**Atomic Swap ERC20**

This is a smart contract written in Solidity language to facilitate atomic swap of ERC20 tokens.

**Description**

This contract allows two parties to exchange ERC20 tokens in a trustless and decentralized way. Atomic swap ensures that either both parties successfully complete the trade or the transaction is reverted. This contract also implements a time lock mechanism to ensure timely completion of the trade.

**Requirements**

* Ethereum wallet with support for ERC20 tokens
* Solidity compiler
* ERC20 token addresses and their corresponding values

**Usage**

1. Deploy the contract on the Ethereum network using the Solidity compiler.
2. Use the **open** function to initiate a swap by providing the ERC20 token addresses and values.
3. Wait for the other party to respond with a corresponding swap offer.
4. Use the **close** function to complete the swap and transfer tokens to the respective parties.
5. Use the **expire** function to refund the original token holder if the other party fails to complete the swap within the specified time frame.

**License**

This code is licensed under the MIT license.

**Lending and Borrowing**

This Solidity code defines a smart contract for lending and borrowing tokens with a collateral requirement. Here's a summary of the contract:

* The contract is called **LendingAndBorrowing** and imports the OpenZeppelin **Ownable** and **IERC20** contracts.
* The contract contains several mappings to keep track of the amount of tokens lent and borrowed, the amount of collateral held, and the amount of collateral locked.
* The contract also contains two arrays, **tokensForLending** and **tokensForBorrowing**, which hold information about the tokens that are available for lending and borrowing, respectively.
* The contract contains several functions to add new tokens to the lending and borrowing lists, change the collateral token, and view the lending and borrowing lists.
* The **toLend** function allows users to lend tokens to the contract.
* The **toWithdrawLentTokens** function allows users to withdraw tokens they have lent to the contract.
* The **depositCollateral** function allows users to deposit collateral in order to borrow tokens.
* The **borrowTokens** function allows users to borrow tokens using collateral.
* The **payDebt** function allows users to pay back the tokens they have borrowed.
* The **releaseCollateral** function allows users to release their collateral after they have paid back their debt.

Note that this is a simplified summary and the actual contract may contain additional complexity and details.