pragma solidity ^0.4.24;

/\*\*

\* @title SafeMath

\* @dev Math operations with safety checks that throw on error

\*/

library SafeMath {

function mul(uint256 a, uint256 b) internal pure returns (uint256) {

uint256 c = a \* b;

assert(a == 0 || c / a == b);

return c;

}

function div(uint256 a, uint256 b) internal pure returns (uint256) {

// assert(b > 0); // Solidity automatically throws when dividing by 0 uint256 c = a / b;

uint256 c = a / b;

// assert(a == b \* c + a % b); // There is no case in which this doesn't hold

return c;

}

function sub(uint256 a, uint256 b) internal pure returns (uint256) {

assert(b <= a);

return a - b;

}

function add(uint256 a, uint256 b) internal pure returns (uint256) {

uint256 c = a + b;

assert(c >= a);

return c;

}

}

/\*\*

\* @title Crowdsale

\* @dev Crowdsale is a base contract for managing a token crowdsale.

\* Crowdsales have a start and end timestamps, where investors can make

\* token purchases and the crowdsale will assign them tokens based

\* on a token per ETH rate. Funds collected are forwarded

to a wallet

\* as they arrive.

\*/

interface token { function transfer(address, uint) external; }

contract Crowdsale {

using SafeMath for uint256;

// uint256 durationInMinutes;

// address where funds are collected

address public wallet;

// token address

address public addressOfTokenUsedAsReward;

uint256 public price = 00;

token tokenReward;

// amount of raised money in wei

uint256 public weiRaised;

/\*\*

\* event for token purchase logging

\* @param purchaser who paid for the tokens

\* @param beneficiary who got the tokens

\* @param value weis paid for purchase

\* @param amount amount of tokens purchased

\*/

event TokenPurchase(address indexed purchaser, address indexed beneficiary, uint256 value, uint256 amount);

constructor () public {

//You will change this to your wallet where you need the ETH

wallet = 0xb659871eA0bF341D3DE8AD768873bed169A8B87a ;

// this the token contract address

addressOfTokenUsedAsReward = 0x90ac96037fba8d5e8142271f15bbaa9604cae25f ;

tokenReward = token(addressOfTokenUsedAsReward);

}

bool public started = true;

function stopSale() public {

require (msg.sender == wallet);

started = false;

}

// fallback function can be used to buy tokens

function () payable public {

buyTokens(msg.sender);

}

// low level token purchase function

function buyTokens(address beneficiary) payable public {

require(beneficiary != 0x0);

require(validPurchase());

uint256 weiAmount = msg.value;

require (weiAmount >= 5\*\*16);

// calculate token amount to be sent

uint256 tokens = (weiAmount/10\*\*10) \* price;// weiamount/(10\*\*(18-decimals)) \* price

// update state

weiRaised = weiRaised.add(weiAmount);

tokenReward.transfer(beneficiary, tokens);

emit TokenPurchase(msg.sender, beneficiary, weiAmount, tokens);

forwardFunds();

}

// send ether to the fund collection wallet

// override to create custom fund forwarding mechanisms

function forwardFunds() internal {

wallet.transfer(msg.value);

}

// @return true if the transaction can buy tokens

function validPurchase() internal constant returns (bool) {

bool withinPeriod = started;

bool nonZeroPurchase = msg.value != 0;

return withinPeriod && nonZeroPurchase;

}

function withdrawTokens(uint256 \_amount) public {

require (msg.sender==wallet);

tokenReward.transfer(wallet,\_amount);

}

}