Lecture 36 - Example Contract

A comparative example: An Auction Contraact

```
# Open Auction

# Auction params
# Beneficiary receives money from the highest bidder
beneficiary: public(address)
auctionStart: public(uint256)
auctionEnd: public(uint256)

# Current state of auction
highestBidder: public(address)
highestBid: public(uint256)

# Set to true at the end, disallows any change
ended: public(bool)

# Keep track of refunded bids so we can follow the withdraw pattern
pendingReturns: public(HashMap[address, uint256])

# Create a simple auction with `_auction_start` and
# `_bidding_time` seconds bidding time on behalf of the
# beneficiary address `_beneficiary`.
@external
def __init__(_beneficiary: address, _auction_start: uint256, _bidding_time: uint256):
    self.beneficiary = _beneficiary
    self.auctionStart = _auction_start  # auction start time can be in the past, present or future
    self.auctionEnd = self.auctionStart + _bidding_time
    assert block.timestamp < self.auctionEnd # auction end time should be in the future

# Bid on the auction with the value sent
# together with this transaction.
# The value will only be refunded if the
# auction is not won.
@external
@payable
def bid():
    # Check if bidding period has started.
    assert block.timestamp >= self.auctionStart
    # Check if bidding period is over.
    assert block.timestamp < self.auctionEnd
    # Check if bid is high enough
    assert msg.value > self.highestBid
    # Track the refund for the previous high bidder
    self.pendingReturns[self.highestBidder] += self.highestBid
    # Track new high bid
    self.highestBidder = msg.sender
    self.highestBid = msg.value

# Withdraw a previously refunded bid. The withdraw pattern is
# used here to avoid a security issue. If refunds were directly
# sent as part of bid(), a malicious bidding contract could block
# those refunds and thus block new higher bids from coming in.
@external
def withdraw():
    pending_amount: uint256 = self.pendingReturns[msg.sender]
    self.pendingReturns[msg.sender] = 0
    send(msg.sender, pending_amount)

```

```
# End the auction and send the highest bid
# to the beneficiary.
@external
def endAuction():
    # It is a good guideline to structure functions that interact
    # with other contracts (i.e. they call functions or send Ether)
    # into three phases:
    # 1. checking conditions
    # 2. performing actions (potentially changing conditions)
    # 3. interacting with other contracts
    # If these phases are mixed up, the other contract could call
    # back into the current contract and modify the state or cause
    # effects (Ether payout) to be performed multiple times.
    # If functions called internally include interaction with external
    # contracts, they also have to be considered interaction with
    # external contracts.

    # 1. Conditions
    # Check if auction endtime has been reached
    assert block.timestamp >= self.auctionEnd
    # Check if this function has already been called
    assert not self.ended

    # 2. Effects
    self.ended = True

    # 3. Interaction
    send(self.beneficiary, self.highestBid)
```

```

create table accounts (
    account_id    serial not null primary key,
    amount        number
);

create or replace function transfer ( p_to int, p_amount number )
    returns text
    as $$
DECLARE
    l_data                    text;
BEGIN
    update accounts
        set amount = amoutn + p_amount
        where acount_id = p_to
        ;
    RETURN l_data;
END;
$$ LANGUAGE plpgsql;


-- Open Auction

-- # Auction params
-- # Beneficiary receives money from the highest bidder
create table open_auction (
    open_auction_id serial not null primary key,
    beneficiary     number,    -- Account Number
    auction_start     timestamp default current_timestamp not null,
    auction_end     timestamp not null,
    highest_bidder     int default 0 not null,
    higest_bid         number default 0 not null,
    ended             boolean default false,
    updated         timestamp,
    created         timestamp default current_timestamp not null
);

-- # Keep track of refunded bids so we can follow the withdraw pattern
create table open_auction_bids (
    open_auction_bids_id serial not null primary key,
    open_auction_id     int not null references open_auction ( open_auction_id ),
    bidder                 int default 0 not null,
    bid                 number default 0 not null,
    updated             timestamp,
    created             timestamp default current_timestamp not null
);


-- # Create the auction at the start
create or replace function setup_auction ( beneficiary int, auction_start timestamp, biding_time int )
    returns text
    as $$
DECLARE
    l_data                    text;
    l_ended                    boolean;
BEGIN
    select ended
        into l_ended
        from open_auction
        ;
    if not found then
        if l_ended then
            update open_auction
                set
```

```
                    beneficiary = p_benefeciary,
                    auction_start = p_auction_start,
                    auction_end = p_auction_start + interval bidding_time,
                    highest_bidder = 0,
                    higest_bid = 0,
                    ended = false
                ;
            data = '{"status":"success"}';
        else
            raise ( 'Auction currently in progress' );
        end if;
    else
        insert into open_auction ( beneficiary, auction_start, auction_end, highest_bidder, higest_bid, ended )
            values ( p_benefeciary, p_auction_start, p_acution_start + interval bidding_time, 0, false );
        data = '{"status":"success"}';
    end if;
    RETURN l_data;
END;
$$ LANGUAGE plpgsql;



-- # Bid on the auction with the value sent together with this transaction.
-- # The value will only be refunded if the auction is not won.
create or replace function place_bid ( bid number )
    returns text
    as $$
DECLARE
    l_data                    text;
    l_open_auction_id        int;
    l_bidder                 number;
    l_bid                     number;
BEGIN

    select bidder, bid
        int l_bidder, l_bid
        from open_auction_bids
        where open_auciton_id = l_open_auction_id
          and open_auction_bids_id = (
                select max(open_auction_bids_id)
                from open_auction_bids
            );
    if not found then
        -- this is the first bid, insert
        insert into open_auciton_bids_id ( bidder, bid, open_auction_id ) values
            ( p_bidder, p_bid, l_open_auction_id );
        l_data '{"status":"bid-accepted"}';
    else
        if p_bid > l_bid then
            insert into open_auciton_bids_id ( bidder, bid, open_auction_id ) values
                ( p_bidder, p_bid, l_open_auction_id );
            l_data '{"status":"bid-accepted"}';
        else
            l_data '{"status":"bid-rejected","msg":"not highest bid"}';
        end if;
    end if;

    RETURN l_data;
END;
$$ LANGUAGE plpgsql;


-- # Withdraw a previously refunded bid. The withdraw pattern is
-- # used here to avoid a security issue. If refunds were directly
-- # sent as part of bid(), a malicious bidding contract could block
-- # those refunds and thus block new higher bids from coming in.
```

```
create or replace function widtraw_funds ( to_acct number )
    returns text
    as $$
DECLARE
    l_data                    text;
    l_highest_bidder        int;
    l_highest_bid            number;
BEGIN

    select t2.bidder, t2.bid
        into l_highest_bidder, l_highest_bid
    from open_auction_bids as t2
    where t2.open_auction_bids_id = (
            select max(open_auction_bids_id)
            from open_auction_bids
        )
        ;

    l_data = '{"status":"auction still in progress"}';
    select 'found' into l_data
        from open_auction
        where ended = true
        ;
    if found then

        select transfer ( highest_bidder, l_highest_bid )
            int l_data;
        l_data = '{"status":"success"}';

    end if;

    RETURN l_data;
END;
$$ LANGUAGE plpgsql;



-- # End the auction and send the highest bid
-- # to the beneficiary.
create or replace function end_auction ()
    returns text
    as $$
DECLARE
    l_data                    text;
BEGIN

    -- Pick out the high bid.  End the auction.
    update open_auction as t1
        set ended = true
            ( highest_bidder, higest_bid ) = (
                select t2.bidder, t2.bid
                from open_auction_bids as t2
                where t2.open_auciton_id = t1.open_auction_id
                  and t2.open_auction_bids_id = (
                        select max(open_auction_bids_id)
                        from open_auction_bids
                    )
            );
        where auction_end >= current_timestamp
        ;

    -- return bids that were not the highest bid.
    select transfer ( t2.bidder, t2.bid )
        from open_auction_bids as t2
        where t2.open_auction_bids_id < (
                select max(open_auction_bids_id)
                from open_auction_bids
            )
        ;

    RETURN l_data;
END;
$$ LANGUAGE plpgsql;



```