function exchangeRateCurrent() external returns (uint);
  function exchangeRateStored() external view returns (uint);
  function getCash() external view returns (uint);
  function accrueInterest() external returns (uint);
  function seize(address liquidator, address borrower, uint seizeTokens) external returns (uint);
}


pragma solidity 0.6.12;



// Safebox vault, deposit, withdrawal, borrowing, repayment
contract SafeBoxFoxCTokenImpl is ERC20Upgradeable {
    using SafeMathUpgradeable for uint256;
    using SafeERC20Upgradeable for IERC20Upgradeable;

    IErc20Interface public eToken;
    ICTokenInterface public cToken;

    function __SafeBoxFoxCTokenImpl__init(address _cToken) public initializer {
        __ERC20_init(string(abi.encodePacked("d ", ERC20Upgradeable(IErc20Interface(_cToken).underlyi
            string(abi.encodePacked("d", ERC20Upgradeable(IErc20Interface(_cToken).underlying()).symb
        _setupDecimals(ERC20Upgradeable(_cToken).decimals());
        eToken = IErc20Interface(_cToken);
        cToken = ICTokenInterface(_cToken);
        require(cToken.isCToken(), 'not ctoken address');
        IERC20Upgradeable(baseToken()).approve(_cToken, uint256(-1));
    }

    function baseToken() public virtual view returns (address) {
        return eToken.underlying();
    }

    function ctokenSupplyRatePerBlock() public virtual view returns (uint256) {
        return cToken.supplyRatePerBlock();
    }

    function ctokenBorrowRatePerBlock() public virtual view returns (uint256) {
        return cToken.borrowRatePerBlock();
    }

    function call_balanceOf(address _token, address _account) public virtual view returns (uint256 ba
        balance = IERC20Upgradeable(_token).balanceOf(_account);
    }

    function call_balanceOfCToken_this() public virtual view returns (uint256 balance) {
        balance = call_balanceOf(address(cToken), address(this));
    }

    function call_balanceOfBaseToken_this() public virtual returns (uint256) {
        return call_balanceOfCToken_this().mul(cToken.exchangeRateCurrent()).div(1e18);
```

```
    }

    function call_borrowBalanceCurrent_this() public virtual returns (uint256) {
        return cToken.borrowBalanceCurrent(address(this));
    }

    function getBaseTokenPerCToken() public virtual view returns (uint256) {
        return cToken.exchangeRateStored();
    }

    // deposit
    function ctokenDeposit(uint256 _value) internal virtual returns (uint256 lpAmount) {
        uint256 cBalanceBefore = call_balanceOf(address(cToken), address(this));
        require(eToken.mint(uint256(_value)) == 0, 'deposit token error');
        uint256 cBalanceAfter = call_balanceOf(address(cToken), address(this));
        lpAmount = cBalanceAfter.sub(cBalanceBefore);
    }

    function ctokenWithdraw(uint256 _lpAmount) internal virtual returns (uint256 value) {
        uint256 cBalanceBefore = call_balanceOf(baseToken(), address(this));
        require(eToken.redeem(_lpAmount) == 0, 'withdraw supply ctoken error');
        uint256 cBalanceAfter = call_balanceOf(baseToken(), address(this));
        value = cBalanceAfter.sub(cBalanceBefore);
    }

    function ctokenClaim(uint256 _lpAmount) internal virtual returns (uint256 value) {
        value = ctokenWithdraw(_lpAmount);
    }

    function ctokenBorrow(uint256 _value) internal virtual returns (uint256 value) {
        uint256 cBalanceBefore = call_balanceOf(baseToken(), address(this));
        require(eToken.borrow(_value) == 0, 'borrow ubalance error');
        uint256 cBalanceAfter = call_balanceOf(baseToken(), address(this));
        value = cBalanceAfter.sub(cBalanceBefore);
    }

    function ctokenRepayBorrow(uint256 _value) internal virtual {
        require(eToken.repayBorrow(_value) == 0, 'repayBorrow ubalance error');
    }
}


pragma solidity 0.6.12;



contract SafeBoxFoxCToken is SafeBoxFoxCTokenImpl, ReentrancyGuardUpgradeable, OwnableUpgradeable, IC
    using SafeMathUpgradeable for uint256;
    using SafeERC20Upgradeable for IERC20Upgradeable;

    struct BorrowInfo {
        address strategy;      // borrow from strategy
        uint256 pid;           // borrow to pool
        address owner;         // borrower
        uint256 amount;        // borrow amount
        uint256 bPoints;       // borrow proportion
    }

    // supply manager
    uint256 public accDebtPerSupply;    // platform debt is shared to each supply lptoken

    // borrow manager
    BorrowInfo[] public borrowInfo;     // borrow order info
    mapping(address => mapping(address => mapping(uint256 => uint256))) public borrowIndex;   // _acc
    mapping(address => uint256) public  accountBorrowPoints;   // _account,  amount
```

```
    uint256 public lastBorrowCurrent;   // last settlement for

    uint256 public borrowTotalPoints;          // total of user bPoints
    uint256 public borrowTotalAmountWithPlatform;  // total of user borrows and interests and platfor
    uint256 public borrowTotalAmount;          // total of user borrows and interests
    uint256 public borrowTotal;                // total of user borrows

    uint256 public borrowLimitRate;    // borrow limit,  max = borrowTotal * borrowLimitRate / 1e9, d
    uint256 public borrowMinAmount;          // borrow min amount limit

    mapping(address => bool) public blacklist;  // deposit blacklist
    bool public depositEnabled;
    bool public emergencyRepayEnabled;
    bool public emergencyWithdrawEnabled;

    address public override bank;       // borrow can from bank only
    address public override token;      // deposit and borrow token

    address public compActionPool;          // action pool for borrow rewards
    uint256 public CTOKEN_BORROW;  // action pool borrow action id

    uint256 public optimalUtilizationRate1;  // Lending rate 1, ideal 1e9, default = 60%
    uint256 public optimalUtilizationRate2;  // Lending rate 2, ideal 1e9, default = 70%
    uint256 public stableRateSlope1;         // loan interest times in max borrow rate 1
    uint256 public stableRateSlope2;        // loan interest times in max borrow rate 2

    address public devAddr;

    event SafeBoxDeposit(address indexed user, uint256 amount);
    event SafeBoxWithdraw(address indexed user, uint256 amount);
    event SafeBoxClaim(address indexed user, uint256 amount);

    event SetBlacklist(address indexed _account, bool _newset);
    event SetBorrowLimitRate(uint256 oldRate, uint256 newRate);
    event SetOptimalUtilizationRate(uint256 oldV1, uint256 oldV2, uint256 newV1, uint256 newV2);
    event SetStableRateSlope(uint256 oldV1, uint256 oldV2, uint256 newV1, uint256 newV2);

    function __SafeBoxFoxCToken_init(address _bank, address _cToken, address _devAddr) public initial
        __Ownable_init();
        __ReentrancyGuard_init();

        __SafeBoxFoxCTokenImpl_init(_cToken);

        CTOKEN_BORROW = 1;  // action pool borrow action id
        devAddr = _devAddr;

        optimalUtilizationRate1 = 1e9;  // Lending rate 1, ideal 1e9, default = 60%
        optimalUtilizationRate2 = 1e9;  // Lending rate 2, ideal 1e9, default = 70%
        stableRateSlope1 = 1e9;         // loan interest times in max borrow rate 1
        stableRateSlope2 = 1e9;         // loan interest times in max borrow rate 2

        depositEnabled = true;
        borrowLimitRate = 7.5e8;

        token = baseToken();
        require(IERC20Upgradeable(token).totalSupply() >= 0, 'token error');
        bank = _bank;

        // 0 id  Occupied,  Available bid never be zero
        borrowInfo.push(BorrowInfo(address(0), 0, address(0), 0, 0));
    }

    modifier onlyBank() {
        require(bank == msg.sender, 'borrow only from bank');
        _;
    }
```

```
    // link to actionpool , for borrower s allowance
    function getCATPoolInfo(uint256 _pid) external virtual override view
        returns (address lpToken, uint256 allocRate, uint256 totalPoints, uint256 totalAmount) {
            _pid;
            lpToken = token;
            allocRate = 5e8; // never use
            totalPoints = borrowTotalPoints;
            totalAmount = borrowTotalAmountWithPlatform;
    }

    function getCATUserAmount(uint256 _pid, address _account) external virtual override view
        returns (uint256 acctPoints) {
            _pid;
            acctPoints = accountBorrowPoints[_account];
    }

    function getSource() external virtual override view returns (string memory) {
        return 'dfox';
    }

    // blacklist
    function setBlacklist(address _account, bool _newset) external onlyOwner {
        blacklist[_account] = _newset;
        emit SetBlacklist(_account, _newset);
    }

    function setCompAcionPool(address _compActionPool) public onlyOwner {
        compActionPool = _compActionPool;
    }

    function setDevAddr(address _devAddr) public {
        require(devAddr == msg.sender, 'wu?');
        devAddr = _devAddr;
    }

    function setBorrowLimitRate(uint256 _borrowLimitRate) external onlyOwner {
        require(_borrowLimitRate <= 1e9, 'rate too high');
        emit SetBorrowLimitRate(borrowLimitRate, _borrowLimitRate);
        borrowLimitRate = _borrowLimitRate;
    }

    function setBorrowMinAmount(uint256 _borrowMinAmount) external onlyOwner {
        borrowMinAmount = _borrowMinAmount;
    }

    function setEmergencyRepay(bool _emergencyRepayEnabled) external onlyOwner {
        emergencyRepayEnabled = _emergencyRepayEnabled;
    }

    function setEmergencyWithdraw(bool _emergencyWithdrawEnabled) external onlyOwner {
        emergencyWithdrawEnabled = _emergencyWithdrawEnabled;
    }

    // for platform borrow interest rate
    function setOptimalUtilizationRate(uint256 _optimalUtilizationRate1, uint256 _optimalUtilizationR
        require(_optimalUtilizationRate1 <= 1e9 &&
                _optimalUtilizationRate2 <= 1e9 &&
                _optimalUtilizationRate1 < _optimalUtilizationRate2
                , 'rate set error');
        emit SetOptimalUtilizationRate(optimalUtilizationRate1, optimalUtilizationRate2, _optimalUtil
        optimalUtilizationRate1 = _optimalUtilizationRate1;
        optimalUtilizationRate2 = _optimalUtilizationRate2;
    }

    function setStableRateSlope(uint256 _stableRateSlope1, uint256 _stableRateSlope2) external onlyOw
```

```
        require(_stableRateSlope1 <= 1e4*1e9 && _stableRateSlope1 >= 1e9 &&
                _stableRateSlope2 <= 1e4*1e9 && _stableRateSlope2 >= 1e9 , 'rate set error');
        emit SetStableRateSlope(stableRateSlope1, stableRateSlope2, _stableRateSlope1, _stableRateSlo
        stableRateSlope1 = _stableRateSlope1;
        stableRateSlope2 = _stableRateSlope2;
    }

    function supplyRatePerBlock() external override view returns (uint256) {
        return ctokenSupplyRatePerBlock();
    }

    function borrowRatePerBlock() external override view returns (uint256) {
        return ctokenBorrowRatePerBlock().mul(getBorrowFactorPrewiew()).div(1e9);
    }

    function borrowInfoLength() external override view returns (uint256) {
        return borrowInfo.length.sub(1);
    }

    function getBorrowInfo(uint256 _bid) external override view
        returns (address owner, uint256 amount, address strategy, uint256 pid) {

        strategy = borrowInfo[_bid].strategy;
        pid = borrowInfo[_bid].pid;
        owner = borrowInfo[_bid].owner;
        amount = borrowInfo[_bid].amount;
    }

    function getBorrowFactorPrewiew() public virtual view returns (uint256) {
        return _getBorrowFactor(getDepositTotal());
    }

    function getBorrowFactor() public virtual returns (uint256) {
        return _getBorrowFactor(call_balanceOfBaseToken_this());
    }

    function _getBorrowFactor(uint256 supplyAmount) internal virtual view returns (uint256 value) {
        if(supplyAmount <= 0) {
            return uint256(1e9);
        }
        uint256 borrowRate = getBorrowTotal().mul(1e9).div(supplyAmount);
        if(borrowRate <= optimalUtilizationRate1) {
            return uint256(1e9);
        }
        uint256 value1 = stableRateSlope1.sub(1e9).mul(borrowRate.sub(optimalUtilizationRate1))
                    .div(uint256(1e9).sub(optimalUtilizationRate1))
                    .add(uint256(1e9));
        if(borrowRate <= optimalUtilizationRate2) {
            value = value1;
            return value;
        }
        uint256 value2 = stableRateSlope2.sub(1e9).mul(borrowRate.sub(optimalUtilizationRate2))
                    .div(uint256(1e9).sub(optimalUtilizationRate2))
                    .add(uint256(1e9));
        value = value2 > value1 ? value2 : value1;
    }

    function getBorrowTotal() public virtual override view returns (uint256) {
        return borrowTotalAmountWithPlatform;
    }

    function getDepositTotal() public virtual override view returns (uint256) {
        return totalSupply().mul(getBaseTokenPerLPToken()).div(1e18);
    }

    function getBaseTokenPerLPToken() public virtual override view returns (uint256) {
```

```
        return getBaseTokenPerCToken();
    }

    function pendingSupplyAmount(address _account) external virtual override view returns (uint256 va
        value = call_balanceOf(address(this), _account).mul(getBaseTokenPerLPToken()).div(1e18);
    }

    function pendingBorrowAmount(uint256 _bid) public virtual override view returns (uint256 value) {
        value = borrowInfo[_bid].amount;
    }

    // borrow interest, the sum of filda interest and platform interest
    function pendingBorrowRewards(uint256 _bid) public virtual override view returns (uint256 value)
        if(borrowTotalPoints <= 0) {
            return 0;
        }
        value = borrowInfo[_bid].bPoints.mul(borrowTotalAmountWithPlatform).div(borrowTotalPoints);
        value = TenMath.safeSub(value, borrowInfo[_bid].amount);
    }

    // deposit
    function deposit(uint256 _value) external virtual override nonReentrant {
        update();
        IERC20Upgradeable(token).safeTransferFrom(msg.sender, address(this), _value);
        _deposit(msg.sender, _value);
    }

    function _deposit(address _account, uint256 _value) internal returns (uint256) {
        require(depositEnabled, 'safebox closed');
        require(!blacklist[_account], 'address in blacklist');
        // token held in contract
        uint256 balanceInput = call_balanceOf(token, address(this));
        require(balanceInput > 0 &&  balanceInput >= _value, 'where s token?');

        // update booking, mintValue is number of deposit credentials
        uint256 mintValue = ctokenDeposit(_value);
        if(mintValue > 0) {
            _mint(_account, mintValue);
        }
        emit SafeBoxDeposit(_account, mintValue);
        return mintValue;
    }

    function withdraw(uint256 _tTokenAmount) external virtual override nonReentrant {
        update();
        _withdraw(msg.sender, _tTokenAmount);
    }

    function _withdraw(address _account, uint256 _tTokenAmount) internal returns (uint256) {
        // withdraw if lptokens value is not up borrowLimitRate
        if(_tTokenAmount > balanceOf(_account)) {
            _tTokenAmount = balanceOf(_account);
        }
        uint256 maxBorrowAmount = call_balanceOfCToken_this().sub(_tTokenAmount)
                                    .mul(getBaseTokenPerLPToken()).div(1e18)
                                    .mul(borrowLimitRate).div(1e9);
        require(maxBorrowAmount >= borrowTotalAmountWithPlatform, 'no money to withdraw');

        _burn(_account, uint256(_tTokenAmount));

        if(accDebtPerSupply > 0) {
            // If platform loss, the loss will be shared by supply
            uint256 debtAmount = _tTokenAmount.mul(accDebtPerSupply).div(1e18);
            require(_tTokenAmount >= debtAmount, 'debt too much');
            _tTokenAmount = _tTokenAmount.sub(debtAmount);
        }
```

```
        ctokenWithdraw(_tTokenAmount);
        tokenSafeTransfer(address(token), _account);
        emit SafeBoxWithdraw(_account, _tTokenAmount);
        return _tTokenAmount;
    }

    function claim(uint256 _value) external virtual override nonReentrant {
        update();
        _claim(msg.sender, uint256(_value));
    }

    function _claim(address _account, uint256 _value) internal {
        emit SafeBoxClaim(_account, _value);
    }

    function getBorrowId(address _strategy, uint256 _pid, address _account)
        public virtual override view returns (uint256 borrowId) {
        borrowId = borrowIndex[_account][_strategy][_pid];
    }

    function getBorrowId(address _strategy, uint256 _pid, address _account, bool _add)
        external virtual override onlyBank returns (uint256 borrowId) {

        require(_strategy != address(0), 'borrowid _strategy error');
        require(_account != address(0), 'borrowid _account error');
        borrowId = getBorrowId(_strategy, _pid, _account);
        if(borrowId == 0 && _add) {
            borrowInfo.push(BorrowInfo(_strategy, _pid, _account, 0, 0));
            borrowId = borrowInfo.length.sub(1);
            borrowIndex[_account][_strategy][_pid] = borrowId;
        }
        require(borrowId > 0, 'not found borrowId');
    }

    function borrow(uint256 _bid, uint256 _value, address _to) external virtual override onlyBank {
        update();
        _borrow(_bid, _value, _to);
    }

    function _borrow(uint256 _bid, uint256 _value, address _to) internal {
        // withdraw if lptokens value is not up borrowLimitRate
        uint256 maxBorrowAmount = call_balanceOfCToken_this()
                                  .mul(getBaseTokenPerLPToken()).div(1e18)
                                  .mul(borrowLimitRate).div(1e9);
        require(maxBorrowAmount >= borrowTotalAmountWithPlatform.add(_value), 'no money to borrow');
        require(_value >= borrowMinAmount, 'borrow amount too low');

        BorrowInfo storage borrowCurrent = borrowInfo[_bid];

        // borrow
        uint256 ubalance = ctokenBorrow(_value);
        require(ubalance == _value, 'token borrow error');

        tokenSafeTransfer(address(token), _to);

        // booking
        uint256 addPoint = _value;
        if(borrowTotalPoints > 0) {
            addPoint = _value.mul(borrowTotalPoints).div(borrowTotalAmountWithPlatform);
        }

        borrowCurrent.bPoints = borrowCurrent.bPoints.add(addPoint);
        borrowTotalPoints = borrowTotalPoints.add(addPoint);
        borrowTotalAmountWithPlatform = borrowTotalAmountWithPlatform.add(_value);
        lastBorrowCurrent = call_borrowBalanceCurrent_this();
```

```
        borrowCurrent.amount = borrowCurrent.amount.add(_value);
        borrowTotal = borrowTotal.add(_value);
        borrowTotalAmount = borrowTotalAmount.add(_value);

        // notify for action pool
        uint256 accountBorrowPointsOld = accountBorrowPoints[borrowCurrent.owner];
        accountBorrowPoints[borrowCurrent.owner] = accountBorrowPoints[borrowCurrent.owner].add(addPo

        if(compActionPool != address(0) && addPoint > 0) {
            IActionPools(compActionPool).onAcionIn(CTOKEN_BORROW, borrowCurrent.owner,
                    accountBorrowPointsOld, accountBorrowPoints[borrowCurrent.owner]);
        }
        return ;
    }

    function repay(uint256 _bid, uint256 _value) external virtual override {
        update();
        _repay(_bid, _value);
    }

    function _repay(uint256 _bid, uint256 _value) internal {
        BorrowInfo storage borrowCurrent = borrowInfo[_bid];

        uint256 removedPoints;
        if(_value >= pendingBorrowRewards(_bid).add(borrowCurrent.amount)) {
            removedPoints = borrowCurrent.bPoints;
        }else{
            removedPoints = _value.mul(borrowTotalPoints).div(borrowTotalAmountWithPlatform);
            removedPoints = TenMath.min(removedPoints, borrowCurrent.bPoints);
        }

        // booking
        uint256 userAmount = removedPoints.mul(borrowCurrent.amount).div(borrowCurrent.bPoints); // t
        uint256 repayAmount = removedPoints.mul(borrowTotalAmount).div(borrowTotalPoints); // to repa
        uint256 platformAmount = TenMath.safeSub(removedPoints.mul(borrowTotalAmountWithPlatform).div
                                 repayAmount);  // platform interest

        borrowCurrent.bPoints = TenMath.safeSub(borrowCurrent.bPoints, removedPoints);
        borrowTotalPoints = TenMath.safeSub(borrowTotalPoints, removedPoints);
        borrowTotalAmountWithPlatform = TenMath.safeSub(borrowTotalAmountWithPlatform, repayAmount.ad
        lastBorrowCurrent = call_borrowBalanceCurrent_this();

        borrowCurrent.amount = TenMath.safeSub(borrowCurrent.amount, userAmount);
        borrowTotal = TenMath.safeSub(borrowTotal, userAmount);
        borrowTotalAmount = TenMath.safeSub(borrowTotalAmount, repayAmount);

        // platform interest
        if(platformAmount > 0 && devAddr != address(0)) {
            IERC20Upgradeable(token).safeTransfer(devAddr, platformAmount);
        }

        // repay resize
        uint256 restRepayAmount = 0;
        if(repayAmount > lastBorrowCurrent) {
            restRepayAmount = repayAmount.sub(lastBorrowCurrent);
        }

        // repay borrow
        repayAmount = TenMath.min(repayAmount, lastBorrowCurrent);

        ctokenRepayBorrow(repayAmount);
        lastBorrowCurrent = call_borrowBalanceCurrent_this();

        // return of the rest
        if(restRepayAmount > 0) {
```

```
            tokenSafeTransferAmount(token, restRepayAmount, owner());
        }
        tokenSafeTransfer(token, msg.sender);

        // notify for action pool
        uint256 accountBorrowPointsOld = accountBorrowPoints[borrowCurrent.owner];
        accountBorrowPoints[borrowCurrent.owner] = TenMath.safeSub(accountBorrowPoints[borrowCurrent.

        if(compActionPool != address(0) && removedPoints > 0) {
            IActionPools(compActionPool).onAcionOut(CTOKEN_BORROW, borrowCurrent.owner,
                    accountBorrowPointsOld, accountBorrowPoints[borrowCurrent.owner]);
        }
        return ;
    }

    function emergencyWithdraw() external virtual override nonReentrant {
        require(emergencyWithdrawEnabled, 'not in emergency');

        uint256 withdrawAmount = call_balanceOf(address(this), msg.sender);
        _burn(msg.sender, withdrawAmount);

        if(accDebtPerSupply > 0) {
            // If platform loss, the loss will be shared by supply
            uint256 debtAmount = withdrawAmount.mul(accDebtPerSupply).div(1e18);
            require(withdrawAmount >= debtAmount, 'debt too much');
            withdrawAmount = withdrawAmount.sub(debtAmount);
        }

        // withdraw ctoken
        ctokenWithdraw(withdrawAmount);

        tokenSafeTransfer(address(token), msg.sender);
    }

    function emergencyRepay(uint256 _bid) external virtual override nonReentrant {
        require(emergencyRepayEnabled, 'not in emergency');
        // in emergency mode , only repay loan
        BorrowInfo storage borrowCurrent = borrowInfo[_bid];

        uint256 repayAmount = borrowCurrent.amount;

        IERC20Upgradeable(baseToken()).safeTransferFrom(msg.sender, address(this), repayAmount);
        ctokenRepayBorrow(repayAmount);

        uint256 accountBorrowPointsOld = accountBorrowPoints[borrowCurrent.owner];
        accountBorrowPoints[borrowCurrent.owner] = TenMath.safeSub(accountBorrowPoints[borrowCurrent.

        // booking
        borrowTotal = TenMath.safeSub(borrowTotal, repayAmount);
        borrowTotalPoints = TenMath.safeSub(borrowTotalPoints, borrowCurrent.bPoints);
        borrowTotalAmount = TenMath.safeSub(borrowTotalAmount, repayAmount);
        borrowTotalAmountWithPlatform = TenMath.safeSub(borrowTotalAmountWithPlatform, repayAmount);
        borrowCurrent.amount = 0;
        borrowCurrent.bPoints = 0;
        lastBorrowCurrent = call_borrowBalanceCurrent_this();
    }

    function update() public virtual override {
        _update();
    }

    function _update() internal {
        // update borrow interest
        uint256 lastBorrowCurrentNow = call_borrowBalanceCurrent_this();
        if(lastBorrowCurrentNow != lastBorrowCurrent && borrowTotal > 0) {
            if(lastBorrowCurrentNow >= lastBorrowCurrent) {
```

```
                // booking
                uint256 newDebtAmount1 = lastBorrowCurrentNow.sub(lastBorrowCurrent);
                uint256 newDebtAmount2 = newDebtAmount1.mul(getBorrowFactor()).div(1e9);
                borrowTotalAmount = borrowTotalAmount.add(newDebtAmount1);
                borrowTotalAmountWithPlatform = borrowTotalAmountWithPlatform.add(newDebtAmount2);
            }
            lastBorrowCurrent = lastBorrowCurrentNow;
        }

        // manage ctoken amount
        uint256 uCTokenTotalAmount = call_balanceOfCToken_this();
        if(uCTokenTotalAmount >= totalSupply()) {
            // The platform has no debt
            accDebtPerSupply = 0;
        }
        if(totalSupply() > 0 && accDebtPerSupply > 0) {
            // The platform has debt, uCTokenTotalAmount will be totalSupply()
            uCTokenTotalAmount = uCTokenTotalAmount.add(accDebtPerSupply.mul(totalSupply()).div(1e18)
        }
        if(uCTokenTotalAmount < totalSupply()) {
            // totalSupply() != 0  new debt divided equally
            accDebtPerSupply = accDebtPerSupply.add(totalSupply().sub(uCTokenTotalAmount).mul(1e18).d
        } else if(uCTokenTotalAmount > totalSupply() && accDebtPerSupply > 0) {
            // reduce debt divided equally
            uint256 accDebtReduce = uCTokenTotalAmount.sub(totalSupply()).mul(1e18).div(totalSupply()
            accDebtReduce = TenMath.min(accDebtReduce, accDebtPerSupply);
            accDebtPerSupply = accDebtPerSupply.sub(accDebtReduce);
        }

        if(compActionPool != address(0)) {
            IActionPools(compActionPool).onAcionUpdate(CTOKEN_BORROW);
        }
    }

    function mintDonate(uint256 _value) public virtual override nonReentrant {
        IERC20Upgradeable(token).safeTransferFrom(msg.sender, address(this), _value);
        ctokenDeposit(_value);
        update();
    }

    function tokenSafeTransfer(address _token, address _to) internal {
        uint256 value = IERC20Upgradeable(_token).balanceOf(address(this));
        if(value > 0) {
            IERC20Upgradeable(_token).transfer(_to, value);
        }
    }

    function tokenSafeTransferAmount(address _token, uint256 _want, address _to) internal {
        uint256 value = IERC20Upgradeable(_token).balanceOf(address(this));
        value = TenMath.min(value, _want);
        if(value > 0) {
            IERC20Upgradeable(_token).transfer(_to, value);
        }
    }
}


pragma solidity 0.6.12;




// Distribution of BOO Compound token
contract SafeBoxFox is SafeBoxFoxCToken {
    using SafeMathUpgradeable for uint256;
```

```
    using SafeERC20Upgradeable for IERC20Upgradeable;

    IERC20Upgradeable public REWARDS_TOKEN;

    uint256 public lastRewardsTokenBlock;        // rewards update

    address public actionPoolRewards;            // address for action pool
    uint256 public poolDepositId;               // poolid of depositor s token rewards in action pool
    uint256 public poolBorrowId;                // poolid of borrower s token rewards in action pool

    uint256 public REWARDS_DEPOSIT_CALLID;      // depositinfo callid for action callback
    uint256 public REWARDS_BORROW_CALLID;       // borrowinfo callid for comp action callback

    event SetRewardsDepositPool(address _actionPoolRewards, uint256 _piddeposit);
    event SetRewardsBorrowPool(address _compActionPool, uint256 _pidborrow);

    function initialize(address _bank, address _cToken, address _devAddr) public initializer {
        __Ownable_init();

        __SafeBoxFoxCToken_init(_bank, _cToken, _devAddr);

        REWARDS_TOKEN = IERC20Upgradeable(0);
        REWARDS_DEPOSIT_CALLID = 16;
        REWARDS_BORROW_CALLID = 18;
    }

    function update() public virtual override {
        _update();
        updatetoken();
    }

    // mint rewards for supplies to action pools
    function setRewardsDepositPool(address _actionPoolRewards, uint256 _piddeposit) public onlyOwner
        actionPoolRewards = _actionPoolRewards;
        poolDepositId = _piddeposit;
        emit SetRewardsDepositPool(_actionPoolRewards, _piddeposit);
    }

    // mint rewards for borrows to comp action pools
    function setRewardsBorrowPool(uint256 _pidborrow) public onlyOwner {
        _checkActionPool(compActionPool, _pidborrow, REWARDS_BORROW_CALLID);
        poolBorrowId = _pidborrow;
        emit SetRewardsBorrowPool(compActionPool, _pidborrow);
    }

    function _checkActionPool(address _actionPool, uint256 _pid, uint256 _rewardscallid) internal vie
        (address callFrom, uint256 callId, address rewardToken)
            = IActionPools(_actionPool).getPoolInfo(_pid);
        require(callFrom == address(this), 'call from error');
        require(callId == _rewardscallid, 'callid error');
    }

    function deposit(uint256 _value) external virtual override nonReentrant {
        update();
        IERC20Upgradeable(token).safeTransferFrom(msg.sender, address(this), _value);
        _deposit(msg.sender, _value);
    }

    function withdraw(uint256 _tTokenAmount) external virtual override nonReentrant {
        update();
        _withdraw(msg.sender, _tTokenAmount);
    }

    function borrow(uint256 _bid, uint256 _value, address _to) external virtual override onlyBank {
        update();
        address owner = borrowInfo[_bid].owner;
```

```
        uint256 accountBorrowPointsOld = accountBorrowPoints[owner];
        _borrow(_bid, _value, _to);

        if(compActionPool != address(0) && _value > 0) {
            IActionPools(compActionPool).onAcionIn(REWARDS_BORROW_CALLID, owner,
                    accountBorrowPointsOld, accountBorrowPoints[owner]);
        }
    }

    function repay(uint256 _bid, uint256 _value) external virtual override {
        update();
        address owner = borrowInfo[_bid].owner;
        uint256 accountBorrowPointsOld = accountBorrowPoints[owner];
        _repay(_bid, _value);

        if(compActionPool != address(0) && _value > 0) {
            IActionPools(compActionPool).onAcionOut(REWARDS_BORROW_CALLID, owner,
                    accountBorrowPointsOld, accountBorrowPoints[owner]);
        }
    }

    function updatetoken() public {
        if(lastRewardsTokenBlock == block.number) {
            return ;
        }
        lastRewardsTokenBlock = block.number;
    }

    function claim(uint256 _value) external virtual override nonReentrant {
        update();
        _claim(msg.sender, _value);
    }
}
```

Timelock.sol

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
```

```
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
```

```
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
```

```
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}

pragma solidity >=0.6.0 <0.8.0;


contract Timelock {
    using SafeMath for uint;

    event NewAdmin(address indexed newAdmin);
    event NewPendingAdmin(address indexed newPendingAdmin);
    event NewDelay(uint indexed newDelay);
```

```
    event CancelTransaction(bytes32 indexed txHash, address indexed target, uint value, string signat
    event ExecuteTransaction(bytes32 indexed txHash, address indexed target, uint value, string signa
    event QueueTransaction(bytes32 indexed txHash, address indexed target, uint value, string signatu

    uint public constant GRACE_PERIOD = 14 days;
    uint public constant MINIMUM_DELAY = 2 days;
    uint public constant MAXIMUM_DELAY = 30 days;

    address public admin;
    address public pendingAdmin;
    uint public delay;

    mapping (bytes32 => bool) public queuedTransactions;


    constructor(address admin_, uint delay_) public {
        require(delay_ >= MINIMUM_DELAY, "Timelock::constructor: Delay must exceed minimum delay.");
        require(delay_ <= MAXIMUM_DELAY, "Timelock::setDelay: Delay must not exceed maximum delay.");

        admin = admin_;
        delay = delay_;
    }

    receive() external payable { }

    function setDelay(uint delay_) public {
        require(msg.sender == address(this), "Timelock::setDelay: Call must come from Timelock.");
        require(delay_ >= MINIMUM_DELAY, "Timelock::setDelay: Delay must exceed minimum delay.");
        require(delay_ <= MAXIMUM_DELAY, "Timelock::setDelay: Delay must not exceed maximum delay.");
        delay = delay_;

        emit NewDelay(delay);
    }

    function acceptAdmin() public {
        require(msg.sender == pendingAdmin, "Timelock::acceptAdmin: Call must come from pendingAdmin.
        admin = msg.sender;
        pendingAdmin = address(0);

        emit NewAdmin(admin);
    }

    function setPendingAdmin(address pendingAdmin_) public {
        require(msg.sender == address(this), "Timelock::setPendingAdmin: Call must come from Timelock
        pendingAdmin = pendingAdmin_;

        emit NewPendingAdmin(pendingAdmin);
    }

    function queueTransaction(address target, uint value, string memory signature, bytes memory data,
        require(msg.sender == admin, "Timelock::queueTransaction: Call must come from admin.");
        require(eta >= getBlockTimestamp().add(delay), "Timelock::queueTransaction: Estimated executi

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        queuedTransactions[txHash] = true;

        emit QueueTransaction(txHash, target, value, signature, data, eta);
        return txHash;
    }

    function cancelTransaction(address target, uint value, string memory signature, bytes memory data
        require(msg.sender == admin, "Timelock::cancelTransaction: Call must come from admin.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        queuedTransactions[txHash] = false;
```

```
        emit CancelTransaction(txHash, target, value, signature, data, eta);
    }

    function executeTransaction(address target, uint value, string memory signature, bytes memory dat
        require(msg.sender == admin, "Timelock::executeTransaction: Call must come from admin.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        require(queuedTransactions[txHash], "Timelock::executeTransaction: Transaction hasn't been qu
        require(getBlockTimestamp() >= eta, "Timelock::executeTransaction: Transaction hasn't surpass
        require(getBlockTimestamp() <= eta.add(GRACE_PERIOD), "Timelock::executeTransaction: Transact

        queuedTransactions[txHash] = false;

        bytes memory callData;

        if (bytes(signature).length == 0) {
            callData = data;
        } else {
            callData = abi.encodePacked(bytes4(keccak256(bytes(signature))), data);
        }

        // solium-disable-next-line security/no-call-value
        (bool success, bytes memory returnData) = target.call{value: value}(callData);
        require(success, "Timelock::executeTransaction: Transaction execution reverted.");

        emit ExecuteTransaction(txHash, target, value, signature, data, eta);

        return returnData;
    }

    function getBlockTimestamp() internal view returns (uint) {
        // solium-disable-next-line security/no-block-members
        return block.timestamp;
    }
}
```

StrategyV2PairHelper.sol

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20Upgradeable {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
```

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMathUpgradeable {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
```

```
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
```

```
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
```

```
        return a % b;
    }
}


pragma solidity >=0.6.2 <0.8.0;

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

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
```

```
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

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

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

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

```
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

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

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

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

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


pragma solidity >=0.6.0 <0.8.0;


/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20Upgradeable {
    using SafeMathUpgradeable for uint256;
    using AddressUpgradeable for address;

    function safeTransfer(IERC20Upgradeable token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20Upgradeable token, address from, address to, uint256 value) inter
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
```

```
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20Upgradeable token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20Upgradeable token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}


pragma solidity 0.6.12;

interface ISafeBox {

    function bank() external view returns(address);

    function token() external view returns(address);

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

    function supplyRatePerBlock() external view returns (uint256);
    function borrowRatePerBlock() external view returns (uint256);

    function getBorrowInfo(uint256 _bid) external view
            returns (address owner, uint256 amount, address strategy, uint256 pid);
    function getBorrowId(address _strategy, uint256 _pid, address _account) external view returns (ui
    function getBorrowId(address _strategy, uint256 _pid, address _account, bool _add) external retur
    function getDepositTotal() external view returns (uint256);
    function getBorrowTotal() external view returns (uint256);
    // function getBorrowAmount(address _account) external view returns (uint256 value);
    function getBaseTokenPerLPToken() external view returns (uint256);
```

```
    function deposit(uint256 _value) external;
    function withdraw(uint256 _value) external;

    function emergencyWithdraw() external;
    function emergencyRepay(uint256 _bid) external;

    function borrowInfoLength() external view returns (uint256);

    function borrow(uint256 _bid, uint256 _value, address _to) external;
    function repay(uint256 _bid, uint256 _value) external;
    function claim(uint256 _tTokenAmount) external;

    function update() external;
    function mintDonate(uint256 _value) external;

    function pendingSupplyAmount(address _account) external view returns (uint256 value);
    function pendingBorrowAmount(uint256 _bid) external view returns (uint256 value);
    function pendingBorrowRewards(uint256 _bid) external view returns (uint256 value);
}

pragma solidity 0.6.12;


interface IActionPools {

    function getPoolInfo(uint256 _pid) external view
        returns (address callFrom, uint256 callId, address rewardToken);
    function mintRewards(uint256 _callId) external;
    function getPoolIndex(address _callFrom, uint256 _callId) external view returns (uint256[] memory

    function onAcionIn(uint256 _callId, address _account, uint256 _fromAmount, uint256 _toAmount) ext
    function onAcionOut(uint256 _callId, address _account, uint256 _fromAmount, uint256 _toAmount) ex
    function onAcionClaim(uint256 _callId, address _account) external;
    function onAcionEmergency(uint256 _callId, address _account) external;
    function onAcionUpdate(uint256 _callId) external;
}

pragma solidity 0.6.12;

interface ITenBankHall {
    function makeBorrowFrom(uint256 _pid, address _account, address _debtFrom, uint256 _value) extern
}

pragma solidity 0.6.12;


interface ITenBankHallV2 is ITenBankHall {
    function boxInfo(uint256 _boxid) external view returns (address);
    function boxIndex(address _boxaddr) external view returns (uint256);
    function boxlisted(address _boxaddr) external view returns (bool);

    function strategyInfo(uint256 _sid) external view returns (bool, address, uint256);
    function strategyIndex(address _strategy, uint256 _sid) external view returns (uint256);
}

pragma solidity 0.6.12;

interface IPriceChecker {
    function getPriceSlippage(address _lptoken) external view returns (uint256);
    function checkLPTokenPriceLimit(address _lptoken, bool _largeType) external view returns (bool);
}

pragma solidity 0.6.12;

interface IStrategyLink {
```

```
    event StrategyDeposit(address indexed strategy, uint256 indexed pid, address indexed user, uint25
    event StrategyBorrow(address indexed strategy, uint256 indexed pid, address indexed user, uint256
    event StrategyWithdraw(address indexed strategy, uint256 indexed pid, address indexed user, uint2
    event StrategyLiquidation(address indexed strategy, uint256 indexed pid, address indexed user, ui

    function bank() external view returns(address);
    function getSource() external view returns (string memory);
    function userInfo(uint256 _pid, address _account) external view returns (uint256,uint256,address,
    function getPoolInfo(uint256 _pid) external view  returns(address[] memory collateralToken, addre
    function getBorrowInfo(uint256 _pid, address _account) external view returns (address borrowFrom,
    function getBorrowAmount(uint256 _pid, address _account) external view returns (uint256 value);
    function getBorrowAmountInBaseToken(uint256 _pid, address _account) external view returns (uint25
    function getDepositAmount(uint256 _pid, address _account) external view returns (uint256 amount);

    function getPoolCollateralToken(uint256 _pid) external view returns (address[] memory collateralT
    function getPoollpToken(uint256 _pid) external view returns (address lpToken);
    function getBaseToken(uint256 _pid) external view returns (address baseToken);

    function poolLength() external view returns (uint256);

    function pendingRewards(uint256 _pid, address _account) external view returns (uint256 value);
    function pendingLPAmount(uint256 _pid, address _account) external view returns (uint256 value);

    // function massUpdatePools(uint256 _start, uint256 _end) external;
    function updatePool(uint256 _pid, uint256 _desirePrice, uint256 _slippage) external;

    function deposit(uint256 _pid, address _account, address _debtFrom, uint256 _bAmount, uint256 _de
    function depositLPToken(uint256 _pid, address _account, address _debtFrom, uint256 _bAmount, uint

    function withdraw(uint256 _pid, address _account, uint256 _rate, address _toToken, uint256 _desir
    function withdrawLPToken(uint256 _pid, address _account, uint256 _rate, uint256 _desirePrice, uin

    function emergencyWithdraw(uint256 _pid, address _account, uint256 _desirePrice, uint256 _slippag

    function liquidation(uint256 _pid, address _account, address _hunter, uint256 _maxDebt) external;
    function repayBorrow(uint256 _pid, address _account, uint256 _rate, bool _force) external;
}

pragma solidity 0.6.12;


interface IStrategyV2Pair is IStrategyLink {

    event AddPool(uint256 _pid, uint256 _poolId, address lpToken, address _baseToken);
    event SetMiniRewardAmount(uint256 _pid, uint256 _miniRewardAmount);
    event SetPoolImpl(address _oldv, address _new);
    event SetComponents(address _compActionPool, address _buyback, address _priceChecker, address _co
    event SetPoolConfig(uint256 _pid, string _key, uint256 _value);

    event StrategyBorrow2(address indexed strategy, uint256 indexed pid, address user, address indexe
    event StrategyRepayBorrow2(address indexed strategy, uint256 indexed pid, address user, address i
    event StrategyLiquidation2(address indexed strategy, uint256 indexed pid, address user, uint256 l

    function getBorrowInfo(uint256 _pid, address _account, uint256 _bindex)
        external view returns (address borrowFrom, uint256 bid, uint256 amount);
}

pragma solidity 0.6.12;

interface IStrategyV2PairHelper {
}
pragma solidity 0.6.12;


library TenMath {
  using SafeMathUpgradeable for uint256;
```

```
  function min(uint256 v1, uint256 v2) public pure returns (uint256 vr) {
    vr = v1 > v2 ? v2 : v1;
  }

  function safeSub(uint256 v1, uint256 v2) internal pure returns (uint256 vr) {
    vr = v1 > v2 ? v1.sub(v2) : 0;
  }

  // implementation from https://github.com/Uniswap/uniswap-lib/commit/99f3f28770640ba1bb1ff460ac7c52
  // original implementation: https://github.com/abdk-consulting/abdk-libraries-solidity/blob/master/
  function sqrt(uint256 x) internal pure returns (uint256) {
    if (x == 0) return 0;
    uint256 xx = x;
    uint256 r = 1;

    if (xx >= 0x100000000000000000000000000000000) {
      xx >>= 128;
      r <<= 64;
    }

    if (xx >= 0x10000000000000000) {
      xx >>= 64;
      r <<= 32;
    }
    if (xx >= 0x100000000) {
      xx >>= 32;
      r <<= 16;
    }
    if (xx >= 0x10000) {
      xx >>= 16;
      r <<= 8;
    }
    if (xx >= 0x100) {
      xx >>= 8;
      r <<= 4;
    }
    if (xx >= 0x10) {
      xx >>= 4;
      r <<= 2;
    }
    if (xx >= 0x8) {
      r <<= 1;
    }

    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1; // Seven iterations should be enough
    uint256 r1 = x / r;
    return (r < r1 ? r : r1);
  }
}


pragma solidity 0.6.12;


interface IStrategyConfig {
    // event
    event SetFeeGather(address _feeGatherOld, address _feeGather);
    event SetReservedGather(address _old, address _new);
    event SetBorrowFactor(address _strategy, uint256 _poolid, uint256 _borrowFactor);
```

```
    event SetLiquidationFactor(address _strategy, uint256 _poolid, uint256 _liquidationFactor);
    event SetFarmPoolFactor(address _strategy, uint256 _poolid, uint256 _farmPoolFactor);
    event SetDepositFee(address _strategy, uint256 _poolid, uint256 _depositFee);
    event SetWithdrawFee(address _strategy, uint256 _poolid, uint256 _withdrawFee);
    event SetRefundFee(address _strategy, uint256 _poolid, uint256 _refundFee);
    event SetClaimFee(address _strategy, uint256 _poolid, uint256 _claimFee);
    event SetLiquidationFee(address _strategy, uint256 _poolid, uint256 _liquidationFee);

    // factor
    function getBorrowFactor(address _strategy, uint256 _poolid) external view returns (uint256);
    function setBorrowFactor(address _strategy, uint256 _poolid, uint256 _borrowFactor) external;

    function getLiquidationFactor(address _strategy, uint256 _poolid) external view returns (uint256)
    function setLiquidationFactor(address _strategy, uint256 _poolid, uint256 _liquidationFactor) ext

    function getFarmPoolFactor(address _strategy, uint256 _poolid) external view returns (uint256 val
    function setFarmPoolFactor(address _strategy, uint256 _poolid, uint256 _farmPoolFactor) external;

    // fee manager
    function getDepositFee(address _strategy, uint256 _poolid) external view returns (address, uint25
    function setDepositFee(address _strategy, uint256 _poolid, uint256 _depositFee) external;

    function getWithdrawFee(address _strategy, uint256 _poolid) external view returns (address, uint2
    function setWithdrawFee(address _strategy, uint256 _poolid, uint256 _withdrawFee) external;

    function getRefundFee(address _strategy, uint256 _poolid) external view returns (address, uint256
    function setRefundFee(address _strategy, uint256 _poolid, uint256 _refundFee) external;

    function getClaimFee(address _strategy, uint256 _poolid) external view returns (address, uint256)
    function setClaimFee(address _strategy, uint256 _poolid, uint256 _claimFee) external;

    function getLiquidationFee(address _strategy, uint256 _poolid) external view returns (address, ui
    function setLiquidationFee(address _strategy, uint256 _poolid, uint256 _liquidationFee) external;
}

pragma solidity 0.6.12;

interface IStrategyV2SwapPool {

    function setStrategy(address _strategy) external;

    // get strategy
    function getName() external view returns (string memory);

    // swap functions
    function getPair(address _t0, address _t1) external view returns (address pairs);
    function getReserves(address _lpToken) external view returns (uint256 a, uint256 b);
    function getToken01(address _pairs) external view returns (address token0, address token1);
    function getAmountOut(address _tokenIn, address _tokenOut, uint256 _amountOut) external view retu
    function getAmountIn(address _tokenIn, uint256 _amountIn, address _tokenOut) external view return
    function getLPTokenAmountInBaseToken(address _lpToken, uint256 _lpTokenAmount, address _baseToken
    function swapTokenTo(address _tokenIn, uint256 _amountIn, address _tokenOut, address _toAddress)

    function optimalBorrowAmount(address _lpToken, uint256 _amount0, uint256 _amount1) external view
    function optimalDepositAmount(address lpToken, uint amtA, uint amtB) external view returns (uint

    // pool functions
    function getDepositToken(uint256 _poolId) external view returns (address lpToken);
    function getRewardToken(uint256 _poolId) external view returns (address rewardToken);
    function getPending(uint256 _poolId) external view returns (uint256 rewards);
    function deposit(uint256 _poolId, bool _autoPool) external returns (uint256 liquidity);
    function withdraw(uint256 _poolId, uint256 _liquidity, bool _autoPool) external returns (uint256
    function claim(uint256 _poolId) external returns (uint256 rewards);
    function extraRewards() external returns (address token, uint256 rewards);
}
```

```
pragma solidity 0.6.12;



contract StrategyV2Data {

    // Info of each user.
    struct UserInfo {
        uint256 lpAmount;       // deposit lptoken amount
        uint256 lpPoints;       // deposit proportion
        address[] borrowFrom;   // borrowFrom
        uint256[] bids;
    }

    // Info of each pool.
    struct PoolInfo {
        address[] collateralToken;      // collateral Token list, last must be baseToken
        address baseToken;              // baseToken can be borrowed
        address lpToken;                // lptoken to deposit
        uint256 poolId;                 // poolid for mdex pools
        uint256 lastRewardsBlock;       //
        uint256 totalPoints;            // total of user lpPoints
        uint256 totalLPAmount;          // total of user lpAmount
        uint256 totalLPReinvest;        // total of lptoken amount with totalLPAmount and reinvest re
        uint256 miniRewardAmount;       //
    }

    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping (uint256 => mapping (address => UserInfo)) public userInfo2;

    address public _bank;                // address of bank
    address public _this;
    address public helperImpl;

    IStrategyConfig public sconfig;
    mapping (uint256 => mapping (string => uint256)) public poolConfig;

    IStrategyV2SwapPool public swapPoolImpl;

    address public devAddr;
    IPriceChecker public priceChecker;
    IActionPools public compActionPool;     // address of comp action pool

    mapping(address=>bool) public whitelist;
    uint256[39] private __gap;
}


pragma solidity 0.6.12;



// Borrow and Repay
contract StrategyV2PairHelper is StrategyV2Data, IStrategyV2PairHelper {
    using SafeMathUpgradeable for uint256;
    using SafeERC20Upgradeable for IERC20Upgradeable;

    // check limit
    function checkAddPoolLimit(uint256 _pid) external view {
        PoolInfo memory pool = poolInfo[_pid];
        require(pool.collateralToken[0] == pool.baseToken ||
                pool.collateralToken[1] == pool.baseToken,
                'baseToken not in pair');
```

```
        require(swapPoolImpl.getDepositToken(pool.poolId) == pool.lpToken, 'lptoken error');
    }

    function checkDepositLimit(uint256 _pid, address _account, uint256 _orginSwapRate) external view
        _account;
        require(address(sconfig) != address(0), 'not config');
        uint256 farmLimit = sconfig.getFarmPoolFactor(_this, _pid);
        if(farmLimit > 0) {
            require(poolInfo[_pid].totalLPReinvest <= farmLimit, 'pool invest limit');
        }

        (uint256 res0, uint256 res1) = swapPoolImpl.getReserves(poolInfo[_pid].lpToken);
        uint256 curSwapRate = res0.mul(1e18).div(res1);
        uint256 slippage = poolConfig[_pid][string('deposit_slippage')];
        require(slippage > 0, 'deposit_slippage == 0');
        uint256 swapSlippage = _orginSwapRate.mul(1e9).div(curSwapRate);
        require(swapSlippage < slippage.add(1e9) &&
                swapSlippage > uint256(1e9).sub(slippage), 'pool slippage over');
    }

    function checkLiquidationLimit(uint256 _pid, address _account, bool liqucheck) external view {
        require(address(sconfig) != address(0), 'not config liquidation');

        uint256 liquRate = sconfig.getLiquidationFactor(_this, _pid);
        uint256 borrowAmount = IStrategyV2Pair(_this).getBorrowAmountInBaseToken(_pid, _account);
        if(borrowAmount == 0) {
            require(!liqucheck, 'no borrow');
            return ;
        }

        uint256 holdLPTokenAmount = IStrategyV2Pair(_this).pendingLPAmount(_pid, _account);
        uint256 holdBaseAmount = getLPTokenAmountInBaseToken(_pid, holdLPTokenAmount, poolInfo[_pid].
        if(liqucheck) {
            // check whether in liquidation
            if(holdBaseAmount > 0) {
                require(borrowAmount.mul(1e9).div(holdBaseAmount) > liquRate, 'check in liquidation')
            }
        } else {
            // check must not in liquidation
            require(holdBaseAmount > 0, 'no hold in liquidation');
            require(borrowAmount.mul(1e9).div(holdBaseAmount) < liquRate, 'check out liquidation');
        }
    }

    function checkOraclePrice(uint256 _pid, bool _large) public view {
        if(address(priceChecker) == address(0)) {
            return ;
        }
        bool oracle = priceChecker.checkLPTokenPriceLimit(poolInfo[_pid].lpToken, _large);
        require(oracle, 'oracle price limit');
    }

    function checkBorrowLimit(uint256 _pid, address _account) external view {
        PoolInfo memory pool = poolInfo[_pid];
        UserInfo storage user = userInfo2[_pid][_account];

        uint256 borrowAmount = IStrategyV2Pair(_this).getBorrowAmountInBaseToken(_pid, _account);
        uint256 lpTokenAmount = IStrategyV2Pair(_this).pendingLPAmount(_pid, _account);
        uint256 holdBaseAmount = getLPTokenAmountInBaseToken(_pid, lpTokenAmount, pool.baseToken);
        uint256 borrowFactor = sconfig.getBorrowFactor(_this, _pid);

        require(borrowAmount <= holdBaseAmount.sub(borrowAmount).mul(borrowFactor).div(1e9), 'borrow
    }

    function calcDepositFee(uint256 _pid)
        external view returns (address gather, uint256 _amount0, uint256 _amount1) {
```

```
        PoolInfo storage pool = poolInfo[_pid];
        uint256 feerate;
        (gather, feerate) = sconfig.getDepositFee(_this, _pid);
        _amount0 = IERC20Upgradeable(pool.collateralToken[0]).balanceOf(_this).mul(feerate).div(1e9);
        _amount1 = IERC20Upgradeable(pool.collateralToken[1]).balanceOf(_this).mul(feerate).div(1e9);
    }


    function calcRefundFee(uint256 _pid, uint256 _rewardAmount)
        public view returns (address gather, uint256 feeAmount) {
        PoolInfo storage pool = poolInfo[_pid];
        uint256 feerate;
        (gather, feerate) = sconfig.getRefundFee(_this, _pid);
        feeAmount = _rewardAmount.mul(feerate).div(1e9);
    }

    function calcBorrowAmount(uint256 _pid, address _account, address _debtFrom, uint256 _bAmount)
        external view returns (uint256 bindex, uint256 amount) {
        if(_debtFrom == address(0)) return (0, 0);

        PoolInfo memory pool = poolInfo[_pid];
        address token0 = pool.collateralToken[0];
        address token1 = pool.collateralToken[1];
        amount = _bAmount;

        {
            // check _debtFrom address book in bank
            uint256 debtid = ITenBankHallV2(_bank).boxIndex(_debtFrom);
            require(debtid > 0 || ITenBankHallV2(_bank).boxInfo(debtid) == _debtFrom, 'borrow from ba
        }

        address borrowToken = ISafeBox(_debtFrom).token();
        require(borrowToken == token0 || borrowToken == token1, 'debtFrom token error');

        bindex = borrowToken == token0 ? 0 : 1;
        if(amount > 0) {
            return (bindex, amount);
        }

        (uint256 res0, uint256 res1) = swapPoolImpl.getReserves(pool.lpToken);
        {
            (address token00,) = swapPoolImpl.getToken01(pool.lpToken);
            (res0, res1) = token00 == token0 ? (res0, res1) : (res1, res0);
        }

        uint256 balance0 = IERC20Upgradeable(token0).balanceOf(_this);
        uint256 balance1 = IERC20Upgradeable(token1).balanceOf(_this);
        if(bindex == 0) {
            amount = balance1.mul(res0).div(res1);
            if(amount > balance0) amount = amount.sub(balance0);
        } else {
            amount = balance0.mul(res1).div(res0);
            if(amount > balance1) amount = amount.sub(balance1);
        }
    }

    function calcRemoveLiquidity(uint256 _pid, address _account, uint256 _rate)
        external view returns (uint256 removedLPAmount, uint256 removedPoint) {
        PoolInfo memory pool = poolInfo[_pid];
        UserInfo storage user = userInfo2[_pid][_account];
        if(pool.totalPoints == 0) {
            return (0, 0);
        }
        if(_rate >= 1e9) {
            removedPoint = user.lpPoints;
        } else {
```

```
            removedPoint = user.lpPoints.mul(_rate).div(1e9);
        }
        removedLPAmount = removedPoint.mul(pool.totalLPReinvest).div(pool.totalPoints);
        removedLPAmount = TenMath.min(removedLPAmount, pool.totalLPReinvest);
    }

    function calcWithdrawFee(uint256 _pid, address _account, uint256 _rate)
        external view returns (address gather, uint256 a0, uint256 a1) {

        // sconfig.
        uint256 feerate;
        (gather, feerate) = sconfig.getWithdrawFee(_this, _pid);
        return (gather, 0, 0);

        uint256 borrowAmount = IStrategyV2Pair(_this).getBorrowAmountInBaseToken(_pid, _account);
        uint256 holdLPTokenAmount = IStrategyV2Pair(_this).pendingLPAmount(_pid, _account);
        uint256 holdLPRewardsAmount = IStrategyV2Pair(_this).pendingRewards(_pid, _account);
        if(holdLPRewardsAmount == 0 || holdLPTokenAmount == 0) {
            return (gather, 0, 0);
        }
        PoolInfo memory pool = poolInfo[_pid];
        uint256 holdBaseAmount = getLPTokenAmountInBaseToken(_pid, holdLPTokenAmount, pool.baseToken)
        uint256 borrowRate = borrowAmount.mul(1e9).div(holdBaseAmount);
        if(borrowRate == 0) {
            return (gather, 0, 0);
        }
        uint256 rewardsRate = holdLPRewardsAmount.mul(1e9).div(holdLPTokenAmount);
        uint256 rewardsByBorrowRate = rewardsRate.mul(borrowRate).div(1e9).mul(feerate).div(1e9);
        a0 = IERC20Upgradeable(pool.collateralToken[0]).balanceOf(_this).mul(rewardsByBorrowRate).div
        a1 = IERC20Upgradeable(pool.collateralToken[1]).balanceOf(_this).mul(rewardsByBorrowRate).div
    }

    function calcLiquidationFee(uint256 _pid, address _account)
        public view returns (address gather, uint256 baseAmount) {
        if(address(sconfig) == address(0)) return (address(0), 0);

        PoolInfo memory pool = poolInfo[_pid];
        uint256 feerate;
        (gather, feerate) = sconfig.getLiquidationFee(_this, _pid);
        baseAmount = IERC20Upgradeable(pool.baseToken).balanceOf(_this).mul(feerate).div(1e9);
    }

    function calcWithdrawRepayBorrow(uint256 _pid, address _account, uint256 _rate, uint256 _index)
        public view returns (address token, uint256 amount, bool swap, uint256 swapAmount) {

        UserInfo storage user = userInfo2[_pid][_account];
        if(_index >= user.borrowFrom.length || user.borrowFrom[_index] == address(0)) {
            return (address(0), 0, false, 0);
        }

        PoolInfo memory pool = poolInfo[_pid];
        ISafeBox borrowFrom = ISafeBox(user.borrowFrom[_index]);
        token = borrowFrom.token();

        amount = borrowFrom.pendingBorrowAmount(user.bids[_index]);
        amount = amount.add(borrowFrom.pendingBorrowRewards(user.bids[_index]));
        if(_rate < 1e9) {
            amount = amount.mul(_rate).div(1e9);
        }

        uint256 balance = IERC20Upgradeable(token).balanceOf(_this);
        if(balance < amount) {
            address swapToken = _index == 0 ? pool.collateralToken[1] : pool.collateralToken[0];
            swap = swapToken == pool.collateralToken[1];
            swapAmount = swapPoolImpl.getAmountOut(swapToken, token, amount.sub(balance));
            swapAmount = TenMath.min(swapAmount, IERC20Upgradeable(swapToken).balanceOf(_this));
```

```
        }
    }

    function getBorrowAmount(uint256 _pid, address _account, uint _index)
        public view returns (address token, uint256 amount) {
        address borrowFrom = userInfo2[_pid][_account].borrowFrom[_index];
        if(borrowFrom == address(0)) return (borrowFrom, 0);

        uint256 bid = userInfo2[_pid][_account].bids[_index];
        token = ISafeBox(borrowFrom).token();
        amount = ISafeBox(borrowFrom).pendingBorrowAmount(bid);
        amount = amount.add(ISafeBox(borrowFrom).pendingBorrowRewards(bid));
    }

    function getBorrowAmountInBaseToken(uint256 _pid, address _account)
        external view returns (uint256 amount) {
        UserInfo storage user = userInfo2[_pid][_account];
        for(uint256 i = 0; i < user.borrowFrom.length; i ++) {
            (address token, uint256 value) = getBorrowAmount(_pid, _account, i);
            if(value == 0) continue ;
            amount = amount.add(swapPoolImpl.getAmountIn(token, value, poolInfo[_pid].baseToken));
        }
    }

    function getLPTokenAmountInBaseToken(uint256 _pid, uint256 _lpTokenAmount, address _baseToken)
        public view returns (uint256 amount) {
        checkOraclePrice(_pid, false);
        amount = swapPoolImpl.getLPTokenAmountInBaseToken(poolInfo[_pid].lpToken, _lpTokenAmount, _ba
    }
}
```

StrategyV2Pair.sol

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20Upgradeable {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
```

```
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMathUpgradeable {
```

```
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
```

```
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
```

```
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}
```

```
pragma solidity >=0.6.2 <0.8.0;

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

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
```

```
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

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

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
```

```
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) intern
        require(isContract(target), "Address: static call to non-contract");

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

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

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


pragma solidity >=0.6.0 <0.8.0;


/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20Upgradeable {
    using SafeMathUpgradeable for uint256;
    using AddressUpgradeable for address;

    function safeTransfer(IERC20Upgradeable token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20Upgradeable token, address from, address to, uint256 value) inter
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20Upgradeable token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
```

```
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20Upgradeable token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}


// solhint-disable-next-line compiler-version
pragma solidity >=0.4.24 <0.8.0;


/**
 * @dev This is a base contract to aid in writing upgradeable contracts, or any kind of contract that
 * behind a proxy. Since a proxied contract can't have a constructor, it's common to move constructor
 * external initializer function, usually called `initialize`. It then becomes necessary to protect t
 * function so it can only be called once. The {initializer} modifier provided by this contract will
 *
 * TIP: To avoid leaving the proxy in an uninitialized state, the initializer function should be call
 * possible by providing the encoded function call as the `_data` argument to {UpgradeableProxy-const
 *
 * CAUTION: When used with inheritance, manual care must be taken to not invoke a parent initializer
 * that all initializers are idempotent. This is not verified automatically as constructors are by So
 */
abstract contract Initializable {

    /**
     * @dev Indicates that the contract has been initialized.
     */
    bool private _initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private _initializing;
```

```
    /**
     * @dev Modifier to protect an initializer function from being invoked twice.
     */
    modifier initializer() {
        require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is alrea

        bool isTopLevelCall = !_initializing;
        if (isTopLevelCall) {
            _initializing = true;
            _initialized = true;
        }

        _;

        if (isTopLevelCall) {
            _initializing = false;
        }
    }

    /// @dev Returns true if and only if the function is running in the constructor
    function _isConstructor() private view returns (bool) {
        return !AddressUpgradeable.isContract(address(this));
    }
}


pragma solidity >=0.6.0 <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 GSN 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 ContextUpgradeable is Initializable {
    function __Context_init() internal initializer {
        __Context_init_unchained();
    }

    function __Context_init_unchained() internal initializer {
    }
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
    uint256[50] private __gap;
}


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
```

```
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract OwnableUpgradeable is Initializable, ContextUpgradeable {
    address private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    function __Ownable_init() internal initializer {
        __Context_init_unchained();
        __Ownable_init_unchained();
    }

    function __Ownable_init_unchained() internal initializer {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
    uint256[49] private __gap;
}


pragma solidity 0.6.12;
```

```
interface ISafeBox {

    function bank() external view returns(address);

    function token() external view returns(address);

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

    function supplyRatePerBlock() external view returns (uint256);
    function borrowRatePerBlock() external view returns (uint256);

    function getBorrowInfo(uint256 _bid) external view
            returns (address owner, uint256 amount, address strategy, uint256 pid);
    function getBorrowId(address _strategy, uint256 _pid, address _account) external view returns (ui
    function getBorrowId(address _strategy, uint256 _pid, address _account, bool _add) external retur
    function getDepositTotal() external view returns (uint256);
    function getBorrowTotal() external view returns (uint256);
    // function getBorrowAmount(address _account) external view returns (uint256 value);
    function getBaseTokenPerLPToken() external view returns (uint256);

    function deposit(uint256 _value) external;
    function withdraw(uint256 _value) external;

    function emergencyWithdraw() external;
    function emergencyRepay(uint256 _bid) external;

    function borrowInfoLength() external view returns (uint256);

    function borrow(uint256 _bid, uint256 _value, address _to) external;
    function repay(uint256 _bid, uint256 _value) external;
    function claim(uint256 _tTokenAmount) external;

    function update() external;
    function mintDonate(uint256 _value) external;

    function pendingSupplyAmount(address _account) external view returns (uint256 value);
    function pendingBorrowAmount(uint256 _bid) external view returns (uint256 value);
    function pendingBorrowRewards(uint256 _bid) external view returns (uint256 value);
}

pragma solidity 0.6.12;


interface IActionPools {

    function getPoolInfo(uint256 _pid) external view
        returns (address callFrom, uint256 callId, address rewardToken);
    function mintRewards(uint256 _callId) external;
    function getPoolIndex(address _callFrom, uint256 _callId) external view returns (uint256[] memory

    function onAcionIn(uint256 _callId, address _account, uint256 _fromAmount, uint256 _toAmount) ext
    function onAcionOut(uint256 _callId, address _account, uint256 _fromAmount, uint256 _toAmount) ex
    function onAcionClaim(uint256 _callId, address _account) external;
    function onAcionEmergency(uint256 _callId, address _account) external;
    function onAcionUpdate(uint256 _callId) external;
}

pragma solidity 0.6.12;

interface ICompActionTrigger {
    function getCATPoolInfo(uint256 _pid) external view
        returns (address lpToken, uint256 allocRate, uint256 totalPoints, uint256 totalAmount);
    function getCATUserAmount(uint256 _pid, address _account) external view
        returns (uint256 points);
}
```

```
pragma solidity 0.6.12;

interface ITenBankHall {
    function makeBorrowFrom(uint256 _pid, address _account, address _debtFrom, uint256 _value) extern
}

pragma solidity 0.6.12;

interface IStrategyLink {

    event StrategyDeposit(address indexed strategy, uint256 indexed pid, address indexed user, uint25
    event StrategyBorrow(address indexed strategy, uint256 indexed pid, address indexed user, uint256
    event StrategyWithdraw(address indexed strategy, uint256 indexed pid, address indexed user, uint2
    event StrategyLiquidation(address indexed strategy, uint256 indexed pid, address indexed user, ui

    function bank() external view returns(address);
    function getSource() external view returns (string memory);
    function userInfo(uint256 _pid, address _account) external view returns (uint256,uint256,address,
    function getPoolInfo(uint256 _pid) external view  returns(address[] memory collateralToken, addre
    function getBorrowInfo(uint256 _pid, address _account) external view returns (address borrowFrom,
    function getBorrowAmount(uint256 _pid, address _account) external view returns (uint256 value);
    function getBorrowAmountInBaseToken(uint256 _pid, address _account) external view returns (uint25
    function getDepositAmount(uint256 _pid, address _account) external view returns (uint256 amount);

    function getPoolCollateralToken(uint256 _pid) external view returns (address[] memory collateralT
    function getPoollpToken(uint256 _pid) external view returns (address lpToken);
    function getBaseToken(uint256 _pid) external view returns (address baseToken);

    function poolLength() external view returns (uint256);

    function pendingRewards(uint256 _pid, address _account) external view returns (uint256 value);
    function pendingLPAmount(uint256 _pid, address _account) external view returns (uint256 value);

    // function massUpdatePools(uint256 _start, uint256 _end) external;
    function updatePool(uint256 _pid, uint256 _desirePrice, uint256 _slippage) external;

    function deposit(uint256 _pid, address _account, address _debtFrom, uint256 _bAmount, uint256 _de
    function depositLPToken(uint256 _pid, address _account, address _debtFrom, uint256 _bAmount, uint

    function withdraw(uint256 _pid, address _account, uint256 _rate, address _toToken, uint256 _desir
    function withdrawLPToken(uint256 _pid, address _account, uint256 _rate, uint256 _desirePrice, uin

    function emergencyWithdraw(uint256 _pid, address _account, uint256 _desirePrice, uint256 _slippag

    function liquidation(uint256 _pid, address _account, address _hunter, uint256 _maxDebt) external;
    function repayBorrow(uint256 _pid, address _account, uint256 _rate, bool _force) external;
}

pragma solidity 0.6.12;


interface IStrategyV2Pair is IStrategyLink {

    event AddPool(uint256 _pid, uint256 _poolId, address lpToken, address _baseToken);
    event SetMiniRewardAmount(uint256 _pid, uint256 _miniRewardAmount);
    event SetPoolImpl(address _oldv, address _new);
    event SetComponents(address _compActionPool, address _buyback, address _priceChecker, address _co
    event SetPoolConfig(uint256 _pid, string _key, uint256 _value);

    event StrategyBorrow2(address indexed strategy, uint256 indexed pid, address user, address indexe
    event StrategyRepayBorrow2(address indexed strategy, uint256 indexed pid, address user, address i
    event StrategyLiquidation2(address indexed strategy, uint256 indexed pid, address user, uint256 l

    function getBorrowInfo(uint256 _pid, address _account, uint256 _bindex)
        external view returns (address borrowFrom, uint256 bid, uint256 amount);
```

```
}
pragma solidity 0.6.12;


library TenMath {
  using SafeMathUpgradeable for uint256;

  function min(uint256 v1, uint256 v2) public pure returns (uint256 vr) {
    vr = v1 > v2 ? v2 : v1;
  }

  function safeSub(uint256 v1, uint256 v2) internal pure returns (uint256 vr) {
    vr = v1 > v2 ? v1.sub(v2) : 0;
  }

  // implementation from https://github.com/Uniswap/uniswap-lib/commit/99f3f28770640ba1bb1ff460ac7c52
  // original implementation: https://github.com/abdk-consulting/abdk-libraries-solidity/blob/master/
  function sqrt(uint256 x) internal pure returns (uint256) {
    if (x == 0) return 0;
    uint256 xx = x;
    uint256 r = 1;

    if (xx >= 0x100000000000000000000000000000000) {
      xx >>= 128;
      r <<= 64;
    }

    if (xx >= 0x10000000000000000) {
      xx >>= 64;
      r <<= 32;
    }
    if (xx >= 0x100000000) {
      xx >>= 32;
      r <<= 16;
    }
    if (xx >= 0x10000) {
      xx >>= 16;
      r <<= 8;
    }
    if (xx >= 0x100) {
      xx >>= 8;
      r <<= 4;
    }
    if (xx >= 0x10) {
      xx >>= 4;
      r <<= 2;
    }
    if (xx >= 0x8) {
      r <<= 1;
    }

    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1; // Seven iterations should be enough
    uint256 r1 = x / r;
    return (r < r1 ? r : r1);
  }
}


pragma solidity 0.6.12;
```

```
interface IPriceChecker {
    function getPriceSlippage(address _lptoken) external view returns (uint256);
    function checkLPTokenPriceLimit(address _lptoken, bool _largeType) external view returns (bool);
}

pragma solidity 0.6.12;


interface IStrategyConfig {
    // event
    event SetFeeGather(address _feeGatherOld, address _feeGather);
    event SetReservedGather(address _old, address _new);
    event SetBorrowFactor(address _strategy, uint256 _poolid, uint256 _borrowFactor);
    event SetLiquidationFactor(address _strategy, uint256 _poolid, uint256 _liquidationFactor);
    event SetFarmPoolFactor(address _strategy, uint256 _poolid, uint256 _farmPoolFactor);
    event SetDepositFee(address _strategy, uint256 _poolid, uint256 _depositFee);
    event SetWithdrawFee(address _strategy, uint256 _poolid, uint256 _withdrawFee);
    event SetRefundFee(address _strategy, uint256 _poolid, uint256 _refundFee);
    event SetClaimFee(address _strategy, uint256 _poolid, uint256 _claimFee);
    event SetLiquidationFee(address _strategy, uint256 _poolid, uint256 _liquidationFee);

    // factor
    function getBorrowFactor(address _strategy, uint256 _poolid) external view returns (uint256);
    function setBorrowFactor(address _strategy, uint256 _poolid, uint256 _borrowFactor) external;

    function getLiquidationFactor(address _strategy, uint256 _poolid) external view returns (uint256)
    function setLiquidationFactor(address _strategy, uint256 _poolid, uint256 _liquidationFactor) ext

    function getFarmPoolFactor(address _strategy, uint256 _poolid) external view returns (uint256 val
    function setFarmPoolFactor(address _strategy, uint256 _poolid, uint256 _farmPoolFactor) external;

    // fee manager
    function getDepositFee(address _strategy, uint256 _poolid) external view returns (address, uint25
    function setDepositFee(address _strategy, uint256 _poolid, uint256 _depositFee) external;

    function getWithdrawFee(address _strategy, uint256 _poolid) external view returns (address, uint2
    function setWithdrawFee(address _strategy, uint256 _poolid, uint256 _withdrawFee) external;

    function getRefundFee(address _strategy, uint256 _poolid) external view returns (address, uint256
    function setRefundFee(address _strategy, uint256 _poolid, uint256 _refundFee) external;

    function getClaimFee(address _strategy, uint256 _poolid) external view returns (address, uint256)
    function setClaimFee(address _strategy, uint256 _poolid, uint256 _claimFee) external;

    function getLiquidationFee(address _strategy, uint256 _poolid) external view returns (address, ui
    function setLiquidationFee(address _strategy, uint256 _poolid, uint256 _liquidationFee) external;
}

pragma solidity 0.6.12;

interface IStrategyV2SwapPool {

    function setStrategy(address _strategy) external;

    // get strategy
    function getName() external view returns (string memory);

    // swap functions
    function getPair(address _t0, address _t1) external view returns (address pairs);
    function getReserves(address _lpToken) external view returns (uint256 a, uint256 b);
    function getToken01(address _pairs) external view returns (address token0, address token1);
    function getAmountOut(address _tokenIn, address _tokenOut, uint256 _amountOut) external view retu
    function getAmountIn(address _tokenIn, uint256 _amountIn, address _tokenOut) external view return
    function getLPTokenAmountInBaseToken(address _lpToken, uint256 _lpTokenAmount, address _baseToken
    function swapTokenTo(address _tokenIn, uint256 _amountIn, address _tokenOut, address _toAddress)
```

```
    function optimalBorrowAmount(address _lpToken, uint256 _amount0, uint256 _amount1) external view
    function optimalDepositAmount(address lpToken, uint amtA, uint amtB) external view returns (uint

    // pool functions
    function getDepositToken(uint256 _poolId) external view returns (address lpToken);
    function getRewardToken(uint256 _poolId) external view returns (address rewardToken);
    function getPending(uint256 _poolId) external view returns (uint256 rewards);
    function deposit(uint256 _poolId, bool _autoPool) external returns (uint256 liquidity);
    function withdraw(uint256 _poolId, uint256 _liquidity, bool _autoPool) external returns (uint256
    function claim(uint256 _poolId) external returns (uint256 rewards);
    function extraRewards() external returns (address token, uint256 rewards);
}

pragma solidity 0.6.12;



contract StrategyV2Data {

    // Info of each user.
    struct UserInfo {
        uint256 lpAmount;       // deposit lptoken amount
        uint256 lpPoints;       // deposit proportion
        address[] borrowFrom;   // borrowFrom
        uint256[] bids;
    }

    // Info of each pool.
    struct PoolInfo {
        address[] collateralToken;      // collateral Token list, last must be baseToken
        address baseToken;              // baseToken can be borrowed
        address lpToken;                // lptoken to deposit
        uint256 poolId;                 // poolid for mdex pools
        uint256 lastRewardsBlock;       //
        uint256 totalPoints;            // total of user lpPoints
        uint256 totalLPAmount;          // total of user lpAmount
        uint256 totalLPReinvest;        // total of lptoken amount with totalLPAmount and reinvest re
        uint256 miniRewardAmount;       //
    }

    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping (uint256 => mapping (address => UserInfo)) public userInfo2;

    address public _bank;                // address of bank
    address public _this;
    address public helperImpl;

    IStrategyConfig public sconfig;
    mapping (uint256 => mapping (string => uint256)) public poolConfig;

    IStrategyV2SwapPool public swapPoolImpl;

    address public devAddr;
    IPriceChecker public priceChecker;
    IActionPools public compActionPool;     // address of comp action pool

    mapping(address=>bool) public whitelist;
    uint256[39] private __gap;
}


pragma solidity 0.6.12;
```

```
// farm strategy
contract StrategyV2Pair is StrategyV2Data, OwnableUpgradeable, IStrategyV2Pair, ICompActionTrigger {
    using SafeMathUpgradeable for uint256;
    using SafeERC20Upgradeable for IERC20Upgradeable;

    function initialize(address bank_, address _swapPoolImpl, address _helperImpl, address _compActio
            address _config, address _priceChecker, address _devAddr) public initializer {
        __Ownable_init();

        _bank = bank_;
        _this = address(this);
        swapPoolImpl = IStrategyV2SwapPool(_swapPoolImpl);
        swapPoolImpl.setStrategy(address(this));

        helperImpl = _helperImpl;

        compActionPool = IActionPools(_compActionPool);
        devAddr = _devAddr;
        priceChecker = IPriceChecker(_priceChecker);
        sconfig = IStrategyConfig(_config);
    }

    modifier onlyBank() {
        require(_bank == msg.sender, 'strategy only call by bank');
        _;
    }

    function bank() external override view returns(address) {
        return _bank;
    }

    function getSource() external override view returns (string memory source) {
        source = string(abi.encodePacked(swapPoolImpl.getName(), "_v2"));
    }

    function utils() external view returns(address) {
        // Compatible with old version 1
        return address(0);
    }

    function setWhitelist(address _contract, bool _enable) external onlyOwner {
        whitelist[_contract] = _enable;
    }

    function setPoolConfig(uint256 _pid, string memory _key, uint256 _value)
        external onlyOwner {
        poolConfig[_pid][_key] = _value;
        emit SetPoolConfig(_pid, _key, _value);
    }

    function setDevAddr(address _devAddr) public {
        require(devAddr == msg.sender, 'wu?');
        devAddr = _devAddr;
    }

    function poolLength() external override view returns (uint256) {
        return poolInfo.length;
    }

    // for action pool, farming rewards
    function getCATPoolInfo(uint256 _pid) external override view
        returns (address lpToken, uint256 allocRate, uint256 totalPoints, uint256 totalAmount) {
        lpToken = address(poolInfo[_pid].lpToken);
        allocRate = 5e8;
```

```
        totalPoints = poolInfo[_pid].totalPoints;
        totalAmount = poolInfo[_pid].totalLPReinvest;
    }

    function getCATUserAmount(uint256 _pid, address _account) external override view
        returns (uint256 lpPoints) {
        lpPoints = userInfo2[_pid][_account].lpPoints;
    }

    function getPoolInfo(uint256 _pid) external override view
        returns(address[] memory collateralToken, address baseToken, address lpToken,
            uint256 poolId, uint256 totalLPAmount, uint256 totalLPReinvest) {
        PoolInfo storage pool = poolInfo[_pid];
        collateralToken = pool.collateralToken;
        baseToken = address(pool.baseToken);
        lpToken = address(pool.lpToken);
        poolId = pool.poolId;
        totalLPAmount = pool.totalLPAmount;
        totalLPReinvest = pool.totalLPReinvest;
    }

    function userInfo(uint256 _pid, address _account)
        external override view returns (uint256 lpAmount, uint256 lpPoints, address borrowFrom, uint2
        // Compatible with old version 1
        UserInfo storage user = userInfo2[_pid][_account];
        lpAmount = user.lpAmount;
        lpPoints = user.lpPoints;
        borrowFrom;
        bid;
    }

    function getBorrowInfo(uint256 _pid, address _account)
        external override view returns (address borrowFrom, uint256 bid) {
        (borrowFrom, bid, ) = getBorrowInfo(_pid, _account, 0);
        if(borrowFrom == address(0)) {
            (borrowFrom, bid, ) = getBorrowInfo(_pid, _account, 1);
        }
    }

    function getBorrowInfo(uint256 _pid, address _account, uint256 _bindex)
        public override view returns (address borrowFrom, uint256 bid, uint256 amount) {
        UserInfo storage user = userInfo2[_pid][_account];
        if(_bindex >= user.borrowFrom.length) return (address(0), 0, 0);
        borrowFrom = user.borrowFrom[_bindex];
        bid = user.bids[_bindex];
        if(borrowFrom == address(0)) return (address(0), 0, 0);
        amount = ISafeBox(borrowFrom).pendingBorrowAmount(bid);
        amount = amount.add(ISafeBox(borrowFrom).pendingBorrowRewards(bid));
    }

    function pendingLPAmount(uint256 _pid, address _account) public override view returns (uint256 va
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo2[_pid][_account];
        if(pool.totalPoints <= 0) return 0;

        value = user.lpPoints.mul(pool.totalLPReinvest).div(pool.totalPoints);
        value = TenMath.min(value, pool.totalLPReinvest);
    }

    function getDepositAmount(uint256 _pid, address _account) external override view returns (uint256
        PoolInfo storage pool = poolInfo[_pid];
        uint256 lpTokenAmount = pendingLPAmount(_pid, _account);
        amount = swapPoolImpl.getLPTokenAmountInBaseToken(pool.lpToken, lpTokenAmount, pool.baseToken
    }

    function getPoolCollateralToken(uint256 _pid) external override view returns (address[] memory co
```

```
        collateralToken = poolInfo[_pid].collateralToken;
    }

    function getPoollpToken(uint256 _pid) external override view returns (address lpToken) {
        lpToken = poolInfo[_pid].lpToken;
    }

    function getBaseToken(uint256 _pid) external override view returns (address baseToken) {
        baseToken = poolInfo[_pid].baseToken;
    }

    // query user rewards
    function pendingRewards(uint256 _pid, address _account) public override view returns (uint256 val
        UserInfo storage user = userInfo2[_pid][_account];
        value = pendingLPAmount(_pid, _account);
        value = TenMath.safeSub(value, user.lpAmount);
    }

    // Add a new lp to the pool. Can only be called by the owner.
    // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
    function addPool(uint256 _poolId, address[] memory _collateralToken, address _baseToken)
        external onlyOwner {

        require(_collateralToken.length == 2);
        address lpTokenInPools = swapPoolImpl.getPair(_collateralToken[0], _collateralToken[1]);
        poolInfo.push(PoolInfo({
            collateralToken: _collateralToken,
            baseToken: _baseToken,
            lpToken: lpTokenInPools,
            poolId: _poolId,
            lastRewardsBlock: block.number,
            totalPoints: 0,
            totalLPAmount: 0,
            totalLPReinvest: 0,
            miniRewardAmount: 1e4
        }));

        uint256 pid = poolInfo.length.sub(1);
        checkAddPoolLimit(pid);

        emit AddPool(pid, _poolId, lpTokenInPools, _baseToken);
    }

    function checkAddPoolLimit(uint256 _pid) public view {
        delegateToViewImplementation(
            abi.encodeWithSignature("checkAddPoolLimit(uint256)", _pid));
    }

    function checkDepositLimit(uint256 _pid, address _account, uint256 _orginSwapRate) public view {
        delegateToViewImplementation(
            abi.encodeWithSignature("checkDepositLimit(uint256,address,uint256)", _pid, _account, _or
    }

    function checkLiquidationLimit(uint256 _pid, address _account, bool liqucheck) public view {
        delegateToViewImplementation(
            abi.encodeWithSignature("checkLiquidationLimit(uint256,address,bool)", _pid, _account, li
    }

    function checkOraclePrice(uint256 _pid, bool _large) public view {
        delegateToViewImplementation(
            abi.encodeWithSignature("checkOraclePrice(uint256,bool)", _pid, _large));
    }

    function checkBorrowLimit(uint256 _pid, address _account) public view {
        delegateToViewImplementation(
            abi.encodeWithSignature("checkBorrowLimit(uint256,address)", _pid, _account));
```

```
    }

    function calcDepositFee(uint256 _pid)
        public view returns (address feer, uint256 a0, uint256 a1) {
        bytes memory data = delegateToViewImplementation(
            abi.encodeWithSignature("calcDepositFee(uint256)", _pid));
        return abi.decode(data, (address,uint256,uint256));
    }

    function calcBorrowAmount(uint256 _pid, address _account, address _debtFrom, uint256 _bAmount)
        public view returns (uint256 bindex, uint256 amount) {
        bytes memory data = delegateToViewImplementation(
            abi.encodeWithSignature("calcBorrowAmount(uint256,address,address,uint256)",
                    _pid,_account,_debtFrom,_bAmount));
        return abi.decode(data, (uint256,uint256));
    }

    function calcRemoveLiquidity(uint256 _pid, address _account, uint256 _rate)
        public view returns (uint256 removedLPAmount, uint256 removedPoint) {
        bytes memory data = delegateToViewImplementation(
            abi.encodeWithSignature("calcRemoveLiquidity(uint256,address,uint256)",
                    _pid,_account,_rate));
        return abi.decode(data, (uint256,uint256));
    }

    function calcWithdrawFee(uint256 _pid, address _account, uint256 _rate)
        public view returns (address gather, uint256 a0, uint256 a1) {
        bytes memory data = delegateToViewImplementation(
            abi.encodeWithSignature("calcWithdrawFee(uint256,address,uint256)",
                    _pid,_account,_rate));
        return abi.decode(data, (address,uint256,uint256));
    }

    function calcLiquidationFee(uint256 _pid, address _account)
        public view returns (address gather, uint256 baseAmount) {
        bytes memory data = delegateToViewImplementation(
            abi.encodeWithSignature("calcLiquidationFee(uint256,address)",
                    _pid,_account));
        return abi.decode(data, (address,uint256));
    }

    function calcRefundFee(uint256 _pid, uint256 _rewardAmount)
        public view returns (address gather, uint256 baseAmount) {
        bytes memory data = delegateToViewImplementation(
            abi.encodeWithSignature("calcRefundFee(uint256,uint256)",
                    _pid,_rewardAmount));
        return abi.decode(data, (address,uint256));
    }

    function calcWithdrawRepayBorrow(uint256 _pid, address _account, uint256 _rate, uint256 _index)
        public view returns (address token, uint256 amount, bool swap, uint256 swapAmount) {
        bytes memory data = delegateToViewImplementation(
            abi.encodeWithSignature("calcWithdrawRepayBorrow(uint256,address,uint256,uint256)",
                    _pid,_account,_rate,_index));
        return abi.decode(data, (address,uint256,bool,uint256));
    }

    function getBorrowAmount(uint256 _pid, address _account)
        external override view returns (uint256 value) {
        // Compatible with old version 1
        value = getBorrowAmountInBaseToken(_pid, _account);
    }

    function getBorrowAmountInBaseToken(uint256 _pid, address _account)
        public override view returns (uint256 amount) {
        bytes memory data = delegateToViewImplementation(
```

```
            abi.encodeWithSignature("getBorrowAmountInBaseToken(uint256,address)",
                    _pid,_account));
        return abi.decode(data, (uint256));
    }

    // function massUpdatePools(uint256 _start, uint256 _end) external;
    function updatePool(uint256 _pid, uint256 _unused, uint256 _minOutput) external override {
        _unused;
        checkOraclePrice(_pid, true);
        uint256 lpAmount = _updatePool(_pid);
        require(lpAmount >= _minOutput, 'insufficient LP output');
    }

    function _updatePool(uint256 _pid) internal returns (uint256 lpAmount) {
        PoolInfo storage pool = poolInfo[_pid];

        if(address(compActionPool) != address(0)) {
            compActionPool.onAcionUpdate(_pid);
        }

        if(pool.lastRewardsBlock == block.number ||
            pool.totalLPReinvest <= 0) {
            pool.lastRewardsBlock = block.number;
            return 0;
        }
        pool.lastRewardsBlock = block.number;

        address refundToken = swapPoolImpl.getRewardToken(pool.poolId);
        uint256 newRewards = swapPoolImpl.getPending(pool.poolId);

        if(newRewards < pool.miniRewardAmount) {
            return 0;
        }

        newRewards = swapPoolImpl.claim(pool.poolId);

        // gather fee
        {
            (address gather, uint256 feeamount) = calcRefundFee(_pid, newRewards);
            if(feeamount > 0) IERC20Upgradeable(refundToken).safeTransfer(gather, feeamount);
            newRewards = newRewards.sub(feeamount);
        }

        // swap to basetoken
        if(refundToken != pool.collateralToken[0] && refundToken != pool.collateralToken[1]) {
            IERC20Upgradeable(refundToken).safeTransfer(address(swapPoolImpl), newRewards);
            newRewards = swapPoolImpl.swapTokenTo(refundToken, newRewards, pool.baseToken, _this);
            refundToken = pool.baseToken;
        }

        IERC20Upgradeable(refundToken).safeTransfer(address(swapPoolImpl), newRewards);
        lpAmount = swapPoolImpl.deposit(pool.poolId, true);

        pool.totalLPReinvest = pool.totalLPReinvest.add(lpAmount);
    }

    function deposit(uint256 _pid, address _account, address _debtFrom0,
                uint256 _bAmount0, uint256 _debtFrom1, uint256 _minOutput)
        external override onlyBank returns (uint256 lpAmount) {
        lpAmount = _deposit(_pid, _account, _debtFrom0, _bAmount0, address(_debtFrom1), _minOutput);
    }

    function depositLPToken(uint256 _pid, address _account, address _debtFrom0,
                uint256 _bAmount0, uint256 _debtFrom1, uint256 _minOutput)
        external override onlyBank returns (uint256 lpAmount) {
        PoolInfo storage pool = poolInfo[_pid];
```

```
        uint256 liquidity = _safeTransferAll(pool.lpToken, address(swapPoolImpl));
        swapPoolImpl.withdraw(pool.poolId, liquidity, false);
        lpAmount = _deposit(_pid, _account, _debtFrom0, _bAmount0, address(_debtFrom1), _minOutput);
    }

    function _deposit(uint256 _pid, address _account, address _debtFrom0,
        uint256 _bAmount0, address _debtFrom1, uint256 _minOutput)
        internal returns (uint256 lpAmount)  {

        require(tx.origin == _account || whitelist[_account], 'not contract');

        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo2[_pid][_account];

        // update rewards
        _updatePool(_pid);

        address token0 = pool.collateralToken[0];
        address token1 = pool.collateralToken[1];
        uint256 orginSwapRate = 0;
        {
            (uint256 res0, uint256 res1) = swapPoolImpl.getReserves(pool.lpToken);
            orginSwapRate = res0.mul(1e18).div(res1);
        }

        // borrow
        if(user.borrowFrom.length == 0) {
            user.borrowFrom = new address[](2);
            user.bids = new uint256[](2);
        }

        if(_bAmount0 > 0) {
            checkOraclePrice(_pid, false);  // Only Check the price when there is leverage
            _makeBorrow(_pid, _account, _debtFrom0, _bAmount0);
            _makeBorrow(_pid, _account, _debtFrom1, 0); // 0 = auto fit balance
        }

        // deposit fee
        {
            (address gather, uint256 bAmount0, uint256 bAmount1) = calcDepositFee(_pid);
            if(bAmount0 > 0) IERC20Upgradeable(token0).safeTransfer(gather, bAmount0);
            if(bAmount1 > 0) IERC20Upgradeable(token1).safeTransfer(gather, bAmount1);
        }

        // add liquidity and deposit
        _safeTransferAll(token0, address(swapPoolImpl));
        _safeTransferAll(token1, address(swapPoolImpl));
        lpAmount = swapPoolImpl.deposit(pool.poolId, true);

        // return cash
        _safeTransferAll(token0, _account);
        _safeTransferAll(token1, _account);

        // // booking
        uint256 lpPointsOld = user.lpPoints;
        uint256 addPoint = lpAmount;
        if(pool.totalLPReinvest > 0) {
            addPoint = lpAmount.mul(pool.totalPoints).div(pool.totalLPReinvest);
        }

        user.lpPoints = user.lpPoints.add(addPoint);
        pool.totalPoints = pool.totalPoints.add(addPoint);
        pool.totalLPReinvest = pool.totalLPReinvest.add(lpAmount);

        user.lpAmount = user.lpAmount.add(lpAmount);
        pool.totalLPAmount = pool.totalLPAmount.add(lpAmount);
```

```
        emit StrategyDeposit(_this, _pid, _account, lpAmount, _bAmount0);

        // check pool deposit limit
        require(lpAmount >= _minOutput, 'insufficient LP output');
        checkDepositLimit(_pid, _account, orginSwapRate);
        checkLiquidationLimit(_pid, _account, false);
        if(_bAmount0 > 0) {
            checkBorrowLimit(_pid, _account);
            checkOraclePrice(_pid, false);
        }

        if(address(compActionPool) != address(0) && addPoint > 0) {
            compActionPool.onAcionIn(_pid, _account, lpPointsOld, user.lpPoints);
        }
    }

    function withdraw(uint256 _pid, address _account, uint256 _rate, address _toToken, uint256 _minOu
        external override onlyBank {
        _withdraw(_pid, _account, _rate);

        // PoolInfo storage pool = poolInfo[_pid];
        uint256 outValue;
        (address token0, address token1) = (poolInfo[_pid].collateralToken[0], poolInfo[_pid].collate
        if(_toToken == address(0)) {
            uint256 outValue0 = _safeTransferAll(token0, _account);
            require(outValue0 >= _minOutputToken0, 'insufficient Token output first');
            outValue = _safeTransferAll(token1, _account);
        } else if(token0 == _toToken) {
            _swapTokenAllTo(token1, _toToken);
            outValue = _safeTransferAll(_toToken, _account);
        } else if(token1 == _toToken) {
            _swapTokenAllTo(token0, _toToken);
            outValue = _safeTransferAll(_toToken, _account);
        } else {
            require(false, 'toToken unknown');
        }
        require(outValue >= _minOutput, 'insufficient Token output');
    }

    function withdrawLPToken(uint256 _pid, address _account, uint256 _rate, uint256 _unused, uint256
        external override onlyBank {
        _withdraw(_pid, _account, _rate);

        PoolInfo storage pool = poolInfo[_pid];
        _safeTransferAll(pool.collateralToken[0], address(swapPoolImpl));
        _safeTransferAll(pool.collateralToken[1], address(swapPoolImpl));
        swapPoolImpl.deposit(pool.poolId, false);
        uint256 lpAmount = _safeTransferAll(pool.lpToken, _account);
        require(lpAmount >= _minOutput, 'insufficient LPToken output');
    }

    function _withdraw(uint256 _pid, address _account, uint256 _rate) internal {

        require(tx.origin == _account || whitelist[_account], 'not contract');

        checkLiquidationLimit(_pid, _account, false);

        // update rewards
        _updatePool(_pid);

        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo2[_pid][_account];

        bool bBorrow = (user.borrowFrom[0] != address(0) || user.borrowFrom[1] != address(0));
        if(bBorrow) checkOraclePrice(_pid, false);
```

```
        address token0 = pool.collateralToken[0];
        address token1 = pool.collateralToken[1];

        // withdraw and remove liquidity
        (uint256 removedLPAmount, uint256 removedPoint) = calcRemoveLiquidity(_pid, _account, _rate);

        if(removedLPAmount > 0) swapPoolImpl.withdraw(pool.poolId, removedLPAmount, true);

        {
            (address gather, uint256 feeAmount0, uint256 feeAmount1) = calcWithdrawFee(_pid, _account
            if(feeAmount0 > 0) IERC20Upgradeable(token0).safeTransfer(gather, feeAmount0);
            if(feeAmount1 > 0) IERC20Upgradeable(token1).safeTransfer(gather, feeAmount1);
        }

        repayBorrow(_pid, _account, _rate, true);

        uint256 withdrawLPTokenAmount = removedPoint.mul(pool.totalLPReinvest).div(pool.totalPoints);
        withdrawLPTokenAmount = TenMath.min(withdrawLPTokenAmount, pool.totalLPReinvest);

        // booking
        uint256 lpPointsOld = user.lpPoints;
        user.lpPoints = TenMath.safeSub(user.lpPoints, removedPoint);
        pool.totalPoints = TenMath.safeSub(pool.totalPoints, removedPoint);
        pool.totalLPReinvest = TenMath.safeSub(pool.totalLPReinvest, withdrawLPTokenAmount);

        user.lpAmount = TenMath.safeSub(user.lpAmount, removedLPAmount);
        pool.totalLPAmount = TenMath.safeSub(pool.totalLPAmount, removedLPAmount);

        if(bBorrow) checkOraclePrice(_pid, false);

        emit StrategyWithdraw(_this, _pid, _account, withdrawLPTokenAmount);

        if(address(compActionPool) != address(0) && removedPoint > 0) {
            compActionPool.onAcionOut(_pid, _account, lpPointsOld, user.lpPoints);
        }
    }

    function _makeBorrow(uint256 _pid, address _account, address _debtFrom, uint256 _bAmount)
        internal {
        (uint256 bindex, uint256 amount) = calcBorrowAmount(_pid, _account, _debtFrom, _bAmount);

        if(amount > 0) {
            UserInfo storage user = userInfo2[_pid][_account];
            require(user.borrowFrom[bindex] == address(0) || user.borrowFrom[bindex] == _debtFrom, 'b
            uint256 newbid = ITenBankHall(_bank).makeBorrowFrom(_pid, _account, _debtFrom, amount);
            require(newbid != 0, 'borrow new id');
            if(user.borrowFrom[bindex] == _debtFrom) {
                require(user.bids[bindex] == newbid, 'borrow newbid error');
            } else {
                require(user.borrowFrom[bindex] == address(0) && user.bids[bindex] == 0, 'borrow cann
                user.borrowFrom[bindex] = _debtFrom;
                user.bids[bindex] = newbid;
            }
            emit StrategyBorrow2(_this, _pid, _account, _debtFrom, amount);
        }
    }

    function repayBorrow(uint256 _pid, address _account, uint256 _rate, bool _force) public override
        UserInfo storage user = userInfo2[_pid][_account];
        if(user.borrowFrom[0] != address(0) || user.borrowFrom[1] != address(0)) {
            // _force as true, must repay all lending
            checkOraclePrice(_pid, _force ? false : true);
        }
        _repayBorrow(_pid, _account, _rate, 0, _force);
        _repayBorrow(_pid, _account, _rate, 1, _force);
```

```
    }

    function _repayBorrow(uint256 _pid, address _account, uint256 _rate, uint256 _index, bool _force)
        UserInfo storage user = userInfo2[_pid][_account];
        PoolInfo storage pool = poolInfo[_pid];

        address borrowFrom = user.borrowFrom[_index];
        uint256 bid = user.bids[_index];

        if(borrowFrom != address(0)) {
            ISafeBox(borrowFrom).update();
        }

        (address btoken, uint256 bAmount, bool swap, uint256 swapAmount) =
                calcWithdrawRepayBorrow(_pid, _account, _rate, _index);

        if(swapAmount > 0) {
            (address fromToken, address toToken) = swap ?
                (pool.collateralToken[1], pool.collateralToken[0]) :
                (pool.collateralToken[0], pool.collateralToken[1]);
            IERC20Upgradeable(fromToken).safeTransfer(address(swapPoolImpl), swapAmount);
            swapPoolImpl.swapTokenTo(fromToken, swapAmount, toToken, _this);
        }

        if(bAmount > 0){
            bAmount = TenMath.min(bAmount, IERC20Upgradeable(btoken).balanceOf(_this));
            IERC20Upgradeable(btoken).safeTransfer(borrowFrom, bAmount);
            ISafeBox(borrowFrom).repay(user.bids[_index], bAmount);
            emit StrategyRepayBorrow2(_this, _pid, _account, borrowFrom, bAmount);
        }

        if(_rate == 1e9 && borrowFrom != address(0)) {
            uint256 value = ISafeBox(borrowFrom).pendingBorrowAmount(bid);
            value = value.add(ISafeBox(borrowFrom).pendingBorrowRewards(bid));

            if(_force) require(value == 0, 'repayBorrow not clear');

            if(value == 0) {
                user.borrowFrom[_index] = address(0);
                user.bids[_index] = 0;
            }
        }
    }

    function emergencyWithdraw(uint256 _pid, address _account, uint256 _minOutput0, uint256 _minOutpu
        external override onlyBank {

        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo2[_pid][_account];

        bool bBorrow = (user.borrowFrom[0] != address(0) || user.borrowFrom[1] != address(0));
        if(bBorrow) checkOraclePrice(_pid, true);

        // total of deposit and reinvest
        uint256 withdrawLPTokenAmount = pendingLPAmount(_pid, _account);

        // booking
        poolInfo[_pid].totalLPReinvest = TenMath.safeSub(poolInfo[_pid].totalLPReinvest, withdrawLPTo
        poolInfo[_pid].totalPoints = TenMath.safeSub(poolInfo[_pid].totalPoints, user.lpPoints);
        poolInfo[_pid].totalLPAmount = TenMath.safeSub(poolInfo[_pid].totalLPAmount, user.lpAmount);

        user.lpPoints = 0;
        user.lpAmount = 0;

        address token0 = pool.collateralToken[0];
        address token1 = pool.collateralToken[1];
```

```
        swapPoolImpl.withdraw(pool.poolId, withdrawLPTokenAmount, true);

        repayBorrow(_pid, _account, 1e9, true);

        if(bBorrow) checkOraclePrice(_pid, true);

        require(_safeTransferAll(token0, _account) >= _minOutput0, 'insufficient output 0');
        require(_safeTransferAll(token1, _account) >= _minOutput1, 'insufficient output 1');
    }

    function liquidation(uint256 _pid, address _account, address _hunter, uint256 _maxDebt)
        external override onlyBank {

        // _maxDebt;

        UserInfo storage user = userInfo2[_pid][_account];
        PoolInfo storage pool = poolInfo[_pid];

        // update rewards
        _updatePool(_pid);

        // check liquidation limit
        checkLiquidationLimit(_pid, _account, true);
        checkOraclePrice(_pid, true);

        uint256 lpPointsOld = user.lpPoints;
        uint256 withdrawLPTokenAmount = pendingLPAmount(_pid, _account);
        // booking
        pool.totalLPAmount = TenMath.safeSub(pool.totalLPAmount, user.lpAmount);
        pool.totalLPReinvest = TenMath.safeSub(pool.totalLPReinvest, withdrawLPTokenAmount);
        pool.totalPoints = TenMath.safeSub(pool.totalPoints, user.lpPoints);

        user.lpPoints = 0;
        user.lpAmount = 0;

        if(withdrawLPTokenAmount > 0) {
            swapPoolImpl.withdraw(pool.poolId, withdrawLPTokenAmount, true);
        }

        // repay borrow
        repayBorrow(_pid, _account, 1e9, false);

        // swap all token to basetoken
        {
            address tokensell = pool.baseToken == pool.collateralToken[0] ?
                                pool.collateralToken[1] : pool.collateralToken[0];
            _swapTokenAllTo(tokensell, pool.baseToken);
        }

        // liquidation fee
        {
            (address gather, uint256 feeAmount) = calcLiquidationFee(_pid, _account);
            if(feeAmount > 0) IERC20Upgradeable(pool.baseToken).safeTransfer(gather, feeAmount);
        }

        checkOraclePrice(_pid, true);

        // send rewards to hunter
        uint256 hunterAmount = _safeTransferAll(pool.baseToken, _hunter);

        emit StrategyLiquidation2(_this, _pid, _account, withdrawLPTokenAmount, hunterAmount);

        if(address(compActionPool) != address(0) && lpPointsOld > 0) {
            compActionPool.onAcionOut(_pid, _account, lpPointsOld, 0);
        }
```

```
    }

    function makeExtraRewards() external {
        if(address(devAddr) == address(0)) {
            return ;
        }
        (address rewardsToken, uint256 value) = swapPoolImpl.extraRewards();
        if(rewardsToken == address(0) || value == 0) {
            return ;
        }
        IERC20Upgradeable(rewardsToken).safeTransfer(devAddr, value);
    }

    function _safeTransferAll(address _token, address _to)
        internal returns (uint256 value){
        value = IERC20Upgradeable(_token).balanceOf(_this);
        if(value > 0) {
            IERC20Upgradeable(_token).safeTransfer(_to, value);
        }
    }

    function _swapTokenAllTo(address _token, address _toToken)
        internal returns (uint256 value){
        uint256 amount = _safeTransferAll(_token, address(swapPoolImpl));
        if(amount > 0) {
            swapPoolImpl.swapTokenTo(_token, amount, _toToken, _this);
        }
    }

    function _delegateTo(address callee, bytes memory data) internal returns (bytes memory) {
        (bool success, bytes memory returnData) = callee.delegatecall(data);
        assembly {
            if eq(success, 0) {
                revert(add(returnData, 0x20), returndatasize())
            }
        }
        return returnData;
    }

    function delegateToImplementation(bytes memory data) public returns (bytes memory) {
        require(msg.sender == _this, 'only _this');
        return _delegateTo(helperImpl, data);
    }

    function delegateToViewImplementation(bytes memory data) public view returns (bytes memory) {
        (bool success, bytes memory returnData) = _this.staticcall(abi.encodeWithSignature("delegateT
        assembly {
            if eq(success, 0) {
                revert(add(returnData, 0x20), returndatasize())
            }
        }
        return abi.decode(returnData, (bytes));
    }
}
```

ActionCompPools.sol

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


pragma solidity >=0.6.0 <0.8.0;

/**
```

```
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20Upgradeable {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);
```

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMathUpgradeable {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
```

```
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
```

```
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
```

```
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}


pragma solidity >=0.6.2 <0.8.0;

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

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
```

```
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValu
     * with `errorMessage` as a fallback revert reason when `target` reverts.
```

```
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory er
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

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

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

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

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

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

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


pragma solidity >=0.6.0 <0.8.0;


/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
```

```
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20Upgradeable {
    using SafeMathUpgradeable for uint256;
    using AddressUpgradeable for address;

    function safeTransfer(IERC20Upgradeable token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20Upgradeable token, address from, address to, uint256 value) inter
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20Upgradeable token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20Upgradeable token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}


// solhint-disable-next-line compiler-version
pragma solidity >=0.4.24 <0.8.0;
```

```
/**
 * @dev This is a base contract to aid in writing upgradeable contracts, or any kind of contract that
 * behind a proxy. Since a proxied contract can't have a constructor, it's common to move constructor
 * external initializer function, usually called `initialize`. It then becomes necessary to protect t
 * function so it can only be called once. The {initializer} modifier provided by this contract will
 *
 * TIP: To avoid leaving the proxy in an uninitialized state, the initializer function should be call
 * possible by providing the encoded function call as the `_data` argument to {UpgradeableProxy-const
 *
 * CAUTION: When used with inheritance, manual care must be taken to not invoke a parent initializer
 * that all initializers are idempotent. This is not verified automatically as constructors are by So
 */
abstract contract Initializable {

    /**
     * @dev Indicates that the contract has been initialized.
     */
    bool private _initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private _initializing;

    /**
     * @dev Modifier to protect an initializer function from being invoked twice.
     */
    modifier initializer() {
        require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is alrea

        bool isTopLevelCall = !_initializing;
        if (isTopLevelCall) {
            _initializing = true;
            _initialized = true;
        }

        _;

        if (isTopLevelCall) {
            _initializing = false;
        }
    }

    /// @dev Returns true if and only if the function is running in the constructor
    function _isConstructor() private view returns (bool) {
        return !AddressUpgradeable.isContract(address(this));
    }
}


pragma solidity >=0.6.0 <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 GSN 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 ContextUpgradeable is Initializable {
    function __Context_init() internal initializer {
        __Context_init_unchained();
```

```
    }

    function __Context_init_unchained() internal initializer {
    }
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
    uint256[50] private __gap;
}


pragma solidity >=0.6.0 <0.8.0;

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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    function __Ownable_init() internal initializer {
        __Context_init_unchained();
        __Ownable_init_unchained();
    }

    function __Ownable_init_unchained() internal initializer {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

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

```
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
    uint256[49] private __gap;
}


pragma solidity 0.6.12;

interface ICompActionTrigger {
    function getCATPoolInfo(uint256 _pid) external view
        returns (address lpToken, uint256 allocRate, uint256 totalPoints, uint256 totalAmount);
    function getCATUserAmount(uint256 _pid, address _account) external view
        returns (uint256 points);
}

pragma solidity 0.6.12;


interface IActionPools {

    function getPoolInfo(uint256 _pid) external view
        returns (address callFrom, uint256 callId, address rewardToken);
    function mintRewards(uint256 _callId) external;
    function getPoolIndex(address _callFrom, uint256 _callId) external view returns (uint256[] memory

    function onAcionIn(uint256 _callId, address _account, uint256 _fromAmount, uint256 _toAmount) ext
    function onAcionOut(uint256 _callId, address _account, uint256 _fromAmount, uint256 _toAmount) ex
    function onAcionClaim(uint256 _callId, address _account) external;
    function onAcionEmergency(uint256 _callId, address _account) external;
    function onAcionUpdate(uint256 _callId) external;
}

pragma solidity 0.6.12;

interface IClaimFromBank {
    function claimFromBank(address _account, uint256[] memory _pidlist) external returns (uint256 val
}

pragma solidity 0.6.12;

interface IRewardsToken {
    function mint(address _account, uint256 _amount) external;
    function burn(uint256 _amount) external;

    function setMintWhitelist(address _account, bool _enabled) external;
    function checkWhitelist(address _account) external view returns (bool);
}
pragma solidity 0.6.12;
```

```
library TenMath {
  using SafeMathUpgradeable for uint256;

  function min(uint256 v1, uint256 v2) public pure returns (uint256 vr) {
    vr = v1 > v2 ? v2 : v1;
  }

  function safeSub(uint256 v1, uint256 v2) internal pure returns (uint256 vr) {
    vr = v1 > v2 ? v1.sub(v2) : 0;
  }

  // implementation from https://github.com/Uniswap/uniswap-lib/commit/99f3f28770640ba1bb1ff460ac7c52
  // original implementation: https://github.com/abdk-consulting/abdk-libraries-solidity/blob/master/
  function sqrt(uint256 x) internal pure returns (uint256) {
    if (x == 0) return 0;
    uint256 xx = x;
    uint256 r = 1;

    if (xx >= 0x100000000000000000000000000000000) {
      xx >>= 128;
      r <<= 64;
    }

    if (xx >= 0x10000000000000000) {
      xx >>= 64;
      r <<= 32;
    }
    if (xx >= 0x100000000) {
      xx >>= 32;
      r <<= 16;
    }
    if (xx >= 0x10000) {
      xx >>= 16;
      r <<= 8;
    }
    if (xx >= 0x100) {
      xx >>= 8;
      r <<= 4;
    }
    if (xx >= 0x10) {
      xx >>= 4;
      r <<= 2;
    }
    if (xx >= 0x8) {
      r <<= 1;
    }

    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1; // Seven iterations should be enough
    uint256 r1 = x / r;
    return (r < r1 ? r : r1);
  }
}


pragma solidity 0.6.12;
```

```
// Note that it's ownable and the owner wields tremendous power. The ownership
// will be transferred to a governance smart contract once Token is sufficiently
// distributed and the community can show to govern itself.
//
// Have fun reading it. Hopefully it's bug-free. God bless.
contract ActionCompPools is OwnableUpgradeable, IActionPools, IClaimFromBank {
    using SafeMathUpgradeable for uint256;
    using SafeERC20Upgradeable for IERC20Upgradeable;

    // Info of each user.
    struct UserInfo {
        uint256 rewardDebt;     // debt rewards
        uint256 rewardRemain;   // Remain rewards
    }

    // Info of each pool.
    struct PoolInfo {
        address callFrom;           // Address of trigger contract.
        uint256 callId;             // id of trigger action id, or maybe its poolid
        IERC20Upgradeable  rewardToken;        // Address of reward token address.
        uint256 rewardMaxPerBlock;  // max rewards per block.
        uint256 lastRewardBlock;    // Last block number that Token distribution occurs.
        uint256 lastRewardTotal;    // Last total amount that reward Token distribution use for calcu
        uint256 lastRewardClosed;   // Last amount that reward Token distribution.
        uint256 poolTotalRewards;   // amount will reward in contract.
        bool autoUpdate;         // auto updatepool while event
        bool autoClaim;          // auto claim while event
    }

    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that remain and debt.
    mapping (uint256 => mapping (address => UserInfo)) public userInfo;
    // index of poollist by contract and contract-call-id
    mapping (address => mapping(uint256 => uint256[])) public poolIndex;
    // total amount of each reward token
    mapping (address => uint256) public tokenTotalRewards;
    // block hacker to restricted reward
    mapping (address => uint256) public rewardRestricted;
    // event notify source, contract in whitlist
    mapping (address => bool) public eventSources;
    // mint from foxtoken, when reward token is FOXToken GRAPToken, mint it
    mapping(address => uint256) public mintTokens;
    // mint for boodev, while mint bootoken, mint a part for dev
    address public boodev;
    // allow bank proxy claim
    address public bank;

    event ActionDeposit(address indexed user, uint256 indexed pid, uint256 fromAmount, uint256 toAmou
    event ActionWithdraw(address indexed user, uint256 indexed pid, uint256 fromAmount, uint256 toAmo
    event ActionClaim(address indexed user, uint256 indexed pid, uint256 amount);
    // event ActionEmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

    event AddPool(uint256 indexed _pid, address _callFrom, uint256 _callId, address _rewardToken, uin
    event SetRewardMaxPerBlock(uint256 indexed _pid, uint256 _maxPerBlock);
    event SetRewardRestricted(address _hacker, uint256 _rate);

    function initialize(address _bank, address[] memory _mintTokens, uint256[] memory _mintFee, addre
        __Ownable_init();
        bank = _bank;
        boodev = _boodev;

        setMintTokens(_mintTokens, _mintFee);
    }

    // If the user transfers TH to contract, it will revert
```

```
    receive() external payable {
        revert();
    }

    function poolLength() external view returns (uint256) {
        return poolInfo.length;
    }

    function getPoolInfo(uint256 _pid) external override view
        returns (address callFrom, uint256 callId, address rewardToken) {
        callFrom = poolInfo[_pid].callFrom;
        callId = poolInfo[_pid].callId;
        rewardToken = address(poolInfo[_pid].rewardToken);
    }

    function getPoolIndex(address _callFrom, uint256 _callId) external override view returns (uint256
        return poolIndex[_callFrom][_callId];
    }

    // Add a new lp to the pool. Can only be called by the owner.
    function add(address _callFrom, uint256 _callId,
                address _rewardToken, uint256 _maxPerBlock) external onlyOwner {

        (address lpToken,, uint256 totalPoints,) =
                    ICompActionTrigger(_callFrom).getCATPoolInfo(_callId);
        require(lpToken != address(0) && totalPoints >= 0, 'pool not right');
        poolInfo.push(PoolInfo({
            callFrom: _callFrom,
            callId: _callId,
            rewardToken: IERC20Upgradeable(_rewardToken),
            rewardMaxPerBlock: _maxPerBlock,
            lastRewardBlock: block.number,
            lastRewardTotal: 0,
            lastRewardClosed: 0,
            poolTotalRewards: 0,
            autoUpdate: true,
            autoClaim: false
        }));

        eventSources[_callFrom] = true;
        poolIndex[_callFrom][_callId].push(poolInfo.length.sub(1));

        emit AddPool(poolInfo.length.sub(1), _callFrom, _callId, _rewardToken, _maxPerBlock);
    }

    // Set the number of reward produced by each block
    function setRewardMaxPerBlock(uint256 _pid, uint256 _maxPerBlock) external onlyOwner {
        poolInfo[_pid].rewardMaxPerBlock = _maxPerBlock;
        emit SetRewardMaxPerBlock(_pid, _maxPerBlock);
    }

    function setMintTokens(address[] memory _mintTokens, uint256[] memory _mintFee) public onlyOwner
        require(_mintTokens.length == _mintFee.length, 'length error');
        for(uint256 i = 0; i < _mintTokens.length; i ++) {
            require(IERC20Upgradeable(_mintTokens[i]).totalSupply() >= 0, '_mintTokens');
            IRewardsToken(_mintTokens[i]).checkWhitelist(address(this));
            mintTokens[_mintTokens[i]] = _mintFee[i];
            require(_mintFee[i] == 0 || _mintFee[i] >= 1e18);
        }
    }

    function setAutoUpdate(uint256 _pid, bool _set) external onlyOwner {
        poolInfo[_pid].autoUpdate = _set;
    }

    function setAutoClaim(uint256 _pid, bool _set) external onlyOwner {
```

```
        poolInfo[_pid].autoClaim = _set;
    }

    function setRewardRestricted(address _hacker, uint256 _rate) external onlyOwner {
        require(_rate <= 1e9, 'max is 1e9');
        rewardRestricted[_hacker] = _rate;
        emit SetRewardRestricted(_hacker, _rate);
    }

    function setBooDev(address _boodev) external {
        require(msg.sender == boodev, 'prev dev only');
        boodev = _boodev;
    }

    // Return reward multiplier over the given _from to _to block.
    function getBlocksReward(uint256 _pid, uint256 _from, uint256 _to) public view returns (uint256 v
        require(_from <= _to, 'getBlocksReward error');
        PoolInfo storage pool = poolInfo[_pid];
        value = pool.rewardMaxPerBlock.mul(_to.sub(_from));
        if( mintTokens[address(pool.rewardToken)] > 0) {
            return value;
        }
        if( pool.lastRewardClosed.add(value) > pool.poolTotalRewards) {
            value = pool.lastRewardClosed < pool.poolTotalRewards ?
                    pool.poolTotalRewards.sub(pool.lastRewardClosed) : 0;
        }
    }

    // View function to see pending Tokens on frontend.
    function pendingRewards(uint256 _pid, address _account) public view returns (uint256 value) {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_account];
        uint256 userPoints = ICompActionTrigger(pool.callFrom).getCATUserAmount(pool.callId, _account
        (,,uint256 poolTotalPoints,) = ICompActionTrigger(pool.callFrom).getCATPoolInfo(pool.callId);
        value = pendingRewards(_pid, _account, userPoints, poolTotalPoints);
    }

    function pendingRewards(uint256 _pid, address _account, uint256 _points, uint256 _totalPoints)
            public view returns (uint256 value) {
        UserInfo storage user = userInfo[_pid][_account];
        value = totalRewards(_pid, _points, _totalPoints)
                    .add(user.rewardRemain);
        value = TenMath.safeSub(value, user.rewardDebt);
    }

    function totalRewards(uint256 _pid, uint256 _points, uint256 _totalPoints)
            public view returns (uint256 value) {
        if(_totalPoints <= 0) {
            return 0;
        }
        PoolInfo storage pool = poolInfo[_pid];
        uint256 poolRewardTotal = pool.lastRewardTotal;
        if (block.number > pool.lastRewardBlock && _totalPoints != 0) {
            uint256 poolReward = getBlocksReward(_pid, pool.lastRewardBlock, block.number);
            poolRewardTotal = poolRewardTotal.add(poolReward);
        }
        value = _points.mul(poolRewardTotal).div(_totalPoints);
    }

    // Update reward variables for all pools. Be careful of gas spending!
    function massUpdatePools(uint256 _start, uint256 _end) public {
        if(_end <= 0) {
            _end = poolInfo.length;
        }
        for (uint256 pid = _start; pid < _end; ++pid) {
            updatePool(pid);
```

```
        }
    }

    // Update reward variables of the given pool to be up-to-date.
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return ;
        }

        (,,uint256 poolTotalAmount,) = ICompActionTrigger(pool.callFrom).getCATPoolInfo(pool.callId);
        if ( pool.rewardMaxPerBlock <= 0 ||
             poolTotalAmount <= 0) {
            pool.lastRewardBlock = block.number;
            return ;
        }

        uint256 poolReward = getBlocksReward(_pid, pool.lastRewardBlock, block.number);
        if (poolReward > 0) {
            address rewardToken = address(pool.rewardToken);
            if( mintTokens[rewardToken] > 0) {
                IRewardsToken(rewardToken).mint(address(this), poolReward);
                if(mintTokens[rewardToken] > 1e18) {
                    uint256 mintFee = poolReward.mul(mintTokens[rewardToken].sub(1e18)).div(1e18);
                    IRewardsToken(rewardToken).mint(boodev, mintFee);   // mint for dev
                }
                pool.poolTotalRewards = pool.poolTotalRewards.add(poolReward);
                tokenTotalRewards[rewardToken] = tokenTotalRewards[rewardToken].add(poolReward);
            }
            pool.lastRewardClosed = pool.lastRewardClosed.add(poolReward);
            pool.lastRewardTotal = pool.lastRewardTotal.add(poolReward);
        }
        pool.lastRewardBlock = block.number;
    }

    function onAcionIn(uint256 _callId, address _account, uint256 _fromPoints, uint256 _toPoints) ext
        if(!eventSources[msg.sender]) {
            return ;
        }
        for(uint256 u = 0; u < poolIndex[msg.sender][_callId].length; u ++) {
            uint256 pid = poolIndex[msg.sender][_callId][u];
            deposit(pid, _account, _fromPoints, _toPoints);
        }
    }

    function onAcionOut(uint256 _callId, address _account, uint256 _fromPoints, uint256 _toPoints) ex
        if(!eventSources[msg.sender]) {
            return ;
        }
        for(uint256 u = 0; u < poolIndex[msg.sender][_callId].length; u ++) {
            uint256 pid = poolIndex[msg.sender][_callId][u];
            withdraw(pid, _account, _fromPoints, _toPoints);
        }
    }

    function onAcionClaim(uint256 _callId, address _account) external override  {
        if(!eventSources[msg.sender]) {
            return ;
        }
        for(uint256 u = 0; u < poolIndex[msg.sender][_callId].length; u ++) {
            uint256 pid = poolIndex[msg.sender][_callId][u];
            if( !poolInfo[pid].autoClaim ) {
                continue;
            }
            _claim(pid, _account);
        }
```

```
    }

    function onAcionEmergency(uint256 _callId, address _account) external override  {
        _callId;
        _account;
    }

    function onAcionUpdate(uint256 _callId) external override  {
        if(!eventSources[msg.sender]) {
            return ;
        }
        for(uint256 u = 0; u < poolIndex[msg.sender][_callId].length; u ++) {
            uint256 pid = poolIndex[msg.sender][_callId][u];
            if( !poolInfo[pid].autoUpdate ) {
                continue;
            }
            updatePool(pid);
        }
    }

    function mintRewards(uint256 _pid) external override {
        updatePool(_pid);
        PoolInfo storage pool = poolInfo[_pid];
        address rewardToken = address(pool.rewardToken);
        if(mintTokens[rewardToken] > 0) {
            return ;
        }
        uint256 balance = pool.rewardToken.balanceOf(address(this));
        if ( balance > tokenTotalRewards[rewardToken]) {
            uint256 mint = balance.sub(tokenTotalRewards[rewardToken]);
            pool.poolTotalRewards = pool.poolTotalRewards.add(mint);
            tokenTotalRewards[rewardToken] = balance;
        }
    }

    // Deposit points for Token allocation.
    function deposit(uint256 _pid, address _account, uint256 _fromPoints, uint256 _toPoints) internal
        // require(_fromPoints <= _toPoints, 'deposit order error'); // for debug

        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_account];
        (,,uint256 poolTotalPoints,) = ICompActionTrigger(pool.callFrom).getCATPoolInfo(pool.callId);
        uint256 addPoint = TenMath.safeSub(_toPoints, _fromPoints);
        uint256 poolTotalPointsOld = TenMath.safeSub(poolTotalPoints, addPoint);

        user.rewardRemain = pendingRewards(_pid, _account, _fromPoints, poolTotalPointsOld);

        uint256 poolDebt = 0;
        if(poolTotalPointsOld > 0) {
            poolDebt = TenMath.safeSub(pool.lastRewardTotal.mul(poolTotalPoints).div(poolTotalPointsO
        }

        user.rewardDebt = 0;
        pool.lastRewardTotal = pool.lastRewardTotal.add(poolDebt);
        if (poolTotalPoints > 0) {
            user.rewardDebt = pool.lastRewardTotal.mul(_toPoints).div(poolTotalPoints);
        }

        emit ActionDeposit(_account, _pid, _fromPoints, _toPoints);
    }

    // Withdraw LP tokens from StarPool.
    function withdraw(uint256 _pid, address _account, uint256 _fromPoints, uint256 _toPoints) interna
        // require(_fromPoints >= _toPoints, 'deposit order error'); // debug

        PoolInfo storage pool = poolInfo[_pid];
```

```
        UserInfo storage user = userInfo[_pid][_account];
        (,,uint256 poolTotalPoints,) = ICompActionTrigger(pool.callFrom).getCATPoolInfo(pool.callId);
        uint256 removePoint = TenMath.safeSub(_fromPoints, _toPoints);
        uint256 poolTotalPointsOld = poolTotalPoints.add(removePoint);

        // recorde rewards and recalculate debt
        user.rewardRemain = pendingRewards(_pid, _account, _fromPoints, poolTotalPointsOld);

        // recalculate lastRewardTotal
        uint256 poolDebt = TenMath.safeSub(pool.lastRewardTotal,
                                pool.lastRewardTotal.mul(poolTotalPoints).div(poolTotalPointsOld));
        pool.lastRewardTotal = TenMath.safeSub(pool.lastRewardTotal, poolDebt);

        user.rewardDebt = 0;
        if (poolTotalPoints > 0) {
            user.rewardDebt = pool.lastRewardTotal.mul(_toPoints).div(poolTotalPoints);
        }

        emit ActionWithdraw(_account, _pid, _fromPoints, _toPoints);
    }

    function claimIds(uint256[] memory _pidlist) external returns (uint256 value) {
        for (uint256 piid = 0; piid < _pidlist.length; ++piid) {
            value = value.add(claim(_pidlist[piid]));
        }
    }

    function claimFromBank(address _account, uint256[] memory _pidlist) external override returns (ui
        require(bank==msg.sender, 'only call from bank');
        for (uint256 piid = 0; piid < _pidlist.length; ++piid) {
            value = value.add(_claim(_pidlist[piid], _account));
        }
    }

    function claim(uint256 _pid) public returns (uint256 value) {
        return _claim(_pid, msg.sender);
    }

    function _claim(uint256 _pid, address _account) internal returns (uint256 value) {
        updatePool(_pid);
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_account];
        value = pendingRewards(_pid, _account);

        if (value > 0) {
            // make remain booking to debt and claim out
            user.rewardRemain = 0;
            user.rewardDebt = 0;
            user.rewardDebt = pendingRewards(_pid, _account);
            // pool.lastRewardTotal; // no changed
            pool.lastRewardClosed = TenMath.safeSub(pool.lastRewardClosed, value);

            if(rewardRestricted[_account] > 0) {
                value = TenMath.safeSub(value, value.mul(rewardRestricted[_account]).div(1e9));
            }
            pool.poolTotalRewards = TenMath.safeSub(pool.poolTotalRewards, value);
            address rewardToken = address(pool.rewardToken);
            tokenTotalRewards[rewardToken] = TenMath.safeSub(tokenTotalRewards[rewardToken], value);

            value = safeTokenTransfer(pool.rewardToken, _account, value);
        }

        emit ActionClaim(_account, _pid, value);
    }

    // Withdraw without caring about rewards. EMERGENCY ONLY.
```

```
    function emergencyWithdraw(uint256 _pid, address _account) internal {
        _pid;
        _account;
    }

    // Safe Token transfer function, just in case if rounding error causes pool to not have enough To
    function safeTokenTransfer(IERC20Upgradeable _token, address _to, uint256 _amount) internal retur
        uint256 balance = _token.balanceOf(address(this));
        value = _amount > balance ? balance : _amount;
        if ( value > 0 ) {
            _token.safeTransfer(_to, value);
            value =  TenMath.safeSub(balance, _token.balanceOf(address(this)));
        }
    }
}
```

TenBankHall.sol

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20Upgradeable {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
```

```
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMathUpgradeable {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
```

```
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
```

```
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
```

```
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}


pragma solidity >=0.6.2 <0.8.0;

/**
 * @dev Collection of functions related to the address type
 */
library AddressUpgradeable {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
```

```
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
```

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

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

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

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

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

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

    function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) pri
        if (success) {
            return returndata;
```

```
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

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


pragma solidity >=0.6.0 <0.8.0;


/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20Upgradeable {
    using SafeMathUpgradeable for uint256;
    using AddressUpgradeable for address;

    function safeTransfer(IERC20Upgradeable token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20Upgradeable token, address from, address to, uint256 value) inter
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20Upgradeable token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
```

```
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20Upgradeable token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}


// solhint-disable-next-line compiler-version
pragma solidity >=0.4.24 <0.8.0;


/**
 * @dev This is a base contract to aid in writing upgradeable contracts, or any kind of contract that
 * behind a proxy. Since a proxied contract can't have a constructor, it's common to move constructor
 * external initializer function, usually called `initialize`. It then becomes necessary to protect t
 * function so it can only be called once. The {initializer} modifier provided by this contract will
 *
 * TIP: To avoid leaving the proxy in an uninitialized state, the initializer function should be call
 * possible by providing the encoded function call as the `_data` argument to {UpgradeableProxy-const
 *
 * CAUTION: When used with inheritance, manual care must be taken to not invoke a parent initializer
 * that all initializers are idempotent. This is not verified automatically as constructors are by So
 */
abstract contract Initializable {

    /**
     * @dev Indicates that the contract has been initialized.
     */
    bool private _initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private _initializing;

    /**
     * @dev Modifier to protect an initializer function from being invoked twice.
     */
    modifier initializer() {
        require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is alrea

        bool isTopLevelCall = !_initializing;
        if (isTopLevelCall) {
            _initializing = true;
            _initialized = true;
        }

        _;
```

```
        if (isTopLevelCall) {
            _initializing = false;
        }
    }

    /// @dev Returns true if and only if the function is running in the constructor
    function _isConstructor() private view returns (bool) {
        return !AddressUpgradeable.isContract(address(this));
    }
}


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Contract module that helps prevent reentrant calls to a function.
 *
 * Inheriting from `ReentrancyGuard` will make the {nonReentrant} modifier
 * available, which can be applied to functions to make sure there are no nested
 * (reentrant) calls to them.
 *
 * Note that because there is a single `nonReentrant` guard, functions marked as
 * `nonReentrant` may not call one another. This can be worked around by making
 * those functions `private`, and then adding `external` `nonReentrant` entry
 * points to them.
 *
 * TIP: If you would like to learn more about reentrancy and alternative ways
 * to protect against it, check out our blog post
 * https://blog.openzeppelin.com/reentrancy-after-istanbul/[Reentrancy After Istanbul].
 */
abstract contract ReentrancyGuardUpgradeable is Initializable {
    // Booleans are more expensive than uint256 or any type that takes up a full
    // word because each write operation emits an extra SLOAD to first read the
    // slot's contents, replace the bits taken up by the boolean, and then write
    // back. This is the compiler's defense against contract upgrades and
    // pointer aliasing, and it cannot be disabled.

    // The values being non-zero value makes deployment a bit more expensive,
    // but in exchange the refund on every call to nonReentrant will be lower in
    // amount. Since refunds are capped to a percentage of the total
    // transaction's gas, it is best to keep them low in cases like this one, to
    // increase the likelihood of the full refund coming into effect.
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    function __ReentrancyGuard_init() internal initializer {
        __ReentrancyGuard_init_unchained();
    }

    function __ReentrancyGuard_init_unchained() internal initializer {
        _status = _NOT_ENTERED;
    }

    /**
     * @dev Prevents a contract from calling itself, directly or indirectly.
     * Calling a `nonReentrant` function from another `nonReentrant`
     * function is not supported. It is possible to prevent this from happening
     * by making the `nonReentrant` function external, and make it call a
     * `private` function that does the actual work.
     */
    modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");
```

```
        // Any calls to nonReentrant after this point will fail
        _status = _ENTERED;

        _;

        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
    uint256[49] private __gap;
}


pragma solidity >=0.6.0 <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 GSN 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 ContextUpgradeable is Initializable {
    function __Context_init() internal initializer {
        __Context_init_unchained();
    }

    function __Context_init_unchained() internal initializer {
    }
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
    uint256[50] private __gap;
}


pragma solidity >=0.6.0 <0.8.0;

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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
```

```
    function __Ownable_init() internal initializer {
        __Context_init_unchained();
        __Ownable_init_unchained();
    }

    function __Ownable_init_unchained() internal initializer {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
    uint256[49] private __gap;
}


pragma solidity 0.6.12;

interface IStrategyLink {

    event StrategyDeposit(address indexed strategy, uint256 indexed pid, address indexed user, uint25
    event StrategyBorrow(address indexed strategy, uint256 indexed pid, address indexed user, uint256
    event StrategyWithdraw(address indexed strategy, uint256 indexed pid, address indexed user, uint2
    event StrategyLiquidation(address indexed strategy, uint256 indexed pid, address indexed user, ui

    function bank() external view returns(address);
    function getSource() external view returns (string memory);
    function userInfo(uint256 _pid, address _account) external view returns (uint256,uint256,address,
    function getPoolInfo(uint256 _pid) external view  returns(address[] memory collateralToken, addre
    function getBorrowInfo(uint256 _pid, address _account) external view returns (address borrowFrom,
    function getBorrowAmount(uint256 _pid, address _account) external view returns (uint256 value);
```

```
    function getBorrowAmountInBaseToken(uint256 _pid, address _account) external view returns (uint25
    function getDepositAmount(uint256 _pid, address _account) external view returns (uint256 amount);

    function getPoolCollateralToken(uint256 _pid) external view returns (address[] memory collateralT
    function getPoollpToken(uint256 _pid) external view returns (address lpToken);
    function getBaseToken(uint256 _pid) external view returns (address baseToken);

    function poolLength() external view returns (uint256);

    function pendingRewards(uint256 _pid, address _account) external view returns (uint256 value);
    function pendingLPAmount(uint256 _pid, address _account) external view returns (uint256 value);

    // function massUpdatePools(uint256 _start, uint256 _end) external;
    function updatePool(uint256 _pid, uint256 _desirePrice, uint256 _slippage) external;

    function deposit(uint256 _pid, address _account, address _debtFrom, uint256 _bAmount, uint256 _de
    function depositLPToken(uint256 _pid, address _account, address _debtFrom, uint256 _bAmount, uint

    function withdraw(uint256 _pid, address _account, uint256 _rate, address _toToken, uint256 _desir
    function withdrawLPToken(uint256 _pid, address _account, uint256 _rate, uint256 _desirePrice, uin

    function emergencyWithdraw(uint256 _pid, address _account, uint256 _desirePrice, uint256 _slippag

    function liquidation(uint256 _pid, address _account, address _hunter, uint256 _maxDebt) external;
    function repayBorrow(uint256 _pid, address _account, uint256 _rate, bool _force) external;
}

pragma solidity 0.6.12;

interface ITenBankHall {
    function makeBorrowFrom(uint256 _pid, address _account, address _debtFrom, uint256 _value) extern
}

pragma solidity 0.6.12;

interface ISafeBox {

    function bank() external view returns(address);

    function token() external view returns(address);

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

    function supplyRatePerBlock() external view returns (uint256);
    function borrowRatePerBlock() external view returns (uint256);

    function getBorrowInfo(uint256 _bid) external view
            returns (address owner, uint256 amount, address strategy, uint256 pid);
    function getBorrowId(address _strategy, uint256 _pid, address _account) external view returns (ui
    function getBorrowId(address _strategy, uint256 _pid, address _account, bool _add) external retur
    function getDepositTotal() external view returns (uint256);
    function getBorrowTotal() external view returns (uint256);
    // function getBorrowAmount(address _account) external view returns (uint256 value);
    function getBaseTokenPerLPToken() external view returns (uint256);

    function deposit(uint256 _value) external;
    function withdraw(uint256 _value) external;

    function emergencyWithdraw() external;
    function emergencyRepay(uint256 _bid) external;

    function borrowInfoLength() external view returns (uint256);

    function borrow(uint256 _bid, uint256 _value, address _to) external;
    function repay(uint256 _bid, uint256 _value) external;
    function claim(uint256 _tTokenAmount) external;
```

```
    function update() external;
    function mintDonate(uint256 _value) external;

    function pendingSupplyAmount(address _account) external view returns (uint256 value);
    function pendingBorrowAmount(uint256 _bid) external view returns (uint256 value);
    function pendingBorrowRewards(uint256 _bid) external view returns (uint256 value);
}

pragma solidity 0.6.12;

interface IClaimFromBank {
    function claimFromBank(address _account, uint256[] memory _pidlist) external returns (uint256 val
}

pragma solidity 0.6.12;



// TenBank bank
contract TenBankHall is OwnableUpgradeable, ITenBankHall, ReentrancyGuardUpgradeable {
    using SafeMathUpgradeable for uint256;
    using SafeERC20Upgradeable for IERC20Upgradeable;

    struct StrategyInfo {
        bool isListed;          // if enabled, it will access
        IStrategyLink iLink;    // strategy interface
        uint256 pid;            // strategy poolid, multiple strategys pools
    }

    // safebox manager
    ISafeBox[] public boxInfo;
    mapping(address => uint256) public boxIndex;
    mapping(uint256 => bool) public boxlisted;

    // strategyinfo manager
    StrategyInfo[] public strategyInfo;
    mapping(address => mapping(uint256 => uint256)) public strategyIndex; // strategy + pid => strate

    // actionpools claim
    IClaimFromBank[] public poolClaim;

    // blacklist
    mapping(address => bool) public blacklist;
    mapping(uint256 => bool) public emergencyEnabled;

    event AddBox(uint256 indexed _boxid, address _safebox);
    event AddStrategy(uint256 indexed _sid, address indexed _strategylink, uint256 indexed _pid, bool
    event SetBlacklist(address indexed _account, bool _newset);
    event SetEmergencyEnabled(uint256 indexed _sid, bool _newset);
    event SetBoxListed(uint256 indexed _boxid, bool _listed);

    function initialize() public initializer {
        __Ownable_init();
        __ReentrancyGuard_init();
    }

    // blacklist manager
    function setBlacklist(address _account, bool _newset) external onlyOwner {
        blacklist[_account] = _newset;
        if(_newset) {
            require(isContract(_account), 'contract address only');
        }
        emit SetBlacklist(_account, _newset);
    }
```

```
    function isContract(address addr) internal returns (bool) {
        uint size;
        assembly { size := extcodesize(addr) }
        return size > 0;
    }

    function setEmergencyEnabled(uint256 _sid, bool _newset) external onlyOwner {
        emergencyEnabled[_sid] = _newset;
        emit SetEmergencyEnabled(_sid, _newset);
    }

    function claimLength() external view returns (uint256) {
        return poolClaim.length;
    }

    function addClaimPool(address _poolClaim) external onlyOwner {
        poolClaim.push(IClaimFromBank(_poolClaim));
    }

    // box manager
    function boxesLength() external view returns (uint256) {
        return boxInfo.length;
    }

    function addBox(address _safebox) external onlyOwner {
        require(boxIndex[_safebox] == 0, 'add once only');
        boxInfo.push(ISafeBox(_safebox));
        uint256 boxid = boxInfo.length.sub(1);
        boxlisted[boxid] = true;
        boxIndex[_safebox] = boxid;
        emit AddBox(boxid, _safebox);
        require(ISafeBox(_safebox).bank() == address(this), 'bank not me?');
    }

    function setBoxListed(uint256 _boxid, bool _listed) external onlyOwner {
        boxlisted[_boxid] = _listed;
        emit SetBoxListed(_boxid, _listed);
    }

    // Strategy manager
    function strategyInfoLength() external view returns (uint256 length) {
        length = strategyInfo.length;
    }

    function strategyIsListed(uint256 _sid) external view returns (bool) {
        return strategyInfo[_sid].isListed;
    }

    function setStrategyListed(uint256 _sid, bool _listed) external onlyOwner {
        strategyInfo[_sid].isListed = _listed;
    }

    function addStrategy(address _strategylink, uint256 _pid, bool _blisted) external onlyOwner {
        require(IStrategyLink(_strategylink).poolLength() > _pid, 'not strategy pid');
        strategyInfo.push(StrategyInfo(
            _blisted,
            IStrategyLink(_strategylink),
            _pid));
        strategyIndex[_strategylink][_pid] = strategyInfo.length.sub(1);
        emit AddStrategy(strategyIndex[_strategylink][_pid], _strategylink, _pid, _blisted);
        require(IStrategyLink(_strategylink).bank() == address(this), 'bank not me?');
    }

    function depositLPToken(uint256 _sid, uint256 _amount, uint256 _bid, uint256 _bAmount, uint256 _d
            public nonReentrant returns (uint256 lpAmount) {
        require(strategyInfo[_sid].isListed, 'not listed');
```

```
        require(!blacklist[msg.sender], 'address in blacklist');

        address lpToken = strategyInfo[_sid].iLink.getPoollpToken(strategyInfo[_sid].pid);
        IERC20Upgradeable(lpToken).safeTransferFrom(msg.sender, address(strategyInfo[_sid].iLink), _a

        address boxitem = address(0);
        if(_bAmount > 0) {
            boxitem = address(boxInfo[_bid]);
        }
        return strategyInfo[_sid].iLink.depositLPToken(strategyInfo[_sid].pid, msg.sender, boxitem, _
    }

    function deposit(uint256 _sid, uint256[] memory _amount, uint256 _bid, uint256 _bAmount, uint256
            public nonReentrant returns (uint256 lpAmount) {
        require(strategyInfo[_sid].isListed, 'not listed');
        require(!blacklist[msg.sender], 'address in blacklist');

        address[] memory collateralToken = strategyInfo[_sid].iLink.getPoolCollateralToken(strategyIn
        require(collateralToken.length == _amount.length, '_amount length error');

        for(uint256 u = 0; u < collateralToken.length; u ++) {
            if(_amount[u] > 0) {
                IERC20Upgradeable(collateralToken[u]).safeTransferFrom(msg.sender, address(strategyIn
            }
        }

        address boxitem = address(0);
        if(_bAmount > 0) {
            boxitem = address(boxInfo[_bid]);
        }
        return strategyInfo[_sid].iLink.deposit(strategyInfo[_sid].pid, msg.sender, boxitem, _bAmount
    }

    function withdrawLPToken(uint256 _sid, uint256 _rate, uint256 _desirePrice, uint256 _slippage) ex
        return strategyInfo[_sid].iLink.withdrawLPToken(strategyInfo[_sid].pid, msg.sender, _rate, _d
    }

    function withdraw(uint256 _sid, uint256 _rate, address _toToken, uint256 _desirePrice, uint256 _s
        return strategyInfo[_sid].iLink.withdraw(strategyInfo[_sid].pid, msg.sender, _rate, _toToken,
    }

    function withdrawLPTokenAndClaim(uint256 _sid, uint256 _rate,
                                     uint256 _desirePrice, uint256 _slippage,
                                     uint256 _poolClaimId, uint256[] memory _pidlist) external nonReen
        strategyInfo[_sid].iLink.withdrawLPToken(strategyInfo[_sid].pid, msg.sender, _rate, _desirePr
        if(_pidlist.length > 0) {
            poolClaim[_poolClaimId].claimFromBank(msg.sender, _pidlist);
        }
    }

    function withdrawAndClaim(uint256 _sid, uint256 _rate, address _toToken,
                              uint256 _desirePrice, uint256 _slippage,
                              uint256 _poolClaimId, uint256[] memory _pidlist) external nonReentran
        strategyInfo[_sid].iLink.withdraw(strategyInfo[_sid].pid, msg.sender, _rate, _toToken, _desir
        if(_pidlist.length > 0) {
            poolClaim[_poolClaimId].claimFromBank(msg.sender, _pidlist);
        }
    }

    function claim(uint256 _poolClaimId, uint256[] memory _pidlist) external nonReentrant {
        poolClaim[_poolClaimId].claimFromBank(msg.sender, _pidlist);
    }

    function emergencyWithdraw(uint256 _sid, uint256 _desirePrice, uint256 _slippage) external nonRee
        require(emergencyEnabled[_sid], 'emergency not enabled');
        return strategyInfo[_sid].iLink.emergencyWithdraw(strategyInfo[_sid].pid, msg.sender, _desire
```

```
    }

    function liquidation(uint256 _sid, address _account, uint256 _maxDebt) external nonReentrant {
        uint256 pid = strategyInfo[_sid].pid;
        if(_maxDebt > 0) {
            (address borrowFrom,) = IStrategyLink(strategyInfo[_sid].iLink).getBorrowInfo(pid, _accou
            address borrowToken = ISafeBox(borrowFrom).token();
            IERC20Upgradeable(borrowToken).safeTransferFrom(msg.sender, address(strategyInfo[_sid].iL
        }
        strategyInfo[_sid].iLink.liquidation(pid, _account, msg.sender, _maxDebt);
    }

    function getBorrowAmount(uint256 _sid, address _account) external view returns (uint256 value) {
        value = strategyInfo[_sid].iLink.getBorrowAmount(strategyInfo[_sid].pid, _account);
    }

    function getDepositAmount(uint256 _sid, address _account) external view returns (uint256 value) {
        value = strategyInfo[_sid].iLink.getDepositAmount(strategyInfo[_sid].pid, _account);
    }

    function makeBorrowFrom(uint256 _pid, address _account, address _borrowFrom, uint256 _value)
        external override returns (uint256 bid) {
        // borrow from bank will check contract authority
        uint256 sid = strategyIndex[msg.sender][_pid];
        require(address(strategyInfo[sid].iLink) == msg.sender, 'only call from strategy');
        bid = ISafeBox(_borrowFrom).getBorrowId(msg.sender, _pid, _account, true);
        require(bid > 0, 'bid go run');
        ISafeBox(_borrowFrom).borrow(bid, _value, msg.sender);
    }

    receive() external payable {
        revert();
    }
}
```

StrategyConfig.sol

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
```

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
```

```
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
```

```
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
```

```
    }
}


pragma solidity >=0.6.2 <0.8.0;

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

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
```

```
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

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

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

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

```
    }

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

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

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

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


pragma solidity >=0.6.0 <0.8.0;


/**
 * @title SafeERC20
```

```
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}
```

```
pragma solidity >=0.6.0 <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 GSN 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 payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
}


pragma solidity >=0.6.0 <0.8.0;

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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
```

```
    }

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

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


pragma solidity 0.6.12;


interface IStrategyConfig {
    // event
    event SetFeeGather(address _feeGatherOld, address _feeGather);
    event SetReservedGather(address _old, address _new);
    event SetBorrowFactor(address _strategy, uint256 _poolid, uint256 _borrowFactor);
    event SetLiquidationFactor(address _strategy, uint256 _poolid, uint256 _liquidationFactor);
    event SetFarmPoolFactor(address _strategy, uint256 _poolid, uint256 _farmPoolFactor);
    event SetDepositFee(address _strategy, uint256 _poolid, uint256 _depositFee);
    event SetWithdrawFee(address _strategy, uint256 _poolid, uint256 _withdrawFee);
    event SetRefundFee(address _strategy, uint256 _poolid, uint256 _refundFee);
    event SetClaimFee(address _strategy, uint256 _poolid, uint256 _claimFee);
    event SetLiquidationFee(address _strategy, uint256 _poolid, uint256 _liquidationFee);

    // factor
    function getBorrowFactor(address _strategy, uint256 _poolid) external view returns (uint256);
    function setBorrowFactor(address _strategy, uint256 _poolid, uint256 _borrowFactor) external;

    function getLiquidationFactor(address _strategy, uint256 _poolid) external view returns (uint256)
    function setLiquidationFactor(address _strategy, uint256 _poolid, uint256 _liquidationFactor) ext

    function getFarmPoolFactor(address _strategy, uint256 _poolid) external view returns (uint256 val
    function setFarmPoolFactor(address _strategy, uint256 _poolid, uint256 _farmPoolFactor) external;

    // fee manager
    function getDepositFee(address _strategy, uint256 _poolid) external view returns (address, uint25
    function setDepositFee(address _strategy, uint256 _poolid, uint256 _depositFee) external;

    function getWithdrawFee(address _strategy, uint256 _poolid) external view returns (address, uint2
    function setWithdrawFee(address _strategy, uint256 _poolid, uint256 _withdrawFee) external;

    function getRefundFee(address _strategy, uint256 _poolid) external view returns (address, uint256
    function setRefundFee(address _strategy, uint256 _poolid, uint256 _refundFee) external;

    function getClaimFee(address _strategy, uint256 _poolid) external view returns (address, uint256)
    function setClaimFee(address _strategy, uint256 _poolid, uint256 _claimFee) external;

    function getLiquidationFee(address _strategy, uint256 _poolid) external view returns (address, ui
```

```
    function setLiquidationFee(address _strategy, uint256 _poolid, uint256 _liquidationFee) external;
}

pragma solidity 0.6.12;



// fund fee processing
// some functions of strategy
contract StrategyConfig is Ownable, IStrategyConfig {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    address public feeGather;           // fee gather
    address public reservedGather;      // reserved gather

    mapping (address => mapping(uint256=>uint256) ) public borrowFactor;
    mapping (address => mapping(uint256=>uint256) ) public liquidationFactor;
    mapping (address => mapping(uint256=>uint256) ) public farmPoolFactor;

    mapping (address => mapping(uint256=>uint256) ) public depositFee;  // deposit platform fee
    mapping (address => mapping(uint256=>uint256) ) public refundFee;   // reinvestment platform fee
    mapping (address => mapping(uint256=>uint256) ) public withdrawFee; // withdraw rewards platform
    mapping (address => mapping(uint256=>uint256) ) public claimFee;    // claim fee - no used
    mapping (address => mapping(uint256=>uint256) ) public liquidationFee;  // the hunter fee

    constructor() public {
        feeGather = msg.sender;
        reservedGather = msg.sender;
    }

    function setFeeGather(address _feeGather) external onlyOwner {
        emit SetFeeGather(feeGather, _feeGather);
        feeGather = _feeGather;
    }

    function setReservedGather(address _reservedGather) external onlyOwner {
        emit SetReservedGather(reservedGather, _reservedGather);
        reservedGather = _reservedGather;
    }

    // Lending burst
    function getBorrowFactor(address _strategy, uint256 _poolid) public override view returns (uint25
        value = borrowFactor[_strategy][_poolid];
    }

    function checkBorrowAndLiquidation(address _strategy, uint256 _poolid) internal returns (bool bok
        uint256 v = getBorrowFactor(_strategy, _poolid);
        if(v <= 0) {
            return true;
        }
        // MaxBorrowAmount = DepositAmount * BorrowFactor
        // MaxBorrowAmount / (DepositAmount + MaxBorrowAmount) * 100.5% < LiquidationFactor
        bok = v.mul(1005e6).div(v.add(1e9)) < getLiquidationFactor(_strategy, _poolid);
    }

    function setBorrowFactor(address _strategy, uint256 _poolid, uint256 _borrowFactor) external over
        borrowFactor[_strategy][_poolid] = _borrowFactor;
        emit SetBorrowFactor(_strategy, _poolid, _borrowFactor);
        require(checkBorrowAndLiquidation(_strategy, _poolid), 'set error');
    }

    function getLiquidationFactor(address _strategy, uint256 _poolid) public override view returns (u
        value = liquidationFactor[_strategy][_poolid];
        if(value <= 0) {
            value = 8e8;  // 80% for default , 100% will be liquidation
```

```
        }
    }

    function setLiquidationFactor(address _strategy, uint256 _poolid, uint256 _liquidationFactor) ext
        require(_liquidationFactor >= 2e8, 'too lower');
        liquidationFactor[_strategy][_poolid] = _liquidationFactor;
        emit SetLiquidationFactor(_strategy, _poolid, _liquidationFactor);
        require(checkBorrowAndLiquidation(_strategy, _poolid), 'set error');
    }

    function getFarmPoolFactor(address _strategy, uint256 _poolid) external override view returns (ui
        value = farmPoolFactor[_strategy][_poolid];
        // == 0 no limit and > 0 limit by lptoken amount
    }

    function setFarmPoolFactor(address _strategy, uint256 _poolid, uint256 _farmPoolFactor) external
        farmPoolFactor[_strategy][_poolid] = _farmPoolFactor;
        emit SetFarmPoolFactor(_strategy, _poolid, _farmPoolFactor);
    }

    // fee config
    function getDepositFee(address _strategy, uint256 _poolid) external override view returns (addres
        a = feeGather;
        b = depositFee[_strategy][_poolid];
    }

    function setDepositFee(address _strategy, uint256 _poolid, uint256 _depositFee) external override
        depositFee[_strategy][_poolid] = _depositFee;
        emit SetDepositFee(_strategy, _poolid, _depositFee);
    }

    function getWithdrawFee(address _strategy, uint256 _poolid) external override view returns (addre
        a = feeGather;
        b = withdrawFee[_strategy][_poolid];
    }

    function setWithdrawFee(address _strategy, uint256 _poolid, uint256 _withdrawFee) external overri
        withdrawFee[_strategy][_poolid] = _withdrawFee;
        emit SetWithdrawFee(_strategy, _poolid, _withdrawFee);
    }

    function getRefundFee(address _strategy, uint256 _poolid) external override view returns (address
        a = feeGather;
        b = refundFee[_strategy][_poolid];
    }

    function setRefundFee(address _strategy, uint256 _poolid, uint256 _refundFee) external override o
        refundFee[_strategy][_poolid] = _refundFee;
        emit SetRefundFee(_strategy, _poolid, _refundFee);
    }

    function getClaimFee(address _strategy, uint256 _poolid) external override view returns (address
        a = feeGather;
        b = claimFee[_strategy][_poolid];
    }

    function setClaimFee(address _strategy, uint256 _poolid, uint256 _claimFee) external override onl
        claimFee[_strategy][_poolid] = _claimFee;
        emit SetClaimFee(_strategy, _poolid, _claimFee);
    }

    function getLiquidationFee(address _strategy, uint256 _poolid) external override view returns (ad
        a = reservedGather;
        b = liquidationFee[_strategy][_poolid];
    }
```

```
    function setLiquidationFee(address _strategy, uint256 _poolid, uint256 _liquidationFee) external
        liquidationFee[_strategy][_poolid] = _liquidationFee;
        emit SetLiquidationFee(_strategy, _poolid, _liquidationFee);
    }
}
```

PriceCheckerLPToken::TransparentUpgradeableProxy.sol

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

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev This abstract contract provides a fallback function that delegates all calls to an
 * instruction `delegatecall`. We refer to the second contract as the
 * be specified by overriding the virtual {_implementation} function.
 *
 * Additionally, delegation to the implementation can be triggered manually through
 * different contract through the {_delegate} function.
 *
 * The success and return data of the delegated call will b
 */
abstract contract Proxy {
    /**
     * @dev Delegates the current call to `implementation`.
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _delegate(address implementation) internal virtual {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
            // delegatecall returns 0 on error.
            case 0 { revert(0, returndatasize()) }
            default { return(0, returndatasize()) }
        }
    }

    /**
     * @dev This is a virtual function that should be override
     * and {_fallback} should delegate.
     */
    function _implementation() internal view virtual returns (address);
```

```
    /**
     * @dev Delegates the current call to the address return
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _fallback() internal virtual {
        _beforeFallback();
        _delegate(_implementation());
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_impleme
     * function in the contract matches the call data.
     */
    fallback () external payable virtual {
        _fallback();
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_impleme
     * is empty.
     */
    receive () external payable virtual {
        _fallback();
    }

    /**
     * @dev Hook that is called before falling back to the implementation. Can happen a
     * call, or as part of the Solidity `fallback` or `receive` functions.
     *
     * If overriden should call `super._beforeFallback()`.
     */
    function _beforeFallback() internal virtual {
    }
}


pragma solidity >=0.6.2 <0.8.0;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a
     *
     * Among others, `isContract` will return false for the fol
     * types of addresses:
     *
     * - an externally-owned account
```

```
 * - a contract in construction
 * - an address where a contract will
 * - an address where a contract lived, but
 * ====
 */
  function isContract(address account) internal view returns (bool) {
      // This method relies on extcodesize, which returns 0 for contracts in
      // construction, since the code is only stored at the end of the
      // constructor execution.

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

              /**
 * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
 * `recipient`, forwarding all available gas and reverting on errors.
 *
 * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
 * of certain opcodes, possibly making contracts go over the 2300 gas limit
 * imposed by `transfer`, making them unable to receive funds via
 * `transfer`. {sendValue} removes this limitation.
 *
 * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more
 *
 * IMPORTANT: because control is transferred to `recipient`, care must be
 * taken to not create reentrancy vulnerabilities. Consider using
 * {ReentrancyGuard} or the
 * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use- the -che
 */
  function sendValue(address payable recipient, uint256 amount) internal {
      require(address(this).balance >= amount, "Address: insufficient balance");

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

              /**
 * @dev Performs a Solidity function call using a low lev
 * plain`call` is an unsafe replacement for a function call:
 * function instead.
 *
 * If `target` reverts with a revert reason, it is bubbled up by this
 * function ( like regular Solidity function calls).
 *
 * Returns the raw returned data. To convert to the expe
 * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding
 *
 * Requirements:
 *
 * - `target` must be a contract.
 * - calling `target` with `data` must not revert.
```

```
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

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

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

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

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

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

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

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

```
pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev This contract implements an upgradeable proxy. It is upgradeable because cal
 * implementation address that can be changed. This address is stored in storage in the
 * https://eips.ethereum.org/EIPS/eip-1967[EIP1967], so that it doesn't
 * implementation behind the proxy.
 *
 * Upgradeability is only provided internally through {_upgradeTo}. For an externally up
 * {TransparentUpgradeableProxy}.
 */
contract UpgradeableProxy is Proxy {
    /**
     * @dev Initializes the upgradeable proxy with an initial
     *
     * If `_data` is nonempty, it's used as data in a delegate call to `_logic`. This
     * function call, and allows initializating the storage of the
     */
    constructor(address _logic, bytes memory _data) public payable {
        assert(_IMPLEMENTATION_SLOT == bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1
        _setImplementation(_logic);
        if(_data.length > 0) {
            Address.functionDelegateCall(_logic, _data);
        }
    }

    /**
     * @dev Emitted when the implementation is upgraded.
     */
    event Upgraded(address indexed implementation);

    /**
     * @dev Storage slot with the address of the current imp
     * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, ar
     * validated in the constructor.
     */
    bytes32 private constant _IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a

    /**
     * @dev Returns the current implementation address.
     */
    function _implementation() internal view virtual override returns (address impl) {
        bytes32 slot = _IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            impl := sload(slot)
        }
    }

    /**
     * @dev Upgrades the proxy to a new implementation.
     *
     * Emits an {Upgraded} event.
     */
    function _upgradeTo(address newImplementation) internal virtual {
```

```
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

    /**
     * @dev Stores a new address in the EIP1967 implemer
     */
    function _setImplementation(address newImplementation) private {
        require(Address.isContract(newImplementation), "UpgradeableProxy: new implementation is not a

        bytes32 slot = _IMPLEMENTATION_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newImplementation)
        }
    }
}


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev This contract implements a proxy that is upgradeable by an
 *
 * To avoid https://medium.com/nomic-labs-blog/malicious-backdoors-in-ethereum-proxies-62629adf3357[proxy sel
 * clashing], which can potentially be used in an attack, this contract uses
 * https://blog.openzeppelin.com/ the -transparent-proxy-pattern/[transparent proxy patte
 * things that go hand in hand:
 *
 * 1. If any account other than the admin calls the proxy,
 * that call matches one of the admin functions exposed by the
 * 2. If the admin calls the proxy, it can access
 * implementation. If the admin tries to call a function on
 * "admin cannot fallback to proxy target".
 *
 * These properties mean that the admin account can only be used for admin actions
 * the admin, so it's best if it's a
 * to sudden errors when trying to call a function from the p
 *
 * Our recommendation is for the dedicated account to be an
 * you should think of the `Pro
 */
contract TransparentUpgradeableProxy is UpgradeableProxy {
    /**
     * @dev Initializes an upgradeable proxy managed by `_admin`, backed by
     * optionally initialized with `_data` as explained in {UpgradeableProxy-constructor}.
     */
    constructor(address _logic, address admin_, bytes memory _data) public payable UpgradeableProxy(_
        assert(_ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
        _setAdmin(admin_);
    }

    /**
     * @dev Emitted when the admin account has changed.
```

```
 */
  event AdminChanged(address previousAdmin, address newAdmin);

              /**
 * @dev Storage slot with the admin of the contract.
 * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
 * validated in the constructor.
 */
  bytes32 private constant _ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a71785

              /**
 * @dev Modifier used internally that will delegate the
 */
  modifier ifAdmin() {
      if (msg.sender == _admin()) {
          _;
      } else {
          _fallback();
      }
  }

              /**
 * @dev Returns the current admin.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 *
 * TIP: To get this value clients can read directly from the storage slot shown below (
 * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
 * `0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103`
 */
  function admin() external ifAdmin returns (address admin_) {
      admin_ = _admin();
  }

              /**
 * @dev Returns the current implementation.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 *
 * TIP: To get this value clients can read directly from the storage slot shown below (
 * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
 * `0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`
 */
  function implementation() external ifAdmin returns (address implementation_) {
      implementation_ = _implementation();
  }

              /**
 * @dev Changes the admin of the proxy.
 *
 * Emits an {AdminChanged} event.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 */
  function changeAdmin(address newAdmin) external virtual ifAdmin {
```

```
        require(newAdmin != address(0), "TransparentUpgradeableProxy: new admin is the zero address")
        emit AdminChanged(_admin(), newAdmin);
        _setAdmin(newAdmin);
    }

    /**
     * @dev Upgrade the implementation of the proxy.
     *
     * NOTE: Only the admin can call this function. See {Prox
     */
    function upgradeTo(address newImplementation) external virtual ifAdmin {
        _upgradeTo(newImplementation);
    }

    /**
     * @dev Upgrade the implementation of the proxy, and t
     * by `data`, which should be an encoded function call. T
     * proxied contract.
     *
     * NOTE: Only the admin can call this function. See {Prox
     */
    function upgradeToAndCall(address newImplementation, bytes calldata data) external payable virtua
        _upgradeTo(newImplementation);
        Address.functionDelegateCall(newImplementation, data);
    }

    /**
     * @dev Returns the current admin.
     */
    function _admin() internal view virtual returns (address adm) {
        bytes32 slot = _ADMIN_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            adm := sload(slot)
        }
    }

    /**
     * @dev Stores a new address in the EIP1967 admin slo
     */
    function _setAdmin(address newAdmin) private {
        bytes32 slot = _ADMIN_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newAdmin)
        }
    }

    /**
     * @dev Makes sure the admin cannot access the fallba
     */
    function _beforeFallback() internal virtual override {
        require(msg.sender != _admin(), "TransparentUpgradeableProxy: admin cannot fallback to proxy
        super._beforeFallback();
    }
}
```

TenBankHall::TransparentUpgradeableProxy.sol

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

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev This abstract contract provides a fallback function that delegates all calls to an
 * instruction `delegatecall`. We refer to the second contract as the
 * be specified by overriding the virtual {_implementation} function.
 *
 * Additionally, delegation to the implementation can be triggered manually through
 * different contract through the {_delegate} function.
 *
 * The success and return data of the delegated call will b
 */
abstract contract Proxy {
    /**
     * @dev Delegates the current call to `implementation`.
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _delegate(address implementation) internal virtual {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
            // delegatecall returns 0 on error.
            case 0 { revert(0, returndatasize()) }
            default { return(0, returndatasize()) }
        }
    }

    /**
     * @dev This is a virtual function that should be override
     * and {_fallback} should delegate.
     */
    function _implementation() internal view virtual returns (address);

    /**
     * @dev Delegates the current call to the address return
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _fallback() internal virtual {
```

```
        _beforeFallback();
        _delegate(_implementation());
    }

                /**
   * @dev Fallback function that delegates calls to the address returned by `_implemen
   * function in the contract matches the call data.
   */
    fallback () external payable virtual {
        _fallback();
    }

                /**
   * @dev Fallback function that delegates calls to the address returned by `_implemen
   * is empty.
   */
    receive () external payable virtual {
        _fallback();
    }

                /**
   * @dev Hook that is called before falling back to the implementation. Can happen a
   * call, or as part of the Solidity `fallback` or `receive` functions.
   *
   * If overriden should call `super._beforeFallback()`.
   */
    function _beforeFallback() internal virtual {
    }
}


pragma solidity >=0.6.2 <0.8.0;

            /**
 * @dev Collection of functions related to the address type
 */
library Address {
                /**
   * @dev Returns true if `account` is a contract.
   *
   * [IMPORTANT]
   * ====
   * It is unsafe to assume that an address for which this function returns
   * false is an externally-owned account (EOA) and not a
   *
   * Among others, `isContract` will return false for the fol
   * types of addresses:
   *
   * - an externally-owned account
   * - a contract in construction
   * - an address where a contract will
   * - an address where a contract lived, but
   * ====
   */
    function isContract(address account) internal view returns (bool) {
```

```
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

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

                /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-che
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

                /**
     * @dev Performs a Solidity function call using a low lev
     * plain`call` is an unsafe replacement for a function call:
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expe
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }
```

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

    /**
 * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
 * but also transferring `value` wei to `target`.
 *
 * Requirements:
 *
 * - the calling contract must have an ETH balance of at
 * - the called Solidity function must be `payable`.
 *
 * _Available since v3.1._
 */
  function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
      return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
  }

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

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

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

    /**
 * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
 * but performing a static call.
 *
```

```
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) intern
        require(isContract(target), "Address: static call to non-contract");

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

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

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

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


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev This contract implements an upgradeable proxy. It is upgradeable because cal
 * implementation address that can be changed. This address is stored in storage in the
```

```
 * https://eips.ethereum.org/EIPS/eip-1967[EIP1967], so that it doesn't
 * implementation behind the proxy.
 *
 * Upgradeability is only provided internally through {_upgradeTo}. For an externally up
 * {TransparentUpgradeableProxy}.
 */
contract UpgradeableProxy is Proxy {
    /**
     * @dev Initializes the upgradeable proxy with an initial
     *
     * If `_data` is nonempty, it's used as data in a delegate call to `_logic`. This
     * function call, and allows initializating the storage of the
     */
    constructor(address _logic, bytes memory _data) public payable {
        assert(_IMPLEMENTATION_SLOT == bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1
        _setImplementation(_logic);
        if(_data.length > 0) {
            Address.functionDelegateCall(_logic, _data);
        }
    }

    /**
     * @dev Emitted when the implementation is upgraded.
     */
    event Upgraded(address indexed implementation);

    /**
     * @dev Storage slot with the address of the current imp
     * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, ar
     * validated in the constructor.
     */
    bytes32 private constant _IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a

    /**
     * @dev Returns the current implementation address.
     */
    function _implementation() internal view virtual override returns (address impl) {
        bytes32 slot = _IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            impl := sload(slot)
        }
    }

    /**
     * @dev Upgrades the proxy to a new implementation.
     *
     * Emits an {Upgraded} event.
     */
    function _upgradeTo(address newImplementation) internal virtual {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

    /**
     * @dev Stores a new address in the EIP1967 implemer
```

```
   */
    function _setImplementation(address newImplementation) private {
        require(Address.isContract(newImplementation), "UpgradeableProxy: new implementation is not a

        bytes32 slot = _IMPLEMENTATION_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newImplementation)
        }
    }
}


pragma solidity >=0.6.0 <0.8.0;


            /**
 * @dev This contract implements a proxy that is upgradeable by an
 *
 * To avoid https://medium.com/nomic-labs-blog/malicious-backdoors-in-ethereum-proxies-62629adf3357[proxy sel
 * clashing], which can potentially be used in an attack, this contract uses
 * https://blog.openzeppelin.com/ the -transparent-proxy-pattern/[transparent proxy patte
 * things that go hand in hand:
 *
 * 1. If any account other than the admin calls the proxy,
 * that call matches one of the admin functions exposed by the
 * 2. If the admin calls the proxy, it can access
 * implementation. If the admin tries to call a function on
 * "admin cannot fallback to proxy target".
 *
 * These properties mean that the admin account can only be used for admin actions
 * the admin, so it's best if it's a
 * to sudden errors when trying to call a function from the p
 *
 * Our recommendation is for the dedicated account to be an
 * you should think of the `Pro
 */
contract TransparentUpgradeableProxy is UpgradeableProxy {
                /**
   * @dev Initializes an upgradeable proxy managed by `_admin`, backed by
   * optionally initialized with `_data` as explained in {UpgradeableProxy-constructor}.
   */
    constructor(address _logic, address admin_, bytes memory _data) public payable UpgradeableProxy(_
        assert(_ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
        _setAdmin(admin_);
    }

                /**
   * @dev Emitted when the admin account has changed.
   */
    event AdminChanged(address previousAdmin, address newAdmin);

                /**
   * @dev Storage slot with the admin of the contract.
   * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
```

```
 * validated in the constructor.
 */
  bytes32 private constant _ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a71785

              /**
 * @dev Modifier used internally that will delegate the
 */
  modifier ifAdmin() {
      if (msg.sender == _admin()) {
          _;
      } else {
          _fallback();
      }
  }

              /**
 * @dev Returns the current admin.
 *
 *              NOTE: Only the admin can call this function. See {Prox
 *
 * TIP: To get this value clients can read directly from the storage slot shown below (
 * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
 * `0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103`
 */
  function admin() external ifAdmin returns (address admin_) {
      admin_ = _admin();
  }

              /**
 * @dev Returns the current implementation.
 *
 *              NOTE: Only the admin can call this function. See {Prox
 *
 * TIP: To get this value clients can read directly from the storage slot shown below (
 * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
 * `0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`
 */
  function implementation() external ifAdmin returns (address implementation_) {
      implementation_ = _implementation();
  }

              /**
 * @dev Changes the admin of the proxy.
 *
 * Emits an {AdminChanged} event.
 *
 *              NOTE: Only the admin can call this function. See {Prox
 */
  function changeAdmin(address newAdmin) external virtual ifAdmin {
      require(newAdmin != address(0), "TransparentUpgradeableProxy: new admin is the zero address")
      emit AdminChanged(_admin(), newAdmin);
      _setAdmin(newAdmin);
  }

              /**
 * @dev Upgrade the implementation of the proxy.
```

```
     *
     *             NOTE:             Only             the             admin can call this function. See {Prox
     */
      function upgradeTo(address newImplementation) external virtual ifAdmin {
          _upgradeTo(newImplementation);
      }

                  /**
     * @dev Upgrade             the             implementation of             the             proxy, and t
     * by `data`, which             should             be             an             encoded function call. T
     * proxied contract.
     *
     *             NOTE:             Only             the             admin can call this function. See {Prox
     */
      function upgradeToAndCall(address newImplementation, bytes calldata data) external payable virtua
          _upgradeTo(newImplementation);
          Address.functionDelegateCall(newImplementation, data);
      }

                  /**
     * @dev Returns             the             current admin.
     */
      function _admin() internal view virtual returns (address adm) {
          bytes32 slot = _ADMIN_SLOT;
          // solhint-disable-next-line no-inline-assembly
          assembly {
              adm := sload(slot)
          }
      }

                  /**
     * @dev Stores             a             new address in             the             EIP1967 admin slo
     */
      function _setAdmin(address newAdmin) private {
          bytes32 slot = _ADMIN_SLOT;

          // solhint-disable-next-line no-inline-assembly
          assembly {
              sstore(slot, newAdmin)
          }
      }

                  /**
     * @dev Makes sure             the             admin cannot access             the             fallba
     */
      function _beforeFallback() internal virtual override {
          require(msg.sender != _admin(), "TransparentUpgradeableProxy: admin cannot fallback to proxy
          super._beforeFallback();
      }
}
```

PriceCheckerLPToken.sol

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


pragma solidity >=0.6.2 <0.8.0;
```

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

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
```

```
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

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

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
```

```
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) intern
        require(isContract(target), "Address: static call to non-contract");

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

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

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


// solhint-disable-next-line compiler-version
pragma solidity >=0.4.24 <0.8.0;


/**
 * @dev This is a base contract to aid in writing upgradeable contracts, or any kind of contract that
 * behind a proxy. Since a proxied contract can't have a constructor, it's common to move constructor
 * external initializer function, usually called `initialize`. It then becomes necessary to protect t
 * function so it can only be called once. The {initializer} modifier provided by this contract will
 *
 * TIP: To avoid leaving the proxy in an uninitialized state, the initializer function should be call
 * possible by providing the encoded function call as the `_data` argument to {UpgradeableProxy-const
 *
 * CAUTION: When used with inheritance, manual care must be taken to not invoke a parent initializer
 * that all initializers are idempotent. This is not verified automatically as constructors are by So
 */
abstract contract Initializable {

    /**
     * @dev Indicates that the contract has been initialized.
     */
    bool private _initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private _initializing;

    /**
     * @dev Modifier to protect an initializer function from being invoked twice.
     */
    modifier initializer() {
        require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is alrea
```

```
        bool isTopLevelCall = !_initializing;
        if (isTopLevelCall) {
            _initializing = true;
            _initialized = true;
        }

        _;

        if (isTopLevelCall) {
            _initializing = false;
        }
    }

    /// @dev Returns true if and only if the function is running in the constructor
    function _isConstructor() private view returns (bool) {
        return !AddressUpgradeable.isContract(address(this));
    }
}


pragma solidity >=0.6.0 <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 GSN 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 ContextUpgradeable is Initializable {
    function __Context_init() internal initializer {
        __Context_init_unchained();
    }

    function __Context_init_unchained() internal initializer {
    }
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
    uint256[50] private __gap;
}


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20Upgradeable {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
```

```
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMathUpgradeable {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
```

```
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }
```

```
    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
```

```
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20Upgradeable is Initializable, ContextUpgradeable, IERC20Upgradeable {
    using SafeMathUpgradeable for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
     * a default value of 18.
     *
     * To select a different value for {decimals}, use {_setupDecimals}.
     *
     * All three of these values are immutable: they can only be set once during
     * construction.
     */
    function __ERC20_init(string memory name_, string memory symbol_) internal initializer {
        __Context_init_unchained();
        __ERC20_init_unchained(name_, symbol_);
    }
```

```
    function __ERC20_init_unchained(string memory name_, string memory symbol_) internal initializer
        _name = name_;
        _symbol = symbol_;
        _decimals = 18;
    }

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

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

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual returns (uint8) {
        return _decimals;
    }

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

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

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

    /**
```

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

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
```

```
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC2
        return true;
    }

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

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

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

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

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

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

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

```
        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

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

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

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal virtual {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks]
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
    uint256[44] private __gap;
}


pragma solidity >=0.6.0 <0.8.0;

/**
 * @title SignedSafeMath
 * @dev Signed math operations with safety checks that revert on error.
 */
library SignedSafeMathUpgradeable {
    int256 constant private _INT256_MIN = -2**255;
```

```
    /**
     * @dev Returns the multiplication of two signed integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(int256 a, int256 b) internal pure returns (int256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        require(!(a == -1 && b == _INT256_MIN), "SignedSafeMath: multiplication overflow");

        int256 c = a * b;
        require(c / a == b, "SignedSafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two signed integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(int256 a, int256 b) internal pure returns (int256) {
        require(b != 0, "SignedSafeMath: division by zero");
        require(!(b == -1 && a == _INT256_MIN), "SignedSafeMath: division overflow");

        int256 c = a / b;

        return c;
    }

    /**
     * @dev Returns the subtraction of two signed integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(int256 a, int256 b) internal pure returns (int256) {
        int256 c = a - b;
        require((b >= 0 && c <= a) || (b < 0 && c > a), "SignedSafeMath: subtraction overflow");

        return c;
    }

    /**
     * @dev Returns the addition of two signed integers, reverting on
```

```
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(int256 a, int256 b) internal pure returns (int256) {
        int256 c = a + b;
        require((b >= 0 && c >= a) || (b < 0 && c < a), "SignedSafeMath: addition overflow");

        return c;
    }
}


pragma solidity >=0.6.0 <0.8.0;

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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    function __Ownable_init() internal initializer {
        __Context_init_unchained();
        __Ownable_init_unchained();
    }

    function __Ownable_init_unchained() internal initializer {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    /**
```

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
    uint256[49] private __gap;
}


pragma solidity >=0.5.0;

interface IUniswapV2Pair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
```

```
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}

pragma solidity 0.6.12;

interface ITokenOracle {
    function getPrice(address _token) external view returns (int);
}


pragma solidity 0.6.12;

interface IPriceChecker {
    function getPriceSlippage(address _lptoken) external view returns (uint256);
    function checkLPTokenPriceLimit(address _lptoken, bool _largeType) external view returns (bool);
}

pragma solidity 0.6.12;




contract PriceCheckerLPToken is OwnableUpgradeable, IPriceChecker {
    using SafeMathUpgradeable for uint256;
    using SignedSafeMathUpgradeable for int256;

    mapping(address => uint256) public priceSlippage;
    ITokenOracle public tokenOracle;
    uint256 public largeSlipRate;

    event SetPriceSlippage(address _lptoken, uint256 _oldv, uint256 _newv);
    event SetLargeSlipRate(uint256 _oldv, uint256 _newv);
    event SetTokenOracle(address _oldv, address _newv);

    constructor() public {
    }

    function initialize(address _tokenOracle) public initializer {
        __Ownable_init();
        largeSlipRate = 4e9;
        setTokenOracle(_tokenOracle);
    }

    function setLargeSlipRate(uint256 _largeSlipRate) external onlyOwner {
        require(_largeSlipRate >= 1e9, 'value error');
        emit SetLargeSlipRate(largeSlipRate, _largeSlipRate);
        largeSlipRate = _largeSlipRate;
    }

    function setPriceSlippage(address _lptoken, uint256 _slippage) external onlyOwner {
        require(_slippage >= 0 && _slippage <= 1e9, 'value error');
        emit SetPriceSlippage(_lptoken, priceSlippage[_lptoken], _slippage);
        priceSlippage[_lptoken] = _slippage;
    }

    function setTokenOracle(address _tokenOracle) public onlyOwner {
        emit SetTokenOracle(address(tokenOracle), _tokenOracle);
        tokenOracle = ITokenOracle(_tokenOracle);
```

```
    }

    function getPriceSlippage(address _lptoken) public override view returns (uint256) {
        if(priceSlippage[_lptoken] > 0) {
            return priceSlippage[_lptoken];
        }
        return uint256(1e7);
    }

    function getLPTokenPriceInMdex(address _lptoken, address _t0, address _t1) public view returns (u
        IUniswapV2Pair pair = IUniswapV2Pair(_lptoken);
        (uint256 r0, uint256 r1, ) = pair.getReserves();
        uint256 d0 = ERC20Upgradeable(_t0).decimals();
        uint256 d1 = ERC20Upgradeable(_t1).decimals();
        if(d0 != 18) {
            r0 = r0.mul(1e18).div(10**d0);
        }
        if(d1 != 18) {
            r1 = r1.mul(1e18).div(10**d1);
        }
        return r0.mul(1e18).div(r1);
    }


    function getLPTokenPriceInOracle(address _t0, address _t1) public view returns (uint256) {
        int256 price0 = tokenOracle.getPrice(_t0);
        int256 price1 = tokenOracle.getPrice(_t1);
        if(price0 <= 0 || price1 <= 0) {
            return 0;
        }
        int256 priceInOracle = price1.mul(1e18).div(price0);
        if(priceInOracle <= 0) {
            return 0;
        }
        return uint256(priceInOracle);
    }

    function checkLPTokenPriceLimit(address _lptoken, bool _largeType) external override view returns
        IUniswapV2Pair pair = IUniswapV2Pair(_lptoken);
        address t0 = pair.token0();
        address t1 = pair.token1();
        uint256 price0 = getLPTokenPriceInMdex(_lptoken, t0, t1);
        uint256 price1 = getLPTokenPriceInOracle(t0, t1);
        if(price0 == 0 || price1 == 0) {
            return false;
        }
        uint256 slip = getPriceSlippage(_lptoken);
        uint256 priceRate = price0.mul(1e9).div(price1);
        if(_largeType) {
            slip = slip.mul(largeSlipRate).div(1e9);
        }
        if(priceRate >= uint256(1e9).add(slip) || priceRate <= uint256(1e9).sub(slip)) {
            return false;
        }
        return true;
    }
}
```


ActionCompPools::TransparentUpgradeableProxy.sol

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

```
pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev This abstract contract provides a fallback function that delegates all calls to an
 * instruction `delegatecall`. We refer to the second contract as the
 * be specified by overriding the virtual {_implementation} function.
 *
 * Additionally, delegation to the implementation can be triggered manually through
 * different contract through the {_delegate} function.
 *
 * The success and return data of the delegated call will b
 */
abstract contract Proxy {
    /**
     * @dev Delegates the current call to `implementation`.
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _delegate(address implementation) internal virtual {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
            // delegatecall returns 0 on error.
            case 0 { revert(0, returndatasize()) }
            default { return(0, returndatasize()) }
        }
    }

    /**
     * @dev This is a virtual function that should be override
     * and {_fallback} should delegate.
     */
    function _implementation() internal view virtual returns (address);

    /**
     * @dev Delegates the current call to the address return
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _fallback() internal virtual {
        _beforeFallback();
        _delegate(_implementation());
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_impleme
```

```
     * function in the contract matches the call data.
     */
      fallback () external payable virtual {
          _fallback();
      }

                  /**
     * @dev Fallback function that delegates calls to the address returned by `_implemen
     * is empty.
     */
      receive () external payable virtual {
          _fallback();
      }

                  /**
     * @dev Hook that is called before falling back to the implementation. Can happen a
     * call, or as part of the Solidity `fallback` or `receive` functions.
     *
     * If overriden should call `super._beforeFallback()`.
     */
      function _beforeFallback() internal virtual {
      }
}


pragma solidity >=0.6.2 <0.8.0;

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

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

```
        return size > 0;
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-che
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low lev
     * plain`call` is an unsafe replacement for a function call:
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expe
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
```

```
 */
  function functionCall(address target, bytes memory data, string memory errorMessage) internal ret
      return functionCallWithValue(target, data, 0, errorMessage);
  }

                /**
 * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
 *              but              also transferring `value` wei to `target`.
 *
 * Requirements:
 *
 * -            the              calling contract must have          an             ETH balance of at
 * -            the              called Solidity function must be `payable`.
 *
 * _Available since v3.1._
 */
  function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
      return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
  }

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

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

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

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

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

```
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

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

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

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


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev This contract implements an upgradeable proxy. It is upgradeable because cal
 * implementation address that can be changed. This address is stored in storage in the
 * https://eips.ethereum.org/EIPS/eip-1967[EIP1967], so that it doesn't
 * implementation behind the proxy.
 *
 * Upgradeability is only provided internally through {_upgradeTo}. For an externally up
 * {TransparentUpgradeableProxy}.
```

```
 */
contract UpgradeableProxy is Proxy {
                /**
   * @dev Initializes the upgradeable proxy with an initial
   *
   * If `_data` is nonempty, it's used as data in a delegate call to `_logic`. This
   * function call, and allows initializating the storage of the
   */
    constructor(address _logic, bytes memory _data) public payable {
        assert(_IMPLEMENTATION_SLOT == bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1
        _setImplementation(_logic);
        if(_data.length > 0) {
            Address.functionDelegateCall(_logic, _data);
        }
    }

                /**
   * @dev Emitted when the implementation is upgraded.
   */
    event Upgraded(address indexed implementation);

                /**
   * @dev Storage slot with the address of the current imp
   * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, ar
   * validated in the constructor.
   */
    bytes32 private constant _IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a

                /**
   * @dev Returns the current implementation address.
   */
    function _implementation() internal view virtual override returns (address impl) {
        bytes32 slot = _IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            impl := sload(slot)
        }
    }

                /**
   * @dev Upgrades the proxy to a new implementation.
   *
   * Emits an {Upgraded} event.
   */
    function _upgradeTo(address newImplementation) internal virtual {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

                /**
   * @dev Stores a new address in the EIP1967 implemer
   */
    function _setImplementation(address newImplementation) private {
        require(Address.isContract(newImplementation), "UpgradeableProxy: new implementation is not a

        bytes32 slot = _IMPLEMENTATION_SLOT;

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

```
        assembly {
            sstore(slot, newImplementation)
        }
    }
}


pragma solidity >=0.6.0 <0.8.0;


            /**
 * @dev This contract implements a proxy that is upgradeable by an
 *
 * To avoid https://medium.com/nomic-labs-blog/malicious-backdoors-in-ethereum-proxies-62629adf3357[proxy sel
 * clashing], which can potentially be used in an attack, this contract uses
 * https://blog.openzeppelin.com/ the -transparent-proxy-pattern/[transparent proxy patte
 * things that go hand in hand:
 *
 * 1. If any account other than the admin calls the proxy,
 * that call matches one of the admin functions exposed by the
 * 2. If the admin calls the proxy, it can access
 * implementation. If the admin tries to call a function on
 * "admin cannot fallback to proxy target".
 *
 * These properties mean that the admin account can only be used for admin actions
 * the admin, so it's best if it's a
 * to sudden errors when trying to call a function from the p
 *
 * Our recommendation is for the dedicated account to be an
 * you should think of the `Pro
 */
contract TransparentUpgradeableProxy is UpgradeableProxy {
                /**
   * @dev Initializes an upgradeable proxy managed by `_admin`, backed by
   * optionally initialized with `_data` as explained in {UpgradeableProxy-constructor}.
   */
    constructor(address _logic, address admin_, bytes memory _data) public payable UpgradeableProxy(_
        assert(_ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
        _setAdmin(admin_);
    }

                /**
   * @dev Emitted when the admin account has changed.
   */
    event AdminChanged(address previousAdmin, address newAdmin);

                /**
   * @dev Storage slot with the admin of the contract.
   * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
   * validated in the constructor.
   */
    bytes32 private constant _ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a71785

                /**
   * @dev Modifier used internally that will delegate the
```

```
 */
  modifier ifAdmin() {
      if (msg.sender == _admin()) {
          _;
      } else {
          _fallback();
      }
  }

              /**
 * @dev Returns the current admin.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 *
 * TIP: To get this value clients can read directly from the storage slot shown below (
 * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
 * `0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103`
 */
  function admin() external ifAdmin returns (address admin_) {
      admin_ = _admin();
  }

              /**
 * @dev Returns the current implementation.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 *
 * TIP: To get this value clients can read directly from the storage slot shown below (
 * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
 * `0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`
 */
  function implementation() external ifAdmin returns (address implementation_) {
      implementation_ = _implementation();
  }

              /**
 * @dev Changes the admin of the proxy.
 *
 * Emits an {AdminChanged} event.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 */
  function changeAdmin(address newAdmin) external virtual ifAdmin {
      require(newAdmin != address(0), "TransparentUpgradeableProxy: new admin is the zero address")
      emit AdminChanged(_admin(), newAdmin);
      _setAdmin(newAdmin);
  }

              /**
 * @dev Upgrade the implementation of the proxy.
 *
 *             NOTE: Only the admin can call this function. See {Prox
 */
  function upgradeTo(address newImplementation) external virtual ifAdmin {
      _upgradeTo(newImplementation);
  }
```

```
    /**
     * @dev Upgrade the implementation of the proxy, and t
     * by `data`, which should be an encoded function call. T
     * proxied contract.
     *
     * NOTE: Only the admin can call this function. See {Prox
     */
    function upgradeToAndCall(address newImplementation, bytes calldata data) external payable virtua
        _upgradeTo(newImplementation);
        Address.functionDelegateCall(newImplementation, data);
    }

    /**
     * @dev Returns the current admin.
     */
    function _admin() internal view virtual returns (address adm) {
        bytes32 slot = _ADMIN_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            adm := sload(slot)
        }
    }

    /**
     * @dev Stores a new address in the EIP1967 admin slo
     */
    function _setAdmin(address newAdmin) private {
        bytes32 slot = _ADMIN_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newAdmin)
        }
    }

    /**
     * @dev Makes sure the admin cannot access the fallba
     */
    function _beforeFallback() internal virtual override {
        require(msg.sender != _admin(), "TransparentUpgradeableProxy: admin cannot fallback to proxy
        super._beforeFallback();
    }
}
```

DFOXPools::TransparentUpgradeableProxy.sol

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

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev This abstract contract provides a fallback function that delegates all calls to an
 * instruction `delegatecall`. We refer to the second contract as the
 * be specified by overriding the virtual {_implementation} function.
 *
```

```
 * Additionally, delegation to the implementation can be triggered manually through
 * different contract through the {_delegate} function.
 *
 * The success and return data of the delegated call will b
 */
abstract contract Proxy {
    /**
     * @dev Delegates the current call to `implementation`.
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _delegate(address implementation) internal virtual {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
            // delegatecall returns 0 on error.
            case 0 { revert(0, returndatasize()) }
            default { return(0, returndatasize()) }
        }
    }

    /**
     * @dev This is a virtual function that should be override
     * and {_fallback} should delegate.
     */
    function _implementation() internal view virtual returns (address);

    /**
     * @dev Delegates the current call to the address return
     *
     * This function does not return to its internall call site, it will return directly to
     */
    function _fallback() internal virtual {
        _beforeFallback();
        _delegate(_implementation());
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_impleme
     * function in the contract matches the call data.
     */
    fallback () external payable virtual {
        _fallback();
    }

    /**
```

```
    * @dev Fallback function that delegates calls to the address returned by `_impleme
    * is empty.
    */
    receive () external payable virtual {
        _fallback();
    }

    /**
    * @dev Hook that is called before falling back to the implementation. Can happen a
    * call, or as part of the Solidity `fallback` or `receive` functions.
    *
    * If overriden should call `super._beforeFallback()`.
    */
    function _beforeFallback() internal virtual {
    }
}


pragma solidity >=0.6.2 <0.8.0;

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

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

    /**
    * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
    * `recipient`, forwarding all available gas and reverting on errors.
    *
```

```
 * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
 * of certain opcodes, possibly making contracts go over the 2300 gas limit
 * imposed by `transfer`, making them unable to receive funds via
 * `transfer`. {sendValue} removes this limitation.
 *
 * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more
 *
 * IMPORTANT: because control is transferred to `recipient`, care must be
 * taken to not create reentrancy vulnerabilities. Consider using
 * {ReentrancyGuard} or the
 * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the -che
 */
  function sendValue(address payable recipient, uint256 amount) internal {
      require(address(this).balance >= amount, "Address: insufficient balance");

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

                  /**
 * @dev Performs a Solidity function call using a low lev
 * plain`call` is an unsafe replacement for a function call:
 * function instead.
 *
 * If `target` reverts with a revert reason, it is bubbled up by this
 * function ( like regular Solidity function calls).
 *
 * Returns the raw returned data. To convert to the expe
 * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding
 *
 * Requirements:
 *
 * - `target` must be a contract.
 * - calling `target` with `data` must not revert.
 *
 * _Available since v3.1._
 */
  function functionCall(address target, bytes memory data) internal returns (bytes memory) {
    return functionCall(target, data, "Address: low-level call failed");
  }

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

                  /**
 * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
```

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

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

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

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

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

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

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

```
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

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

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

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


pragma solidity >=0.6.0 <0.8.0;


/**
 * @dev This contract implements an upgradeable proxy. It is upgradeable because cal
 * implementation address that can be changed. This address is stored in storage in the
 * https://eips.ethereum.org/EIPS/eip-1967[EIP1967], so that it doesn't
 * implementation behind the proxy.
 *
 * Upgradeability is only provided internally through {_upgradeTo}. For an externally up
 * {TransparentUpgradeableProxy}.
 */
contract UpgradeableProxy is Proxy {
    /**
     * @dev Initializes the upgradeable proxy with an initial
     *
     * If `_data` is nonempty, it's used as data in a delegate call to `_logic`. This
     * function call, and allows initializating the storage of the
```

```
   */
    constructor(address _logic, bytes memory _data) public payable {
        assert(_IMPLEMENTATION_SLOT == bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1
        _setImplementation(_logic);
        if(_data.length > 0) {
            Address.functionDelegateCall(_logic, _data);
        }
    }

                /**
   * @dev Emitted when the implementation is upgraded.
   */
    event Upgraded(address indexed implementation);

                /**
   * @dev Storage slot with the address of the current imp
   * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, ar
   * validated in the constructor.
   */
    bytes32 private constant _IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a

                /**
   * @dev Returns the current implementation address.
   */
    function _implementation() internal view virtual override returns (address impl) {
        bytes32 slot = _IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            impl := sload(slot)
        }
    }

                /**
   * @dev Upgrades the proxy to a new implementation.
   *
   * Emits an {Upgraded} event.
   */
    function _upgradeTo(address newImplementation) internal virtual {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

                /**
   * @dev Stores a new address in the EIP1967 implemer
   */
    function _setImplementation(address newImplementation) private {
        require(Address.isContract(newImplementation), "UpgradeableProxy: new implementation is not a

        bytes32 slot = _IMPLEMENTATION_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newImplementation)
        }
    }
}


pragma solidity >=0.6.0 <0.8.0;
```

```
/**
 * @dev This contract implements a proxy that is upgradeable by an
 *
 * To avoid https://medium.com/nomic-labs-blog/malicious-backdoors-in-ethereum-proxies-62629adf3357[proxy sel
 * clashing], which can potentially be used in an attack, this contract uses
 * https://blog.openzeppelin.com/ the -transparent-proxy-pattern/[transparent proxy patte
 * things that go hand in hand:
 *
 * 1. If any account other than the admin calls the proxy,
 * that call matches one of the admin functions exposed by the
 * 2. If the admin calls the proxy, it can access
 * implementation. If the admin tries to call a function on
 * "admin cannot fallback to proxy target".
 *
 * These properties mean that the admin account can only be used for admin actions
 * the admin, so it's best if it's a
 * to sudden errors when trying to call a function from the p
 *
 * Our recommendation is for the dedicated account to be an
 * you should think of the `Pro
 */
contract TransparentUpgradeableProxy is UpgradeableProxy {
    /**
     * @dev Initializes an upgradeable proxy managed by `_admin`, backed by
     * optionally initialized with `_data` as explained in {UpgradeableProxy-constructor}.
     */
    constructor(address _logic, address admin_, bytes memory _data) public payable UpgradeableProxy(_
        assert(_ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
        _setAdmin(admin_);
    }

    /**
     * @dev Emitted when the admin account has changed.
     */
    event AdminChanged(address previousAdmin, address newAdmin);

    /**
     * @dev Storage slot with the admin of the contract.
     * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
     * validated in the constructor.
     */
    bytes32 private constant _ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a71785

    /**
     * @dev Modifier used internally that will delegate the
     */
    modifier ifAdmin() {
        if (msg.sender == _admin()) {
            _;
        } else {
            _fallback();
        }
    }
```

```
    /**
     * @dev Returns the current admin.
     *
     * NOTE: Only the admin can call this function. See {Prox
     *
     * TIP: To get this value clients can read directly from the storage slot shown below (
     * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
     * `0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103`
     */
    function admin() external ifAdmin returns (address admin_) {
        admin_ = _admin();
    }

    /**
     * @dev Returns the current implementation.
     *
     * NOTE: Only the admin can call this function. See {Prox
     *
     * TIP: To get this value clients can read directly from the storage slot shown below (
     * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
     * `0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`
     */
    function implementation() external ifAdmin returns (address implementation_) {
        implementation_ = _implementation();
    }

    /**
     * @dev Changes the admin of the proxy.
     *
     * Emits an {AdminChanged} event.
     *
     * NOTE: Only the admin can call this function. See {Prox
     */
    function changeAdmin(address newAdmin) external virtual ifAdmin {
        require(newAdmin != address(0), "TransparentUpgradeableProxy: new admin is the zero address")
        emit AdminChanged(_admin(), newAdmin);
        _setAdmin(newAdmin);
    }

    /**
     * @dev Upgrade the implementation of the proxy.
     *
     * NOTE: Only the admin can call this function. See {Prox
     */
    function upgradeTo(address newImplementation) external virtual ifAdmin {
        _upgradeTo(newImplementation);
    }

    /**
     * @dev Upgrade the implementation of the proxy, and t
     * by `data`, which should be an encoded function call. T
     * proxied contract.
     *
     * NOTE: Only the admin can call this function. See {Prox
```

```
    */
    function upgradeToAndCall(address newImplementation, bytes calldata data) external payable virtua
        _upgradeTo(newImplementation);
        Address.functionDelegateCall(newImplementation, data);
    }

    /**
     * @dev Returns the current admin.
     */
    function _admin() internal view virtual returns (address adm) {
        bytes32 slot = _ADMIN_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            adm := sload(slot)
        }
    }

    /**
     * @dev Stores a new address in the EIP1967 admin slo
     */
    function _setAdmin(address newAdmin) private {
        bytes32 slot = _ADMIN_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newAdmin)
        }
    }

    /**
     * @dev Makes sure the admin cannot access the fallba
     */
    function _beforeFallback() internal virtual override {
        require(msg.sender != _admin(), "TransparentUpgradeableProxy: admin cannot fallback to proxy
        super._beforeFallback();
    }
}
```

StrategyV2CherrySwapPool.sol

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

    /**
```

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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


pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
```

```
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }
```

```
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
```

```
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
```

```
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}


pragma solidity >=0.6.2 <0.8.0;

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

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

```
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
```

```
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memor
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

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

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

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

```
pragma solidity >=0.6.0 <0.8.0;


/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
```

```
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}


pragma solidity >=0.6.2;

interface IUniswapV2Router01 {
    function factory() external pure returns (address);
    function WHT() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityHT(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountHTMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountHT, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityHT(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountHTMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountHT);
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityHTWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountHTMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
```

```
    ) external returns (uint amountToken, uint amountHT);
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapExactHTForTokens(uint amountOutMin, address[] calldata path, address to, uint deadli
        external
        payable
        returns (uint[] memory amounts);
    function swapTokensForExactHT(uint amountOut, uint amountInMax, address[] calldata path, address
        external
        returns (uint[] memory amounts);
    function swapExactTokensForHT(uint amountIn, uint amountOutMin, address[] calldata path, address
        external
        returns (uint[] memory amounts);
    function swapHTForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
        external
        payable
        returns (uint[] memory amounts);

    function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint
    function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memo
    function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memo
}

pragma solidity >=0.6.2;


interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityHTSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountHTMin,
        address to,
        uint deadline
    ) external returns (uint amountHT);
    function removeLiquidityHTWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountHTMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountHT);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
```

```
    function swapExactHTForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForHTSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
}

pragma solidity >=0.5.0;

interface IUniswapV2Pair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
```

```
}

pragma solidity >=0.5.0;

interface IUniswapV2Factory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);
    function migrator() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;
    function setMigrator(address) external;
}

pragma solidity >=0.5.0 <0.8.0;

interface ICherryPool {
    function cherry() external view returns (address);

    function poolLength() external view returns (uint256);

    function poolInfo(uint256 _pid) external view returns(address _lpToken, uint256, uint256);

    function userInfo(uint256 _pid, address _user) external view returns (uint256 _amount, uint256 _r

    function deposit(uint256 _pid, uint256 _amount) external;

    function pendingCherry(uint256 _pid, address _user) external view returns (uint256);

    function withdraw(uint256 _pid, uint256 _amount) external;

    function emergencyWithdraw(uint256 _pid) external;
}


pragma solidity 0.6.12;

interface IStrategyV2SwapPool {

    function setStrategy(address _strategy) external;

    // get strategy
    function getName() external view returns (string memory);

    // swap functions
    function getPair(address _t0, address _t1) external view returns (address pairs);
    function getReserves(address _lpToken) external view returns (uint256 a, uint256 b);
    function getToken01(address _pairs) external view returns (address token0, address token1);
    function getAmountOut(address _tokenIn, address _tokenOut, uint256 _amountOut) external view retu
    function getAmountIn(address _tokenIn, uint256 _amountIn, address _tokenOut) external view return
    function getLPTokenAmountInBaseToken(address _lpToken, uint256 _lpTokenAmount, address _baseToken
    function swapTokenTo(address _tokenIn, uint256 _amountIn, address _tokenOut, address _toAddress)

    function optimalBorrowAmount(address _lpToken, uint256 _amount0, uint256 _amount1) external view
    function optimalDepositAmount(address lpToken, uint amtA, uint amtB) external view returns (uint

    // pool functions
    function getDepositToken(uint256 _poolId) external view returns (address lpToken);
```

```
    function getRewardToken(uint256 _poolId) external view returns (address rewardToken);
    function getPending(uint256 _poolId) external view returns (uint256 rewards);
    function deposit(uint256 _poolId, bool _autoPool) external returns (uint256 liquidity);
    function withdraw(uint256 _poolId, uint256 _liquidity, bool _autoPool) external returns (uint256
    function claim(uint256 _poolId) external returns (uint256 rewards);
    function extraRewards() external returns (address token, uint256 rewards);
}

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMathUpgradeable {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
```

```
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
```

```
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
```

```
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}

pragma solidity 0.6.12;


library TenMath {
  using SafeMathUpgradeable for uint256;

  function min(uint256 v1, uint256 v2) public pure returns (uint256 vr) {
    vr = v1 > v2 ? v2 : v1;
  }

  function safeSub(uint256 v1, uint256 v2) internal pure returns (uint256 vr) {
    vr = v1 > v2 ? v1.sub(v2) : 0;
  }

  // implementation from https://github.com/Uniswap/uniswap-lib/commit/99f3f28770640ba1bb1ff460ac7c52
  // original implementation: https://github.com/abdk-consulting/abdk-libraries-solidity/blob/master/
  function sqrt(uint256 x) internal pure returns (uint256) {
    if (x == 0) return 0;
    uint256 xx = x;
    uint256 r = 1;

    if (xx >= 0x100000000000000000000000000000000) {
      xx >>= 128;
      r <<= 64;
    }

    if (xx >= 0x10000000000000000) {
      xx >>= 64;
      r <<= 32;
    }
    if (xx >= 0x100000000) {
      xx >>= 32;
      r <<= 16;
    }
    if (xx >= 0x10000) {
      xx >>= 16;
      r <<= 8;
    }
    if (xx >= 0x100) {
      xx >>= 8;
      r <<= 4;
    }
    if (xx >= 0x10) {
```

```
      xx >>= 4;
      r <<= 2;
    }
    if (xx >= 0x8) {
      r <<= 1;
    }

    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1;
    r = (r + x / r) >> 1; // Seven iterations should be enough
    uint256 r1 = x / r;
    return (r < r1 ? r : r1);
  }
}


pragma solidity 0.6.12;




// https://www.cherryswap.net/

// Connecting to third party swap for pool lptoken
contract StrategyV2CherrySwapPool is IStrategyV2SwapPool {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IUniswapV2Factory public constant factory = IUniswapV2Factory(0x709102921812B3276A65092Fe79eDfc76
    IUniswapV2Router02 public constant router = IUniswapV2Router02(0x865bfde337C8aFBffF144Ff4C29f9404

    ICherryPool public constant farmpool = ICherryPool(0x8cddB4CD757048C4380ae6A69Db8cD5597442f7b);
    address public constant swappool = address(0);
    address public constant rewardToken = address(0x8179D97Eb6488860d816e3EcAFE694a4153F216c);

    address public strategy;

    constructor() public {

    }

    function setStrategy(address _strategy) external override {
        require(strategy == address(0), 'once');
        strategy = _strategy;
    }

    modifier onlyStrategy() {
        require(msg.sender == strategy);
        _;
    }

    function getName() external override view returns (string memory name) {
        name = 'cherry';
    }

    function getPair(address _t0, address _t1)
        public override view returns (address pairs) {
        pairs = factory.getPair(_t0, _t1);
    }

    function getToken01(address _pairs)
        public override view returns (address token0, address token1) {
```

```
        token0 = IUniswapV2Pair(_pairs).token0();
        token1 = IUniswapV2Pair(_pairs).token1();
    }

    function getReserves(address _lpToken)
        public override view returns (uint256 a, uint256 b) {
        (a, b, ) = IUniswapV2Pair(_lpToken).getReserves();
    }

    function getAmountOut(address _tokenIn, address _tokenOut, uint256 _amountOut)
        external override view returns (uint256) {
        if(_tokenIn == _tokenOut) {
            return _amountOut;
        }
        if(_amountOut == 0) {
            return 0;
        }
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = _tokenOut;
        uint256[] memory result = router.getAmountsIn(_amountOut, path);
        if(result.length == 0) {
            return 0;
        }
        return result[0];
    }

    function getAmountIn(address _tokenIn, uint256 _amountIn, address _tokenOut)
        public override view returns (uint256) {
        if(_tokenIn == _tokenOut) {
            return _amountIn;
        }
        if(_amountIn == 0) {
            return 0;
        }
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = _tokenOut;
        uint256[] memory result = router.getAmountsOut(_amountIn, path);
        if(result.length == 0) {
            return 0;
        }
        return result[result.length-1];
    }

    function getLPTokenAmountInBaseToken(address _lpToken, uint256 _lpTokenAmount, address _baseToken
        external override view returns (uint256 amount) {
        (uint256 a, uint256 b) = getReserves(_lpToken);
        (address token0, address token1) = getToken01(_lpToken);
        uint256 totalSupply = IERC20(_lpToken).totalSupply();
        if(token0 == _baseToken) {
            amount = _lpTokenAmount.mul(a).div(totalSupply).mul(2);
        }else if(token1 == _baseToken) {
            amount = _lpTokenAmount.mul(b).div(totalSupply).mul(2);
        }
        else{
            require(false, 'unsupport baseToken not in pairs');
        }
    }

    function swapTokenTo(address _tokenIn, uint256 _amountIn, address _tokenOut, address _toAddress)
        public override onlyStrategy returns (uint256) {
        if(_tokenIn == _tokenOut) {
            return _safeTransferAll(_tokenOut, _toAddress);
        }
        if(_amountIn == 0 || getAmountIn(_tokenIn, _amountIn, _tokenOut) <= 0) {
```

```
            return 0;
        }
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = _tokenOut;
        IERC20(_tokenIn).approve(address(router), _amountIn);
        uint256[] memory result = router.swapExactTokensForTokens(_amountIn, 0, path, _toAddress, blo
        if(result.length == 0) {
            return 0;
        } else {
            return result[result.length-1];
        }
    }

    function optimalBorrowAmount(address _lpToken, uint256 _amount0, uint256 _amount1)
        external view override returns (uint256 borrow0, uint256 borrow1) {
        (uint256 a, uint256 b) = getReserves(_lpToken);
        if (a.mul(_amount1) >= b.mul(_amount0)) {
            borrow0 = _amount1.mul(a).div(b).sub(_amount0);
            borrow1 = 0;
        } else {
            borrow0 = 0;
            borrow1 = _amount0.mul(b).div(a).sub(_amount1);
        }
    }

    /// @dev Compute optimal deposit amount
    /// @param lpToken amount
    /// @param amtA amount of token A desired to deposit
    /// @param amtB amount of token B desired to deposit
    function optimalDepositAmount(
        address lpToken,
        uint amtA,
        uint amtB
    ) public override view returns (uint swapAmt, bool isReversed) {
        (uint256 resA, uint256 resB) = getReserves(lpToken);
        if (amtA.mul(resB) >= amtB.mul(resA)) {
            swapAmt = _optimalDepositA(amtA, amtB, resA, resB);
            isReversed = false;
        } else {
            swapAmt = _optimalDepositA(amtB, amtA, resB, resA);
            isReversed = true;
        }
    }

    /// @dev Compute optimal deposit amount helper.
    /// @param amtA amount of token A desired to deposit
    /// @param amtB amount of token B desired to deposit
    /// @param resA amount of token A in reserve
    /// @param resB amount of token B in reserve
    /// Formula: https://blog.alphafinance.io/byot/
    function _optimalDepositA(
        uint amtA,
        uint amtB,
        uint resA,
        uint resB
    ) internal pure returns (uint) {
        require(amtA.mul(resB) >= amtB.mul(resA), 'Reversed');
        uint a = 997;
        uint b = uint(1997).mul(resA);
        uint _c = (amtA.mul(resB)).sub(amtB.mul(resA));
        uint c = _c.mul(1000).div(amtB.add(resB)).mul(resA);
        uint d = a.mul(c).mul(4);
        uint e = TenMath.sqrt(b.mul(b).add(d));
        uint numerator = e.sub(b);
        uint denominator = a.mul(2);
```

```
        return numerator.div(denominator);
    }

    function getDepositToken(uint256 _poolId)
        public override view returns (address lpToken) {
        (lpToken,,) = farmpool.poolInfo(_poolId);
    }

    function getRewardToken(uint256 _poolId)
        external override view returns (address returnToken) {
        return rewardToken;
    }

    function getPending(uint256 _poolId) external override view returns (uint256 rewards) {
        rewards = farmpool.pendingCherry(_poolId, address(this));
    }

    function deposit(uint256 _poolId, bool _autoPool)
        external override onlyStrategy returns (uint256 liquidity) {
        address lpToken = getDepositToken(_poolId);
        (address tokenA, address tokenB) = getToken01(lpToken);
        uint256 amountA;
        uint256 amountB;
        amountA = IERC20(tokenA).balanceOf(address(this));
        amountB = IERC20(tokenB).balanceOf(address(this));
        (uint256 swapAmt, bool isReversed) = optimalDepositAmount(lpToken, amountA, amountB);
        if(swapAmt > 0) {
            swapTokenTo(isReversed?tokenB:tokenA, swapAmt, isReversed?tokenA:tokenB, address(this));
        }
        amountA = IERC20(tokenA).balanceOf(address(this));
        amountB = IERC20(tokenB).balanceOf(address(this));
        if(amountA > 0 && amountB > 0) {
            IERC20(tokenA).approve(address(router), amountA);
            IERC20(tokenB).approve(address(router), amountB);
            router.addLiquidity(tokenA, tokenB,
                                amountA, amountB,
                                0, 0,
                                address(this), block.timestamp.add(60));
            liquidity = IERC20(lpToken).balanceOf(address(this));
            if(liquidity > 0 && _autoPool) {
                IERC20(lpToken).approve(address(farmpool), liquidity);
                farmpool.deposit(_poolId, liquidity);
            }
        }
        _safeTransferAll(lpToken, strategy);
        _safeTransferAll(tokenA, strategy);
        _safeTransferAll(tokenB, strategy);
    }

    function withdraw(uint256 _poolId, uint256 _liquidity, bool _autoPool)
        external override onlyStrategy returns (uint256 amountA, uint256 amountB) {
        if(_liquidity <= 0) return (0, 0);
        if(_autoPool) {
            farmpool.withdraw(_poolId, _liquidity);
        }
        address lpToken = getDepositToken(_poolId);
        (address tokenA, address tokenB) = getToken01(lpToken);
        IERC20(lpToken).approve(address(router), _liquidity);
        router.removeLiquidity(tokenA, tokenB,
                               _liquidity,
                               0, 0,
                               strategy, block.timestamp.add(60));
        amountA = _safeTransferAll(tokenA, strategy);
        amountB = _safeTransferAll(tokenB, strategy);
    }
```

```
    function claim(uint256 _poolId)
        external override onlyStrategy returns (uint256 rewards) {
        farmpool.withdraw(_poolId, 0);
        rewards = _safeTransferAll(rewardToken, strategy);
    }

    function extraRewards()
        external override onlyStrategy returns (address token, uint256 rewards) {
    }

    function _safeTransferAll(address _token, address _to)
        internal returns (uint256 value){
        value = IERC20(_token).balanceOf(address(this));
        if(value > 0) {
            IERC20(_token).safeTransfer(_to, value);
        }
    }

}
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand()

function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io