DOCUMENTATION FOR PRESALE CONTRACT

1) INTRODUCTION

This document provides a detailed explanation of the presale contract. The contract facilitates the creation and management of pre-sale events, allowing investors to lock their investments and later withdraw their tokens based on a vesting schedule. The contract also includes referral functionality and the ability to categorize investors based on their investment amounts.

2) CONTRACT OVERVIEW

The presale contract is implemented in Solidity and contains various functionalities to manage pre-sale events and investor token transactions. The key features and functionalities of the contract include:

- Creating pre-sale events with specified token details and timeframes
- Allowing investors to lock their investments by providing USDT
- Applying a vesting schedule to the locked tokens until the Token Generation Event (TGE)
- Enabling investors to withdraw their unlocked tokens after the lock period
- Supporting referral functionality with referral codes and referral rewards

- Categorizing investors based on their investment amounts at the end of all pre-sale events.

2.1. STRUCTURES USED

The contract defines two structs:

- Phase: Stores the details of a pre-sale phase, including the start time, end time, total tokens, token price, and tokens sold.
- Investor: Stores the details of an investor, including their token balance, unlocked tokens, claimed tokens, lock time, referral code, and status flags.

2.2. STATE VARIABLES USED

The contract defines several state variables, including:

- **tgeTimestamp**: Represents the Token Generation Event timestamp.
- **token**: An instance of the IERC20 token contract.
- **tokenOwner**: The address of the token owner.
- **idWiseInvestorList**: A mapping to store the list of investors for each pre-sale event ID.
- **presaleEventid**: A counter to track the pre-sale event IDs.
- **referralCode**: A counter to generate referral codes.
- **idToPhaseMapping**: A mapping to store the Phase struct for each pre-sale event ID.
- **investorMapping**: A mapping to store the Investor struct for each pre-sale event ID and investor address.

- **referralCodeToReferralAddress**: A mapping to associate referral codes with corresponding addresses.
- **referralAddressToReferralCode**: A mapping to associate addresses with corresponding referral codes.
- **cumulativeInvestment**: A mapping to store the cumulative investment amounts for each investor.
- **decimalFactor**: A variable to handle fractional token prices.

2.3. EVENTS USED

The contract emits the following events:

- **PreSaleCreated**: Triggered when a pre-sale event is created, providing details such as total tokens, token price, start time, end time, and the pre-sale event ID.
- **Tokenslocked:** Triggered when an investor locks their investment, providing details such as the investor address, pre-sale event ID, investment amount, and referral code (if applicable).
- **TokensWithdrawn:** Triggered when an investor claims their unlocked tokens, providing details such as the investor address, pre-sale event ID, and the number of claimed tokens.

2.4. FUNCTIONS USED

The functions used in the contract are as follows:

- **CreatePreSale:** This function is used to create a new pre-sale event with the specified parameters. It requires the total number of tokens for the pre-sale event, the token price in USDT, the start time, and the end time. The function emits the PreSaleCreated event.

- **Lock:** This function allows investors to lock their investments by providing the specified amount of USDT. It requires the pre-sale event ID, the investment amount, and an optional referral code. The function calculates the number of tokens based on the token price and the investment amount and updates the investor's token balance. The function emits the TokensLocked event.

- **Withdraw:** This function allows investors to withdraw their unlocked tokens after the lock period. It requires the pre-sale event ID. The function calculates the number of unlocked tokens based on the lock time and the vesting schedule and updates the investor's token balance. The function emits the TokensUnlocked event.

- **GenerateReferral:** This internal function generates a unique referral code for an investor. It uses the counter function to generate a code for the investors. The generated code is returned as a uint256.

- **FindCategory:** This function allows anyone to find the category of a user based on their token count. It returns the Category.

3. TEST CASES

The below documentation provides an overview of all the test cases implemented on the Presale smart contract.

3.1. LIBRARIES USED

The libraries used for the Test Cases are:

- **Chai:** Chai is an assertion library for JavaScript that provides a set of functions for making assertions in tests. It is used in this test case to assert the expected outcomes.
- **Mocha:** Mocha is a JavaScript test framework that provides a structure for organizing test cases and running them. It is used in this test case to define the test suite and test cases.
- **Time**: The Time library from the **@nomicfoundation/hardhat-network-helpers** package provides helper functions for working with time in Hardhat tests. It is used in this test case to calculate timestamps for the start and end times of the pre-sale events.

3.2. TEST CASES

The Various Test cases for the Presale Smart Contract are described below.

3.2.1. CREATE PRE SALE-FUNCTION

Test cases for the Create pre-sale function are as follows.

3.2.2. Test Case #1 Should Create a Pre-sale with given parameters.

Type: Functional, Positive

Description: This test case verifies that the createPreSale function creates a pre-sale event with the specified parameters.

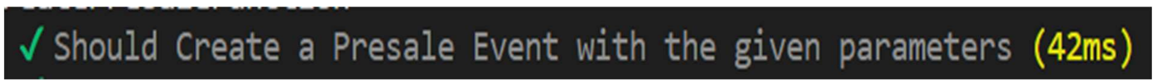
Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens, token price, start time, and end time.

Expected Outcome:

- The pre-sale event ID should be set to 1.
- The mapping idToPhaseMapping should contain the correct phase details.

Output:



```
✓ Should Create a Presale Event with the given parameters (42ms)
```

3.2.3. Only the User should be able to Create Pre Sale

Type: Functional, Negative

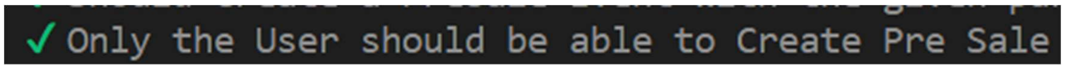
Description: This test case verifies that only the owner can create a pre-sale event.

Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens, token price, start time, and end time using a non-owner account.

Expected Outcome: The transaction should be reverted with the error message "Ownable: caller is not the owner".

Output:



```
✓ Only the User should be able to Create Pre Sale
```

3.2.4. Presale should emit an event when an event is created

Type: Functional, Positive

Description: This test case verifies that the **PreSaleCreated** event is emitted when a pre-sale event is created.

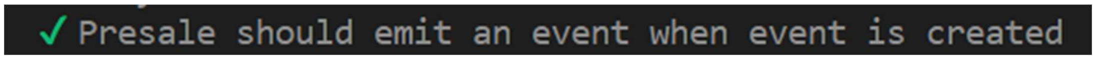
Steps:

1. Deploy the **presale** contract.
2. Call the **createPreSale** function with the total tokens, token price, start time, and end time.
3. Get the transaction receipt for the transaction.

Expected Outcome:

- The **PreSaleCreated** event should be emitted.
- The event arguments should match the provided parameters.

OUTPUT:



```
✓ Presale should emit an event when event is created
```

3.2.5. Should create a presale event when the start time is greater than the current time

Type: Functional, Positive

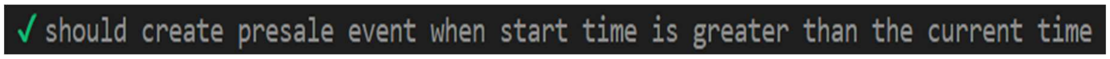
Description: This test case verifies that a pre-sale event can be created when the start time is in the future.

Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens, token price, start time (in the future), and end time.

Expected Outcome: The transaction should emit the PreSaleCreated event.

Output:



```
✓ should create presale event when start time is greater than the current time
```

3.2.6. Should revert when start time is less than the current time

Type: Functional, Negative

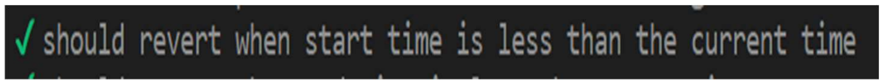
Description: This test case verifies that the createPreSale function reverts when the start time is in the past.

Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens, token price, start time (in the past), and end time.

Expected Outcome: The transaction should be reverted with the error message "Current time is greater than start time".

Output:



```
✓ should revert when start time is less than the current time
```

3.2.7. Should revert when the end time is less than the start time

Type: Functional, Negative

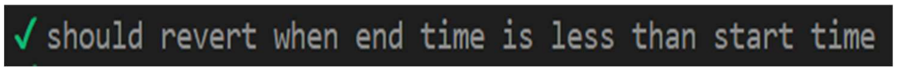
Description: This test case verifies that the createPreSale function reverts when the end time is before the start time.

Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens, token price, start time, and end time (before the start time).

Expected Outcome: The transaction should be reverted with the error message "Start time is greater than end time".

Output:



```
✓ should revert when end time is less than start time
```

3.2.8. Should create a presale event when the end time is greater than the start time

Type: Functional, Positive

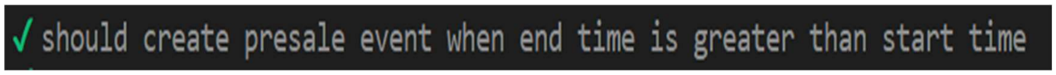
Description: This test case verifies that a pre-sale event can be created when the end time is after the start time.

Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens, token price, start time, and end time (after the start time).

Expected Outcome: The transaction should emit the PreSaleCreated event.

Output:



```
✓ should create presale event when end time is greater than start time
```

3.2.9. Should revert when the start time and the end time are the same

Type: Functional, Negative

Description: This test case verifies that the createPreSale function reverts when the start time and end time are the same.

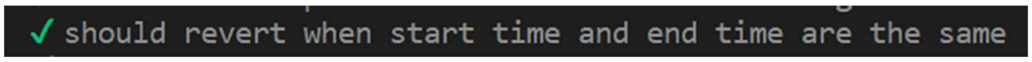
Steps:

- Deploy the presale contract.

- Call the createPreSale function with the total tokens, token price, start time, and end time (same value).

Expected Outcome: The transaction should be reverted with the error message "Start time is greater than end time".

Output:



```
✓ should revert when start time and end time are the same
```

3.2.10. Should revert when the start time and the end time are same

Type: Functional, Negative

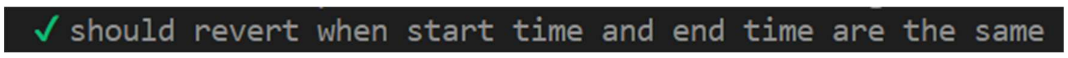
Description: This test case verifies that the createPreSale function reverts when the start time and end time are the same.

Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens, token price, start time, and end time (same value).

Expected Outcome: The transaction should be reverted with the error message "Start time is greater than end time".

Output:



```
✓ should revert when start time and end time are the same
```

3.2.11. Should Revert when the Start time is greater than the TGE timestamp

Type: Functional, Negative

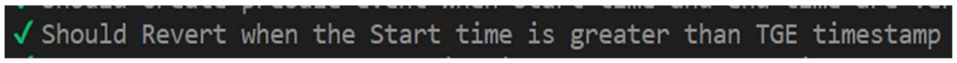
Description: This test case verifies that the createPreSale function reverts when the start time is after the TGE timestamp.

Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens, token price, start time (after TGE), and end time.

Expected Outcome: The transaction should be reverted with the error message "Start time is more than TGE".

Output:



```
✓ Should Revert when the Start time is greater than TGE timestamp
```

3.2.12. Should Revert when the End time is greater than the TGE timestamp

Type: Functional, Negative

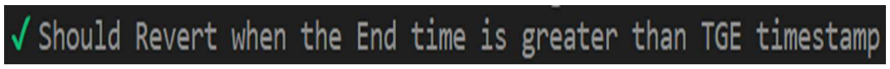
Description: This test case verifies that the createPreSale function reverts when the end time is after the TGE timestamp.

Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens, token price, start time, and end time (after TGE).

Expected Outcome: The transaction should be reverted with the error message "end time should be less than TGE".

Output:



```
✓ Should Revert when the End time is greater than TGE timestamp
```

3.2.13. Should revert when total tokens are Zero

Type: Functional, Negative

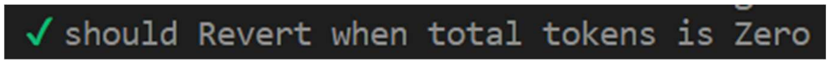
Description: This test case verifies that the createPreSale function reverts when the total tokens are zero.

Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens as zero, token price, start time, and end time.

Expected Outcome: The transaction should be reverted with the error message "ERR-1".

Output:



```
✓ should Revert when total tokens is Zero
```

3.2.14. Should revert when tokens price is Zero

Type: Functional, Negative

Description: This test case verifies that the createPreSale function reverts when the token price is zero.

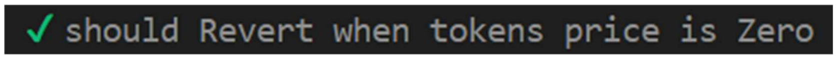
Steps:

- Deploy the presale contract.

- Call the createPreSale function with the total tokens, token price as zero, start time, and end time.

Expected Outcome: The transaction should be reverted with the error message "ERR-1".

Output:



```
✓ should Revert when tokens price is Zero
```

3.2.15. Should Create Overlapping Events

Type: Functional, Positive

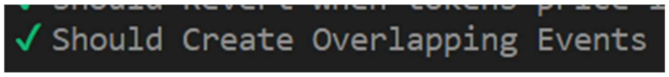
Description: This test case verifies that multiple pre-sale events can be created even if they overlap in time.

Steps:

- Deploy the presale contract.
- Call the createPreSale function with the total tokens, token price, start time, and end time for the first event
- Call the createPreSale function with the total tokens, token price, start time, and end time for the second event (overlapping with the first event).

Expected Outcome: Both transactions should emit the PreSaleCreated event.

Output:



```
✓ Should Create Overlapping Events
```

3.3. TEST CASES FOR THE LOCK FUNCTION

The test cases for the lock function are as follows.

3.3.1. Should revert if the current time is less than the start time

Type: Functional, Negative

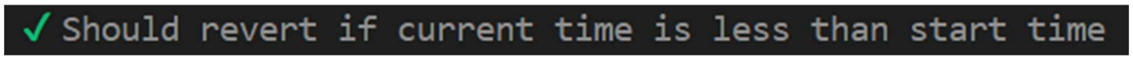
Description: This test case verifies that the lock function reverts when the current time is less than the start time of the pre-sale event.

Steps:

- Deploy the presale contract and create a pre-sale event.
- Call the lock function with the event ID, token amount, and referral code (optional) before the start time.

Expected Outcome: The transaction should be reverted with the error message "ERR-4".

Output:



```
✓ Should revert if current time is less than start time
```

3.3.2. Should update no balance when EventID is invalid

Type: Functional, Negative

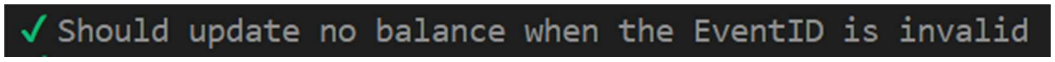
Description: This test case verifies that the lock function reverts when the EventID provided is invalid (zero).

Steps:

- Deploy the presale contract and create a pre-sale event.
- Call the lock function with an invalid EventID (zero), token amount, and referral code (optional).

Expected Outcome: The transaction should be reverted.

Output:



```
✓ Should update no balance when the EventID is invalid
```

3.3.3. Should lock the tokens when the EventID is valid

Type: Functional, Positive

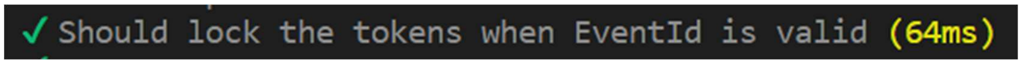
Description: This test case verifies that tokens can be locked when a valid EventID is provided.

Steps:

- Deploy the presale contract and create a pre-sale event.
- Call the lock function with a valid EventID, token amount, and referral code (optional).

Expected Outcome: The transaction should emit the TokensLocked event.

Output:



```
✓ Should lock the tokens when EventId is valid (64ms)
```

3.3.4. Should calculate the correct token amount when USDT is 1

Type: Functional, Positive

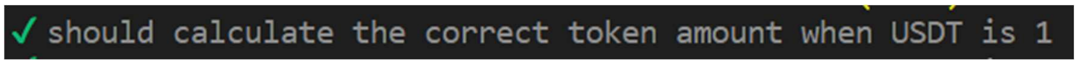
Description: This test case verifies that the lock function calculates the correct token amount when the USDT amount is 1.

Steps:

- Deploy the presale contract and create a pre-sale event.
- Call the lock function with a valid EventID, USDT amount of 1, and referral code (optional).

Expected Outcome: The locked tokens should be calculated correctly based on the token price, and the investor's balance should be updated accordingly.

Output:



```
✓ should calculate the correct token amount when USDT is 1
```

3.3.5. Should calculate the correct token amount when USDT is 10

Type: Functional, Positive

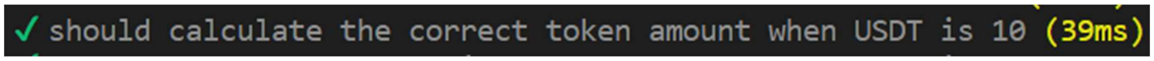
Description: This test case verifies that the lock function calculates the correct token amount when the USDT amount is 10.

Steps:

- Deploy the presale contract and create a pre-sale event.
- Call the lock function with a valid EventID, USDT amount of 10, and referral code (optional).

Expected Outcome: The locked tokens should be calculated correctly based on the token price, and the investor's balance should be updated accordingly.

Output:



```
✓ should calculate the correct token amount when USDT is 10 (39ms)
```

3.3.6. Should revert when the USDT amount is 100 as not enough tokens in the pre-sale

Type: Functional, Negative

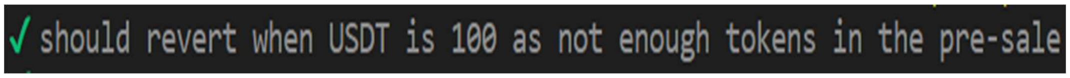
Description: This test case verifies that the lock function reverts when the USDT amount is 100, which is more than the available tokens in the pre-sale.

Steps:

- Deploy the presale contract and create a pre-sale event with a limited number of tokens.
- Call the lock function with a valid EventID and a USDT amount of 100.

Expected Outcome: The transaction should be reverted with the error message "ERR-3".

Output:



```
✓ should revert when USDT is 100 as not enough tokens in the pre-sale
```

3.3.7. Should revert when the USDT amount is zero

Type: Functional, Negative

Description: This test case verifies that the lock function reverts when the USDT amount is zero.

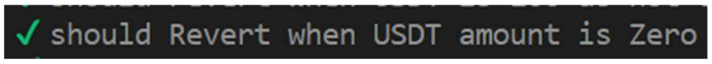
Steps:

- Deploy the presale contract and create a pre-sale event.

- Call the lock function with a valid EventID and a USDT amount of zero.

Expected Outcome: The transaction should be reverted.

Output:



```
✓ should Revert when USDT amount is Zero
```

3.3.8. Should update the idWiseInvestorList mapping

Type: Functional, Positive

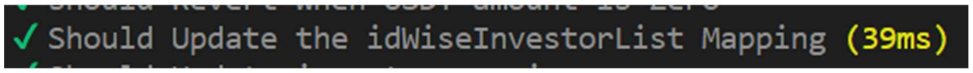
Description: This test case verifies that the idWiseInvestorList mapping is updated correctly when tokens are locked by an investor.

Steps:

- Deploy the presale contract and create a pre-sale event.
- Call the lock function with a valid EventID, token amount, and referral code (optional).

Expected Outcome: The idWiseInvestorList mapping should be updated with the investor's address.

Output:



```
✓ Should Update the idWiseInvestorList Mapping (39ms)
```

3.3.9. Should update the investor mapping

Type: Functional, Positive

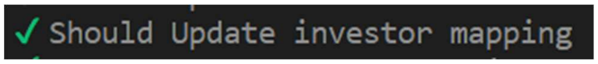
Description: This test case verifies that the investorMapping mapping is updated correctly when tokens are locked by an investor.

Steps:

- Deploy the presale contract and create a pre-sale event.
- Call the lock function with a valid EventID, token amount, and referral code (optional).

Expected Outcome: The investorMapping mapping should be updated with the investor's information, including the token balance, investment status, and lock time.

Output:



```
✓ Should Update investor mapping
```

3.3.10. Should update the cumulative investment mapping when only one Pre-Sale Event

Type: Functional, Positive

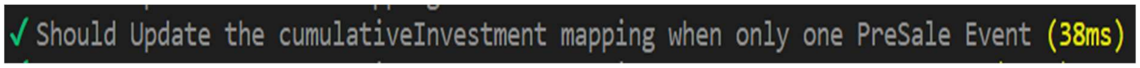
Description: This test case verifies that the cumulative investment mapping is updated correctly when tokens are locked by an investor in a single pre-sale event.

Steps:

- Deploy the presale contract and create a pre-sale event.
- Call the lock function with a valid EventID, token amount, and referral code (optional).

Expected Outcome: The cumulative Investment mapping for the investor's address should be updated with the total token amount invested in the pre-sale event.

Output:



```
✓ Should Update the cumulativeInvestment mapping when only one PreSale Event (38ms)
```

3.3.11. Should update the cumulative investment mapping when multiple Pre-Sale Events

Type: Functional, Positive

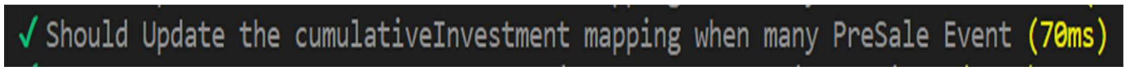
Description: This test case verifies that the cumulative investment mapping is updated correctly when tokens are locked by an investor in multiple pre-sale events.

Steps:

- Deploy the presale contract and create a pre-sale event.
- Call the lock function with a valid EventID, token amount, and referral code (optional).
- Create another pre-sale event with different parameters.
- Call the lock function with a valid EventID from the second pre-sale event, token amount, and referral code (optional).

Expected Output: The cumulative Investment mapping for the investor's address should be updated with the total token amount invested in both pre-sale events.

Output:



```
✓ Should Update the cumulativeInvestment mapping when many PreSale Event (70ms)
```

3.3.12. The sponsor should be rewarded when a valid referral code is provided

Type: Functional, Positive

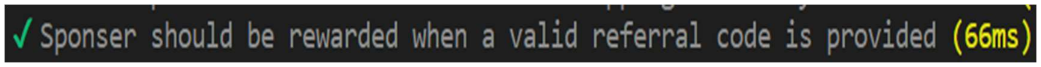
Description: This test case verifies that a sponsor is rewarded with tokens when a valid referral code is provided during token locking.

Steps:

- Deploy the presale contract and create a pre-sale event.
- Generate a referral code for an account.
- Call the lock function with a valid EventID, token amount (more than 10), and the referral code.

Expected Outcome: The sponsor should be rewarded with tokens, and the investor's balance and referral status should be updated accordingly.

Output:



```
√ Sponser should be rewarded when a valid referral code is provided (66ms)
```

3.3.13. Should not give a reward if no referral is used

Type: Functional, Positive

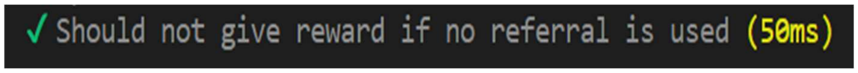
Description: This test case verifies that no reward is given when no referral code is provided during token locking.

Steps:

- Deploy the presale contract and create a pre-sale event.
- Call the lock function with a valid EventID, token amount (more than 10), and an invalid referral code.

Expected Outcome: The investor's balance and referral status should be updated accordingly, but no reward should be given.

Output:



```
√ Should not give reward if no referral is used (50ms)
```

3.4. TEST CASES FOR REFERRAL GENERATION

3.4.1. Only the owner should be able to call the referral generation function

Type: Functional, Positive

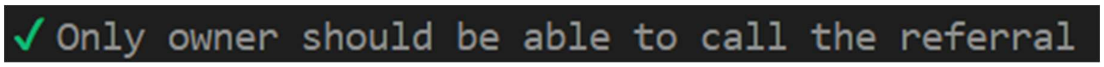
Description: This test case verifies that only the contract owner can call the generateReferal function.

Steps:

- Deploy the presale contract.
- Attempt to call the generateReferal function with a non-owner account.

Expected Outcome: The transaction should be reverted with the error message "Ownable: caller is not the owner".

Output:



```
✓ Only owner should be able to call the referral
```

3.4.2. Should revert if the sponsor already has a generated referral

Type: Functional, Negative

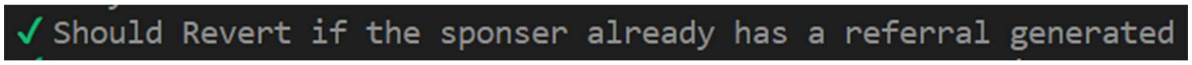
Description: This test case verifies that the generateReferal function reverts if the sponsor already has a generated referral.

Steps:

- Deploy the presale contract.
- Generate a referral for an account.
- Attempt to call the generateReferal function again with the same account address.

Expected Outcome: The transaction should be reverted with the error message "ERR-9".

Output:



```
✓ Should Revert if the sponser already has a referral generated
```

3.4.3. Should update the referralAddressToReferralCode mapping

Type: Functional, Positive

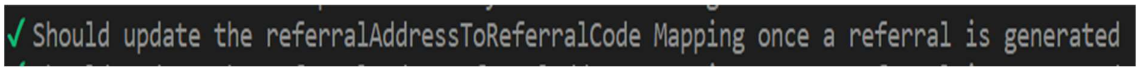
Description: This test case verifies that the referralAddressToReferralCode mapping is updated correctly when a referral is generated.

Steps:

- Deploy the presale contract.
- Generate a referral for an account.
- Retrieve the referral code from the referralAddressToReferralCode mapping for the account address.

Expected Outcome: The retrieved referral code should match the expected value (e.g., 1000).

Output:



```
✓ Should update the referralAddressToReferralCode Mapping once a referral is generated
```

3.4.4. Should update the referralCodeToReferralAddress mapping

Type: Functional, Positive

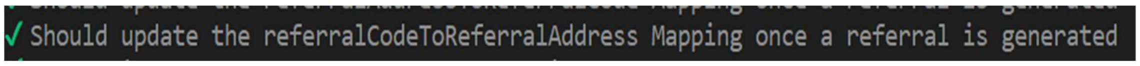
Description: This test case verifies that the referralCodeToReferralAddress mapping is updated correctly when a referral is generated.

Steps:

- Deploy the presale contract.
- Generate a referral for an account.
- Retrieve the referral address from the referralCodeToReferralAddress mapping for the generated referral code.

Expected Outcome: The retrieved referral address should match the expected account address.

Output:



```
✓ Should update the referralCodeToReferralAddress Mapping once a referral is generated
```

3.4.5. Should increment the referral code once it has been generated

Type: Functional, Positive

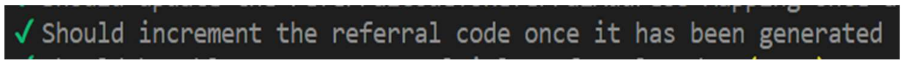
Description: This test case verifies that the referral code is incremented correctly when multiple referrals are generated.

Steps:

- Deploy the presale contract.
- Generate a referral for an account.
- Retrieve the referral code for the first generated referral.
- Generate another referral for a different account.
- Retrieve the referral code for the second generated referral and the current referral code.

Expected Outcome: The referral code for the second generated referral should be incremented by one compared to the first referral code, and the current referral code should be incremented accordingly.

Output:



```
✓ Should increment the referral code once it has been generated
```

3.4.6. Should be able to generate multiple referral codes

Type: Functional, Positive

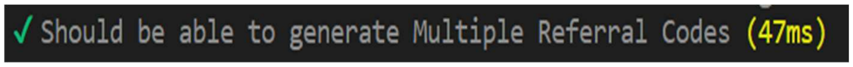
Description: This test case verifies that the contract can generate multiple referral codes for different accounts.

Steps:

- Deploy the presale contract.
- Generate referrals for multiple accounts.
- Retrieve the referral codes for each generated referral.

Expected Outcome: The retrieved referral codes should match the expected values for each respective account.

Output:



```
✓ Should be able to generate Multiple Referral Codes (47ms)
```

3.5. TEST CASES FOR WITHDRAWAL FUNCTION

3.5.1. Should allow withdrawal when tokens locked are unlocked

Type: Functional, Positive

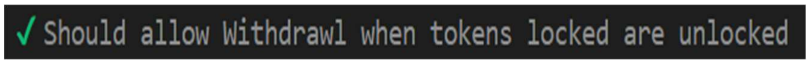
Description: This test case verifies that the withdraw function allows the withdrawal of tokens when the locked tokens are unlocked.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time.
- Call the withdraw function with the presale event ID and the desired token amount to be withdrawn.
- Retrieve the token claimed value for the owner's address in the presale event mapping.

Expected Outcome: The token claimed value should match the withdrawn token amount.

Output:



```
✓ Should allow Withdrawl when tokens locked are unlocked
```

3.5.2. Should revert when the event ID is invalid

Type: Functional, Negative

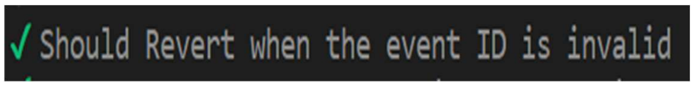
Description: This test case verifies that the withdraw function reverts when an invalid event ID is provided.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time.
- Call the withdraw function with an invalid event ID and the desired token amount to be withdrawn.

Expected Outcome: The transaction should be reverted.

Output:



```
√ Should Revert when the event ID is invalid
```

3.5.3. Should revert when the claimed amount is more than unlocked

Type: Functional, Negative.

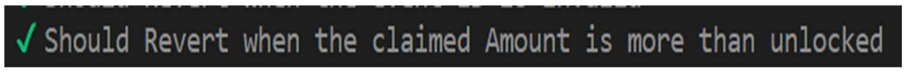
Description: This test case verifies that the withdraw function reverts when the claimed token amount is more than the unlocked tokens.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time.
- Call the withdraw function with the presale event ID and a claimed token amount that is more than the unlocked tokens.

Expected Outcome: The transaction should be reverted with the error message "ERR-8".

Output:



```
✓ Should Revert when the claimed Amount is more than unlocked
```

3.5.4. Should revert when the account is not an investor

Type: Functional, Negative

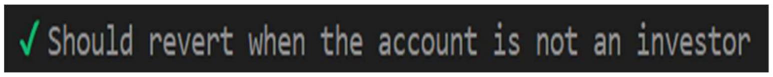
Description: This test case verifies that the withdraw function reverts when the account attempting to withdraw tokens is not an investor in the presale event.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.Increase the timestamp to be greater than the TGE time.
- Call the withdraw function with the presale event ID and the desired token amount to be withdrawn, using an account that is not an investor.

Expected Outcome: The transaction should be reverted with the error message "ERR-6".

Output:



```
√ Should revert when the account is not an investor
```

3.5.5. Should revert if the current time is not greater than the lock time

Type: Functional, Negative

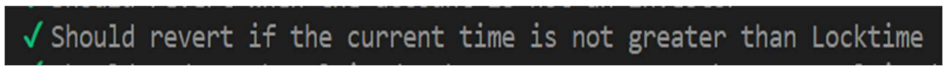
Description: This test case verifies that the withdraw function reverts when the current time is not greater than the lock time of the presale event.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event without increasing the timestamp further.
- Call the withdraw function with the presale event ID and the desired token amount to be withdrawn.

Expected Outcome: The transaction should be reverted with the error message "ERR-7".

Output:



```
✓ Should revert if the current time is not greater than Locktime
```

3.5.6. Should update the claimed tokens amount once tokens are claimed

Type: Functional, Positive

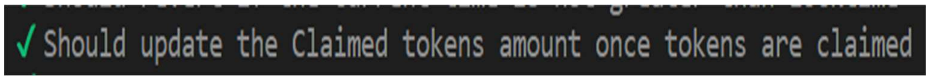
Description: This test case verifies that the withdraw function updates the claimed tokens amount correctly when tokens are claimed.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time.
- Call the withdraw function with the presale event ID and the desired token amount to be withdrawn.
- Retrieve the token claimed value for the owner's address in the presale event mapping.

Expected Outcome: The token claimed value should match the withdrawn token amount.

Output:



```
✓ Should update the Claimed tokens amount once tokens are claimed
```

3.5.7. Should emit an event once the withdrawal is completed

Type: Functional, Positive

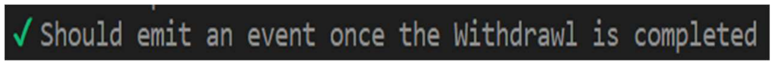
Description: This test case verifies that the withdraw function emits the expected event when the withdrawal is successfully completed.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time.
- Call the withdraw function with the presale event ID and the desired token amount to be withdrawn.

Expected Outcome: The TokenWithdrawn event should be emitted.

Output:



```
✓ Should emit an event once the Withdrawl is completed
```

3.5.8. Emitted parameters must match actual parameters

Type: Functional, Positive

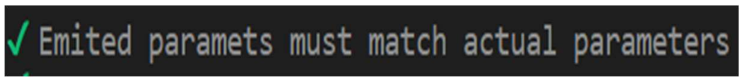
Description: This test case verifies that the parameters emitted in the TokenWithdrawn event match the actual parameters of the withdraw function.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time.
- Call the withdraw function with the presale event ID and the desired token amount to be withdrawn.
- Retrieve the emitted event from the transaction receipt.
- Verify that the emitted event has the correct event name and parameters.

Expected Outcome: The emitted event should have the event name "TokenWithdrawn" and the correct event parameters matching the presale event ID and withdrawn token amount.

Output:



```
✓ Emited paramets must match actual parameters
```

3.5.9. Should transfer the tokens to the investor's address when withdrawal is successful

Type: Functional, Positive

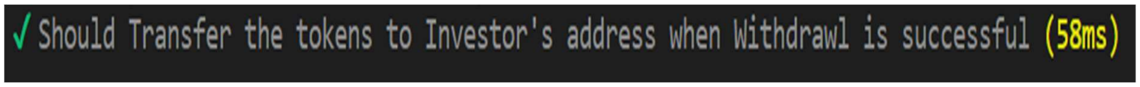
Description: This test case verifies that the withdraw function transfers the withdrawn tokens to the investor's address when the withdrawal is successful.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time.
- Call the withdraw function with the presale event ID and the desired token amount to be withdrawn, using the account of an investor.
- Check the balance of the investor's address to verify that the tokens have been transferred.

Expected Outcome: The balance of the investor's address should be equal to the withdrawn token amount.

Output:



```
√ Should Transfer the tokens to Investor's address when Withdrawl is successful (58ms)
```

3.6. TEST CASES UNLOCK TOKEN FUNCTION

3.6.1. Should unlock all tokens 10 minutes after the presale

Type: Functional, Positive

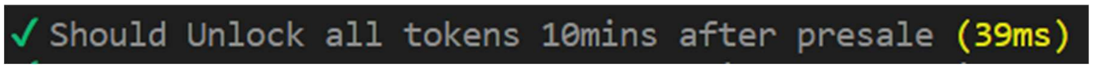
Description: This test case verifies that all tokens are unlocked 10 minutes after the presale event.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time plus 10 minutes.
- Call the unlockedTokens function with the presale event ID and the owner's address.
- Retrieve the unlocked token amount for the owner's address in the presale event mapping.

Expected Outcome: The unlocked token amount should be equal to the total token amount (10 in this case).

Output:



```
✓ Should Unlock all tokens 10mins after presale (39ms)
```

3.6.2. Should unlock 1/4th token after 3 minutes of TGE timestamp

Type: Functional, Positive

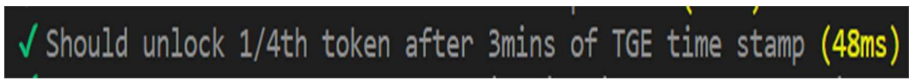
Description: This test case verifies that 1/4th of the tokens are unlocked after 3 minutes of the TGE timestamp.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time plus 3 minutes.
- Call the unlockedTokens function with the presale event ID and the owner's address.
- Retrieve the unlocked token amount for the owner's address in the presale event mapping.

Expected Outcome: The unlocked token amount should be equal to 1/4th of the total token amount (5 in this case).

Output:



```
✓ Should unlock 1/4th token after 3mins of TGE time stamp (48ms)
```

3.6.3. Should unlock 1/2 of the tokens if the time is between 3 and 6 minutes after the TGE

Type: Functional, Positive

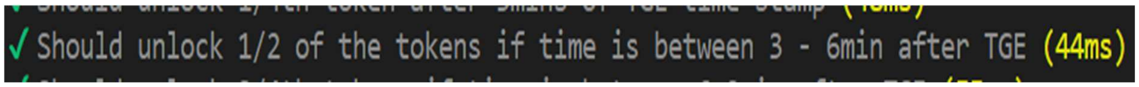
Description: This test case verifies that 1/2 of the tokens are unlocked if the time is between 3 and 6 minutes after the TGE timestamp.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time plus 3 minutes.
- Call the unlockedTokens function with the presale event ID and the owner's address.
- Retrieve the unlocked token amount for the owner's address in the presale event mapping.

Expected Outcome: The unlocked token amount should be equal to 1/2 of the total token amount (10 in this case).

Output:



```
√ Should unlock 1/2 of the tokens if time is between 3 - 6min after TGE (44ms)
```

3.6.4. Should unlock 3/4th tokens if the time is between 6 and 9 minutes after the TGE.

Type: Functional, Positive

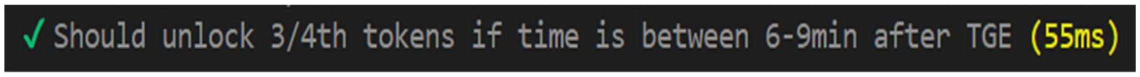
Description: This test case verifies that 3/4th of the tokens is unlocked if the time is between 6 and 9 minutes after the TGE timestamp.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time plus 6 minutes.
- Call the unlockedTokens function with the presale event ID and the owner's address.
- Retrieve the unlocked token amount for the owner's address in the presale event mapping.

Expected Outcome: The unlocked token amount should be equal to 3/4th of the total token amount (15 in this case).

Output:



```
✓ Should unlock 3/4th tokens if time is between 6-9min after TGE (55ms)
```

3.7. FIND CATEGORY TEST CASES

The test cases for the find category function are given below.

3.7.1. Should find the category if the investor has invested

Type: Functional, Positive

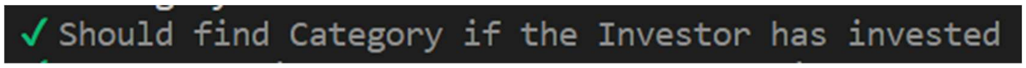
Description: This test case verifies that the correct category is found for an investor who has invested.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time plus 1000.
- Call the findCategory function with the presale event ID and the owner's address.
- Retrieve the category value returned by the function.

Expected Outcome: The category value should be greater than 0.

Output:



```
✓ Should find Category if the Investor has invested
```

3.7.2. Investors with <=500 tokens should be in Category 1

Type: Functional, Positive

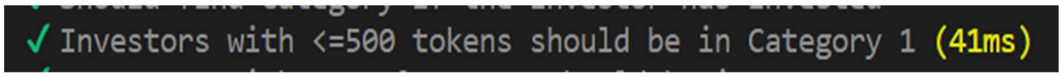
Description: This test case verifies that investors with <=500 tokens are assigned to Category 1.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time plus 1000.
- Call the find Category function with the presale event ID and the owner's address.
- Retrieve the category value returned by the function.

Expected Outcome: The category value should be equal to 1.

Output:



```
✓ Investors with <=500 tokens should be in Category 1 (41ms)
```

3.7.3. Investors with >500 & <=1000 tokens should be in Category 2

Type: Functional, Positive

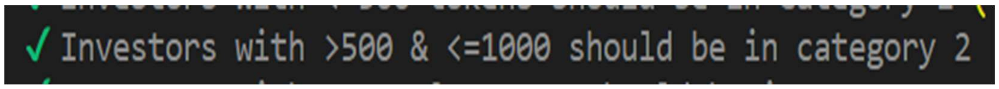
Description: This test case verifies that investors with >500 and <=1000 tokens are assigned to Category 2.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time plus 1000.
- Call the findCategory function with the presale event ID and the owner's address.
- Retrieve the category value returned by the function.

Expected Outcome: The category value should be equal to 2.

Output:



```
✓ Investors with >500 & <=1000 should be in category 2
```

3.7.4. Investors with >1000 & <=5000 tokens should be in Category 3

Type: Functional, Positive

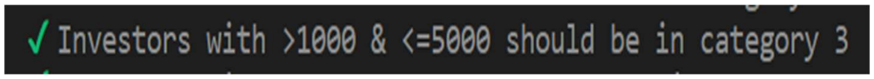
Description: This test case verifies that investors with >1000 and <=5000 tokens are assigned to Category 3.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time plus 1000.
- Call the findCategory function with the presale event ID and the owner's address.
- Retrieve the category value returned by the function.

Expected Outcome: The category value should be equal to 3.

Output:



```
✓ Investors with >1000 & <=5000 should be in category 3
```

3.7.5. Investors with >5000 & <=10000 tokens should be in Category 4

Type: Functional, Positive

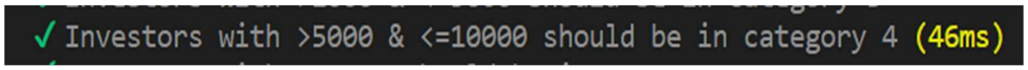
Description: This test case verifies that investors with >5000 and <=10000 tokens are assigned to Category 4.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time plus 1000.
- Call the findCategory function with the presale event ID and the owner's address.
- Retrieve the category value returned by the function.

Expected Outcome: The category value should be equal to 4.

Output:



```
✓ Investors with >5000 & <=10000 should be in category 4 (46ms)
```

3.7.6. Investors with >10000 tokens should be in Category 5

Type: Functional, Positive

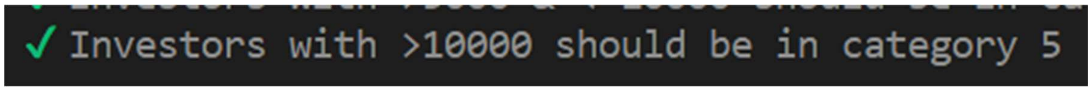
Description: This test case verifies that investors with >10000 tokens are assigned to Category 5.

Steps:

- Deploy the Haawks ERC20 token contract.
- Deploy the presale contract with a specified token address and TGE time.
- Approve the presale contract to spend a certain amount of tokens on behalf of the owner.
- Increase the timestamp to be greater than the start time of the presale event.
- Lock the tokens for the presale event.
- Increase the timestamp to be greater than the TGE time plus 1000.
- Call the findCategory function with the presale event ID and the owner's address.
- Retrieve the category value returned by the function.

Expected Outcome: The category value should be equal to 5.

Output:



```
✓ Investors with >10000 should be in category 5
```

4. CONCLUSION

The test cases for the presale contract have been designed to ensure the correct functionality of various features related to the presale event. These test cases cover Creating presale, locking tokens, withdrawing tokens, unlocking tokens, Generating Referral and finding investor categories based on their token holdings, and ensuring the accurate allocation of categories. By running these test cases, we can verify that the contract functions as intended and that the expected outcomes are achieved.

The test cases include positive scenarios where the contract should behave correctly, based on the defined business logic, and negative scenarios where the contract should revert.

Each test case follows a structured approach, deploying the necessary contracts, simulating the required conditions, and making assertions to verify the expected results.