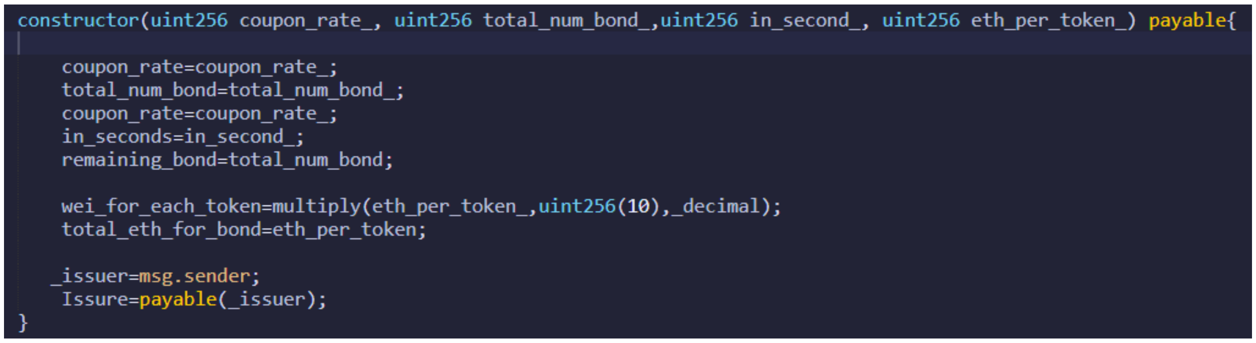
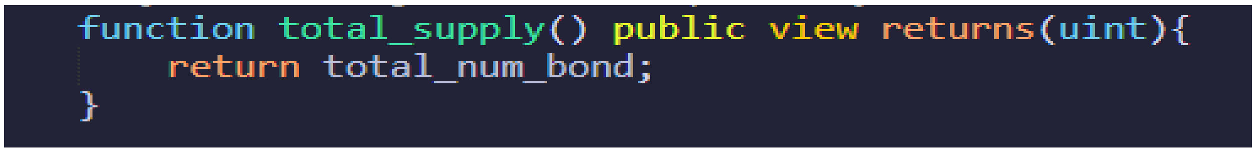
Functions



```
constructor(uint256 coupon_rate_, uint256 total_num_bond_,uint256 in_second_, uint256 eth_per_token_) payable{

    coupon_rate=coupon_rate_;
    total_num_bond=total_num_bond_;
    coupon_rate=coupon_rate_;
    in_seconds=in_second_;
    remaining_bond=total_num_bond;

    wei_for_each_token=multiply(eth_per_token_,uint256(10),_decimal);
    total_eth_for_bond=eth_per_token;

   _issuer=msg.sender;
    Issure=payable(_issuer);
}
```

Figure-1

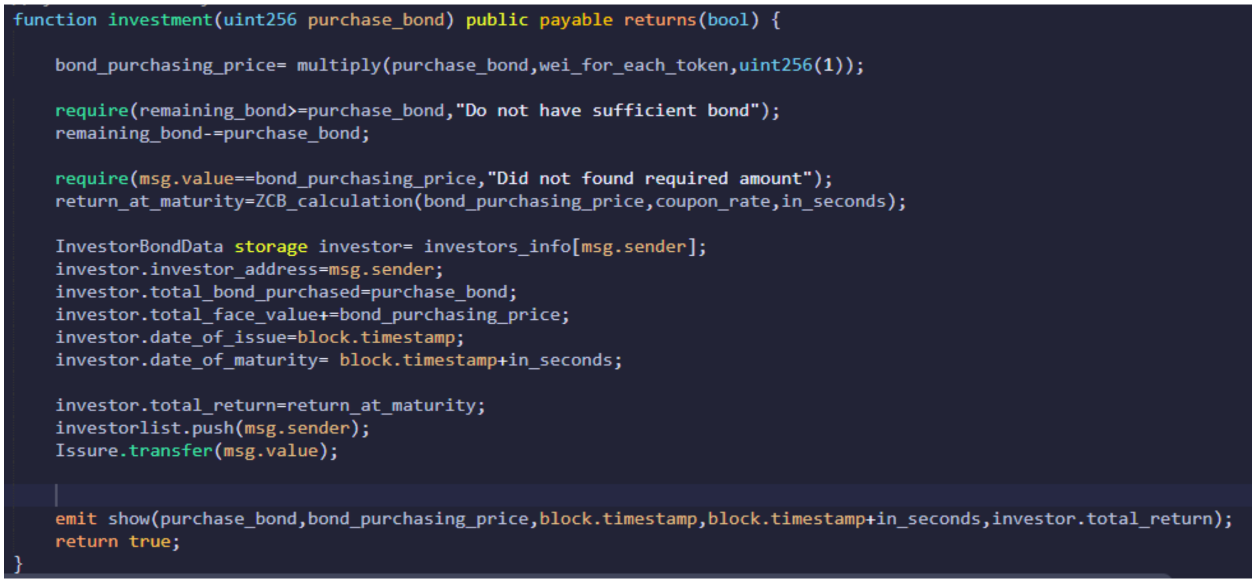
constructor-- At the time of deployment of contract, issuer have to provide information related to bond. Parameters are coupon rate, total number of bonds, duration of maturity and ether per bond. (Figure-1)



```
function total_supply() public view returns(uint){
    return total_num_bond;
}
```

Figure-2

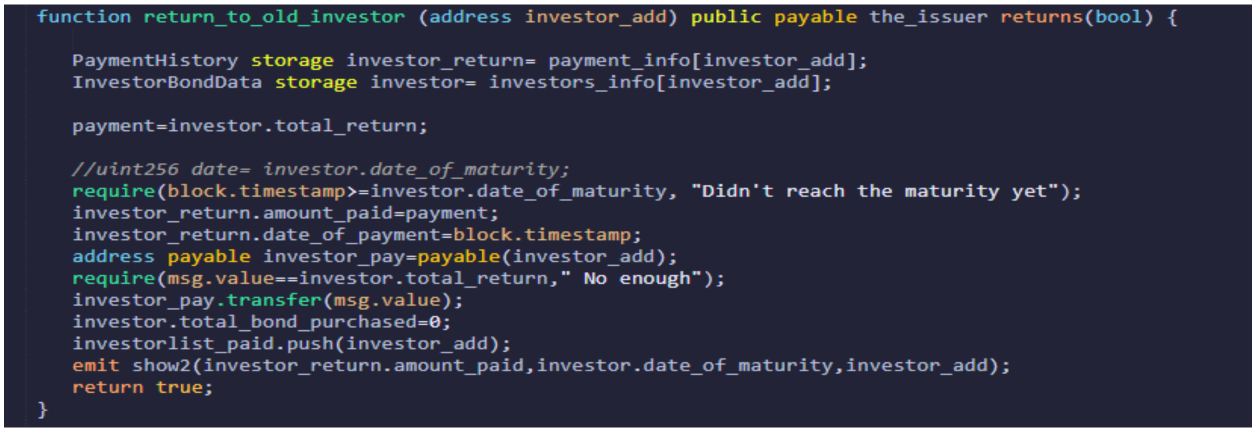
total_supply-- This function will display the quantity of bonds issued by the issuer at the initial stage of the smart contract. It has no parameters. (Figure-2)



```
function investment(uint256 purchase_bond) public payable returns(bool) {

    bond_purchasing_price= multiply(purchase_bond,wei_for_each_token,uint256(1));

    require(remaining_bond>=purchase_bond,"Do not have sufficient bond");
    remaining_bond-=purchase_bond;

    require(msg.value==bond_purchasing_price,"Did not found required amount");
    return_at_maturity=ZCB_calculation(bond_purchasing_price,coupon_rate,in_seconds);

    InvestorBondData storage investor= investors_info[msg.sender];
    investor.investor_address=msg.sender;
    investor.total_bond_purchased=purchase_bond;
    investor.total_face_value+=bond_purchasing_price;
    investor.date_of_issue=block.timestamp;
    investor.date_of_maturity= block.timestamp+in_seconds;

    investor.total_return=return_at_maturity;
    investorlist.push(msg.sender);
    Issure.transfer(msg.value);

    emit show(purchase_bond,bond_purchasing_price,block.timestamp,block.timestamp+in_seconds,investor.total_return);
    return true;
}
```

Figure- 3

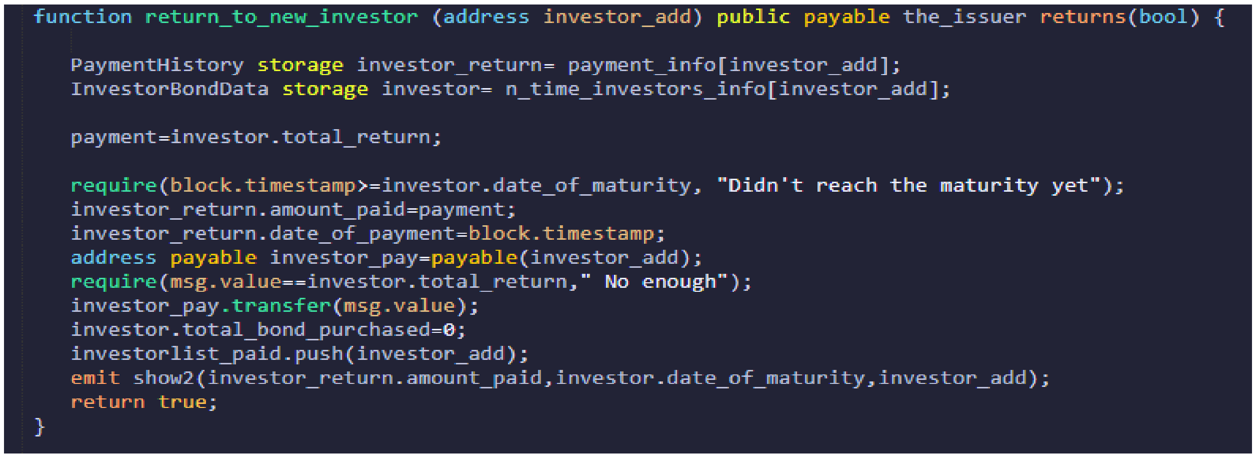
investment-- before investing in a bond, investors have to specify the number of bonds he or she wants to purchase from the issuer, using this function. This number of bonds is used as a parameter of this “investment” function. (Figure-3)



```
function return_to_old_investor (address investor_add) public payable the_issuer returns(bool) {

   PaymentHistory storage investor_return= payment_info[investor_add];
   InvestorBondData storage investor= investors_info[investor_add];

   payment=investor.total_return;

   //uint256 date= investor.date_of_maturity;
   require(block.timestamp>=investor.date_of_maturity, "Didn't reach the maturity yet");
   investor_return.amount_paid=payment;
   investor_return.date_of_payment=block.timestamp;
   address payable investor_pay=payable(investor_add);
   require(msg.value==investor.total_return," No enough");
   investor_pay.transfer(msg.value);
   investor.total_bond_purchased=0;
   investorlist_paid.push(investor_add);
   emit show2(investor_return.amount_paid,investor.date_of_maturity,investor_add);
   return true;
}
```

Figure-4

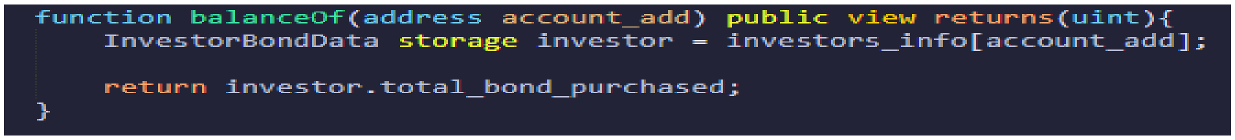
return_to_old_investor-- This function will be called to return the future value of the investment. Issuers can send ether to those investors who bought the bonds directly from the issuer by using this function. This function needs the investor address as the parameter. Transactions using this function can only be made by the issuer. (Figure-4)



```
function return_to_new_investor (address investor_add) public payable the_issuer returns(bool) {

   PaymentHistory storage investor_return= payment_info[investor_add];
   InvestorBondData storage investor= n_time_investors_info[investor_add];

   payment=investor.total_return;

   require(block.timestamp>=investor.date_of_maturity, "Didn't reach the maturity yet");
   investor_return.amount_paid=payment;
   investor_return.date_of_payment=block.timestamp;
   address payable investor_pay=payable(investor_add);
   require(msg.value==investor.total_return," No enough");
   investor_pay.transfer(msg.value);
   investor.total_bond_purchased=0;
   investorlist_paid.push(investor_add);
   emit show2(investor_return.amount_paid,investor.date_of_maturity,investor_add);
   return true;
}
```

Figure-5

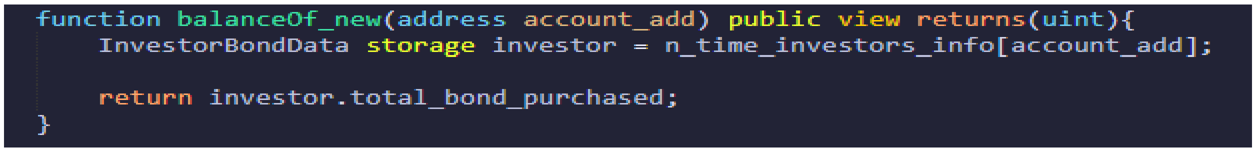
return_to_new_investor-- Similar to the previous function, this will also return the future value to the investor and can only be used by the issuer. But it will return to those investors who did not buy bonds directly from the issuer , rather from other investors.(Figure-5)



```
function balanceOf(address account_add) public view returns(uint){
     InvestorBondData storage investor = investors_info[account_add];

     return investor.total_bond_purchased;
}
```

Figure-6

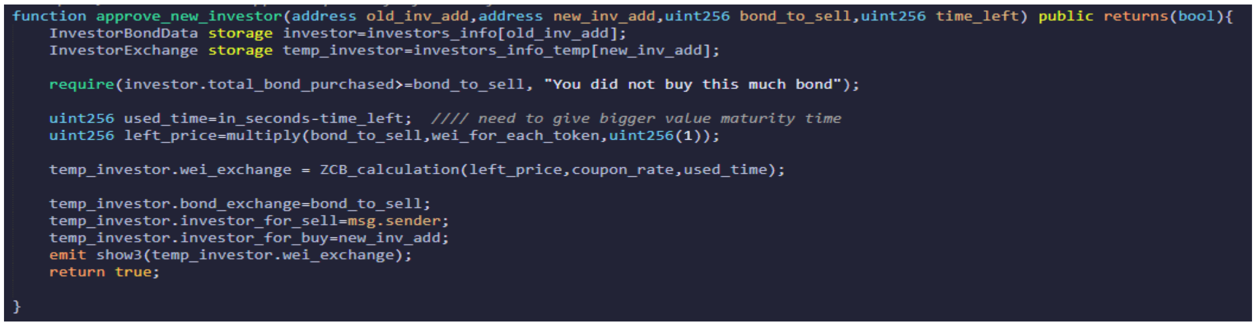
balanceOf-- Called to check the amount of bond an investor holds. Applicable for investors who issued the bonds from the issuer. Need the address of the investor as the parameter. (Figure-6)



```
function balanceOf_new(address account_add) public view returns(uint){
    InvestorBondData storage investor = n_time_investors_info[account_add];

    return investor.total_bond_purchased;
}
```

Figure-7

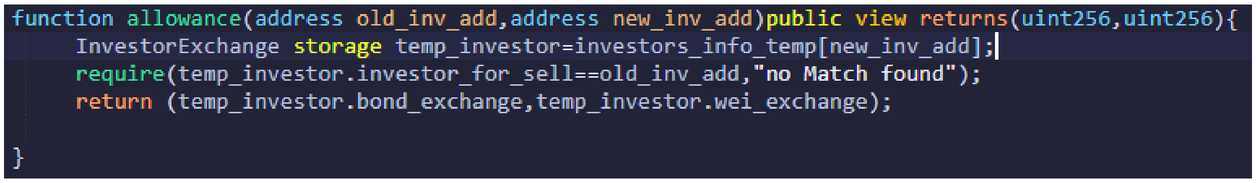
balanceOf_new-- Check the bond quantity of an investor who bought the bond from another investor. Also take a single parameter of an investor's account.(Figure-7)



```
function approve_new_investor(address old_inv_add,address new_inv_add,uint256 bond_to_sell,uint256 time_left) public returns(bool){
    InvestorBondData storage investor=investors_info[old_inv_add];
    InvestorExchange storage temp_investor=investors_info_temp[new_inv_add];

    require(investor.total_bond_purchased>=bond_to_sell, "You did not buy this much bond");

    uint256 used_time=in_seconds-time_left;  //// need to give bigger value maturity time
    uint256 left_price=multiply(bond_to_sell,wei_for_each_token,uint256(1));

    temp_investor.wei_exchange = ZCB_calculation(left_price,coupon_rate,used_time);

    temp_investor.bond_exchange=bond_to_sell;
    temp_investor.investor_for_sell=msg.sender;
    temp_investor.investor_for_buy=new_inv_add;
    emit show3(temp_investor.wei_exchange);
    return true;

}
```

Figure-8

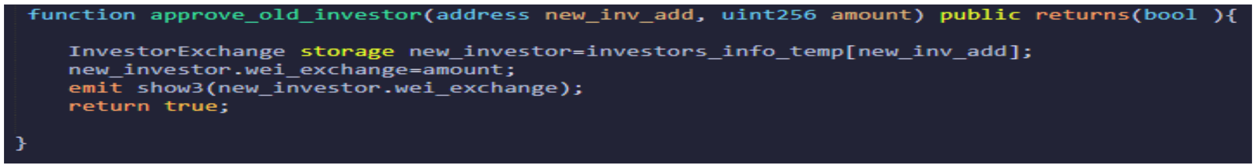
approve_new_investor-- To sell a purchased bond, investors have to call this function. It will take the address of the new investor, address of the old investor, the amount of bond for sale and time left for maturity of those bonds. This will assure how much bond will be allocated to new investors from the investor who wants to sell the bonds.



```
function allowance(address old_inv_add,address new_inv_add)public view returns(uint256,uint256){
    InvestorExchange storage temp_investor=investors_info_temp[new_inv_add];
    require(temp_investor.investor_for_sell==old_inv_add,"no Match found");
    return (temp_investor.bond_exchange,temp_investor.wei_exchange);

}
```

Figure-9

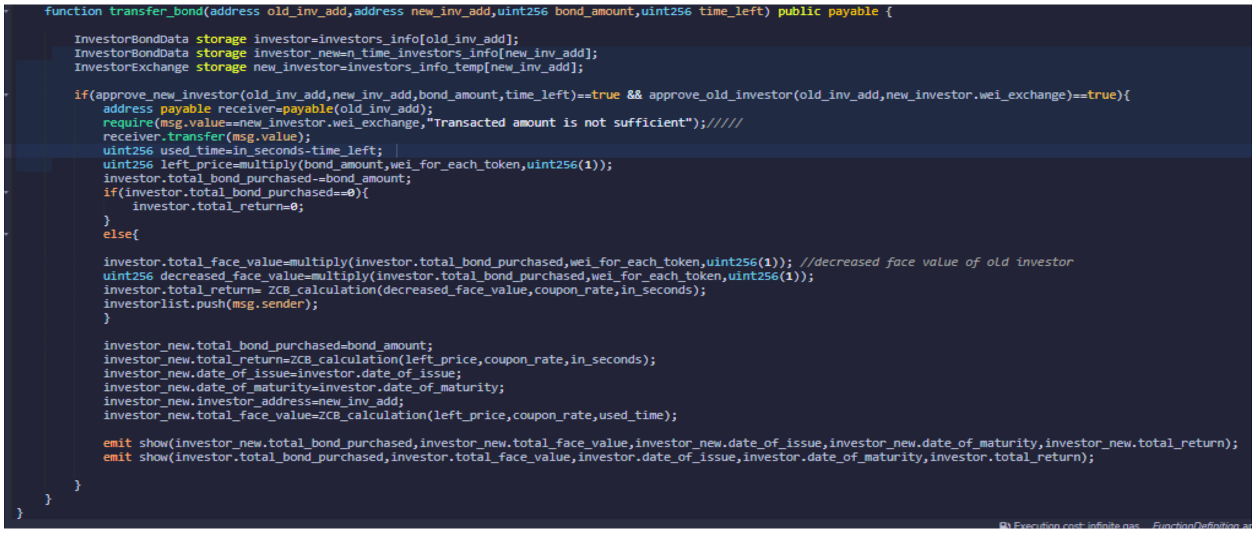
allowance-- Provided for a new investor to check how much bond is allocated for him or her by another investor and also the amount to pay to those selling investors.
It takes the address of the old investor (investor) and the address of the new investor(buyer). (Figure-9)



```
function approve_old_investor(address new_inv_add, uint256 amount) public returns(bool ){

    InvestorExchange storage new_investor=investors_info_temp[new_inv_add];
    new_investor.wei_exchange=amount;
    emit show3(new_investor.wei_exchange);
    return true;
}
```

Figure-10

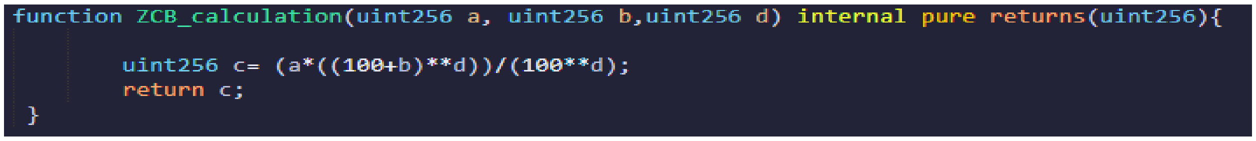
approve_old_investor-- Ensures the purchasing price the new investor wants to provide to the old investor, according to “allowance” function. The two parameters are new investor address and purchase price for the bonds from the old investor. (Figure-10)



```
function transfer_bond(address old_inv_add,address new_inv_add,uint256 bond_amount,uint256 time_left) public payable {

    InvestorBondData storage investor=investors_info[old_inv_add];
    InvestorBondData storage investor_new=n_time_investors_info[new_inv_add];
    InvestorExchange storage new_investor=investors_info_temp[new_inv_add];

    if(approve_new_investor(old_inv_add,new_inv_add,bond_amount,time_left)==true && approve_old_investor(old_inv_add,new_investor.wei_exchange)==true){
        address payable receiver=payable(old_inv_add);
        require(msg.value==new_investor.wei_exchange,"Transacted amount is not sufficient");/////
        receiver.transfer(msg.value);
        uint256 used_time=in_seconds-time_left;
        uint256 left_price=multiply(bond_amount,wei_for_each_token,uint256(1));
        investor.total_bond_purchased-=bond_amount;
        if(investor.total_bond_purchased==0){
            investor.total_return=0;
        }
        else{

        investor.total_face_value=multiply(investor.total_bond_purchased,wei_for_each_token,uint256(1)); //decreased face value of old investor
        uint256 decreased_face_value=multiply(investor.total_bond_purchased,wei_for_each_token,uint256(1));
        investor.total_return= ZCB_calculation(decreased_face_value,coupon_rate,in_seconds);
        investorlist.push(msg.sender);
        }

        investor_new.total_bond_purchased=bond_amount;
        investor_new.total_return=ZCB_calculation(left_price,coupon_rate,in_seconds);
        investor_new.date_of_issue=investor.date_of_issue;
        investor_new.date_of_maturity=investor.date_of_maturity;
        investor_new.investor_address=new_inv_add;
        investor_new.total_face_value=ZCB_calculation(left_price,coupon_rate,used_time);

        emit show(investor_new.total_bond_purchased,investor_new.total_face_value,investor_new.date_of_issue,investor_new.date_of_maturity,investor_new.total_return);
        emit show(investor.total_bond_purchased,investor.total_face_value,investor.date_of_issue,investor.date_of_maturity,investor.total_return);

    }
  }
}
```

Figure- 11

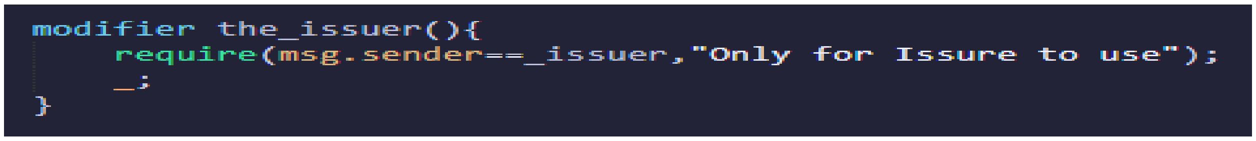
transfer_bonds-- After two investors approve within themselves to exchange bonds, the "transfer_bond" function will be called by the new investor. As parameters, users have to input 4 parameters, which are; address of old investor(seller), address of new investor(buyer), number of bonds to be exchanged and remaining time of the bond to be matured. The ether amount which is needed to transfer the bond has to be similar to exchange price provided using the "allowance" function and amount issued in "approved_old_investor" function.(Figure-11)



```
function ZCB_calculation(uint256 a, uint256 b,uint256 d) internal pure returns(uint256){

        uint256 c= (a*((100+b)**d))/(100**d);
        return c;
 }
```

Figure-12

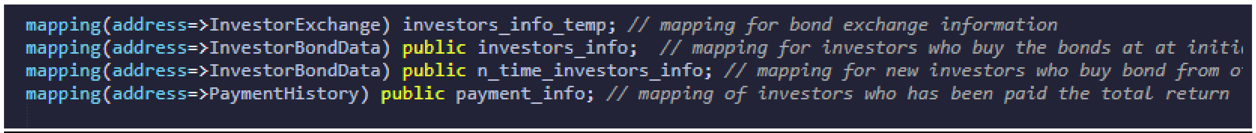
zcb_calculation-- It's an internal function to calculate the future value of an investment in zero coupon bond. Parameters are total face value, coupon rate and duration of maturity. As Solidity programming language do not support float point number until now, we have to modify the calculation process of the original formula to calculate the coupon return.(Figure-12)



```
 modifier the_issuer(){
     require(msg.sender==_issuer,"Only for Issure to use");
     _;
 }
```

Figure- 13

the_issuer- This is a modifier to ensure that a function containing this can only be executed by the issuer.(Figure-13)



```
mapping(address=>InvestorExchange) investors_info_temp; // mapping for bond exchange information
mapping(address=>InvestorBondData) public investors_info;  // mapping for investors who buy the bonds at at initi
mapping(address=>InvestorBondData) public n_time_investors_info; // mapping for new investors who buy bond from o
mapping(address=>PaymentHistory) public payment_info; // mapping of investors who has been paid the total return
```

Figure- 14