

Lecture 36 - Example Contract

A comparative example: An Auction Contract

```
1: # Open Auction
2:
3: # Auction params
4: # Beneficiary receives money from the highest bidder
5: beneficiary: public(address)
6: auctionStart: public(uint256)
7: auctionEnd: public(uint256)
8:
9: # Current state of auction
10: highestBidder: public(address)
11: highestBid: public(uint256)
12:
13: # Set to true at the end, disallows any change
14: ended: public(bool)
15:
16: # Keep track of refunded bids so we can follow the withdraw pattern
17: pendingReturns: public(HashMap[address, uint256])
18:
19: # Create a simple auction with `_auction_start` and
20: # `_bidding_time` seconds bidding time on behalf of the
21: # beneficiary address `_beneficiary`.
22: @external
23: def __init__(_beneficiary: address, _auction_start: uint256, _bidding_time: uint256):
24:     self.beneficiary = _beneficiary
25:     self.auctionStart = _auction_start # auction start time can be in the past, present or future
26:     self.auctionEnd = self.auctionStart + _bidding_time
27:     assert block.timestamp < self.auctionEnd # auction end time should be in the future
28:
29: # Bid on the auction with the value sent
30: # together with this transaction.
31: # The value will only be refunded if the
32: # auction is not won.
33: @external
34: @payable
35: def bid():
36:     # Check if bidding period has started.
37:     assert block.timestamp >= self.auctionStart
38:     # Check if bidding period is over.
39:     assert block.timestamp < self.auctionEnd
40:     # Check if bid is high enough
41:     assert msg.value > self.highestBid
42:     # Track the refund for the previous high bidder
43:     self.pendingReturns[self.highestBidder] += self.highestBid
44:     # Track new high bid
45:     self.highestBidder = msg.sender
46:     self.highestBid = msg.value
47:
48: # Withdraw a previously refunded bid. The withdraw pattern is
49: # used here to avoid a security issue. If refunds were directly
50: # sent as part of bid(), a malicious bidding contract could block
51: # those refunds and thus block new higher bids from coming in.
52: @external
53: def withdraw():
54:     pending_amount: uint256 = self.pendingReturns[msg.sender]
55:     self.pendingReturns[msg.sender] = 0
56:     send(msg.sender, pending_amount)
57:
```

```
58: # End the auction and send the highest bid
59: # to the beneficiary.
60: @external
61: def endAuction():
62:     # It is a good guideline to structure functions that interact
63:     # with other contracts (i.e. they call functions or send Ether)
64:     # into three phases:
65:     # 1. checking conditions
66:     # 2. performing actions (potentially changing conditions)
67:     # 3. interacting with other contracts
68:     # If these phases are mixed up, the other contract could call
69:     # back into the current contract and modify the state or cause
70:     # effects (Ether payout) to be performed multiple times.
71:     # If functions called internally include interaction with external
72:     # contracts, they also have to be considered interaction with
73:     # external contracts.
74:
75:     # 1. Conditions
76:     # Check if auction endtime has been reached
77:     assert block.timestamp >= self.auctionEnd
78:     # Check if this function has already been called
79:     assert not self.ended
80:
81:     # 2. Effects
82:     self.ended = True
83:
84:     # 3. Interaction
85:     send(self.beneficiary, self.highestBid)
```

```

1:
2: create table accounts (
3:     account_id    serial not null primary key,
4:     amount        number
5: );
6:
7: create or replace function transfer ( p_to int, p_amount number )
8:     returns text
9:     as $$
10: DECLARE
11:     l_data          text;
12: BEGIN
13:     update accounts
14:         set amount = amoutn + p_amount
15:         where account_id = p_to
16:         ;
17:     RETURN l_data;
18: END;
19: $$ LANGUAGE plpgsql;
20:
21:
22: -- Open Auction
23:
24: -- # Auction params
25: -- # Beneficiary receives money from the highest bidder
26: create table open_auction (
27:     open_auction_id serial not null primary key,
28:     beneficiary      number,      -- Account Number
29:     auction_start    timestamp default current_timestamp not null,
30:     auction_end      timestamp not null,
31:     highest_bidder    int default 0 not null,
32:     highest_bid       number default 0 not null,
33:     ended            boolean default false,
34:     updated          timestamp,
35:     created          timestamp default current_timestamp not null
36: );
37:
38: -- # Keep track of refunded bids so we can follow the withdraw pattern
39: create table open_auction_bids (
40:     open_auction_bids_id serial not null primary key,
41:     open_auction_id     int not null references open_auction ( open_auction_id ),
42:     bidder              int default 0 not null,
43:     bid                 number default 0 not null,
44:     updated             timestamp,
45:     created             timestamp default current_timestamp not null
46: );
47:
48:
49: -- # Create the auction at the start
50: create or replace function setup_auction ( beneficiary int, auction_start timestamp, bidding_time int )
51:     returns text
52:     as $$
53: DECLARE
54:     l_data          text;
55:     l_ended         boolean;
56: BEGIN
57:     select ended
58:         into l_ended
59:         from open_auction
60:         ;
61:     if not found then
62:         if l_ended then
63:             update open_auction
64:                 set

```

```

65:         beneficiary = p_benefeciary,
66:         auction_start = p_auction_start,
67:         auction_end = p_auction_start + interval bidding_time,
68:         highest_bidder = 0,
69:         highest_bid = 0,
70:         ended = false
71:     ;
72:     data = '{"status":"success"}';
73:     else
74:         raise ( 'Auction currently in progress' );
75:     end if;
76: else
77:     insert into open_auction ( beneficiary, auction_start, auction_end, highest_bidder, highest_bid, ended )
78:         values ( p_benefeciary, p_auction_start, p_auction_start + interval bidding_time, 0, false );
79:     data = '{"status":"success"}';
80: end if;
81: RETURN l_data;
82: END;
83: $$ LANGUAGE plpgsql;
84:
85:
86:
87: -- # Bid on the auction with the value sent together with this transaction.
88: -- # The value will only be refunded if the auction is not won.
89: create or replace function place_bid ( bid number )
90:     returns text
91:     as $$
92: DECLARE
93:     l_data          text;
94:     l_open_auction_id int;
95:     l_bidder        number;
96:     l_bid           number;
97: BEGIN
98:
99:     select bidder, bid
100:        int l_bidder, l_bid
101:        from open_auction_bids
102:        where open_auction_id = l_open_auction_id
103:           and open_auction_bids_id = (
104:               select max(open_auction_bids_id)
105:               from open_auction_bids
106:           );
107:     if not found then
108:         -- this is the first bid, insert
109:         insert into open_auction_bids_id ( bidder, bid, open_auction_id ) values
110:             ( p_bidder, p_bid, l_open_auction_id );
111:         l_data '{"status":"bid-accepted"}';
112:     else
113:         if p_bid > l_bid then
114:             insert into open_auction_bids_id ( bidder, bid, open_auction_id ) values
115:                 ( p_bidder, p_bid, l_open_auction_id );
116:             l_data '{"status":"bid-accepted"}';
117:         else
118:             l_data '{"status":"bid-rejected","msg":"not highest bid"}';
119:         end if;
120:     end if;
121:
122:     RETURN l_data;
123: END;
124: $$ LANGUAGE plpgsql;
125:
126:
127: -- # Withdraw a previously refunded bid. The withdraw pattern is
128: -- # used here to avoid a security issue. If refunds were directly
129: -- # sent as part of bid(), a malicious bidding contract could block
130: -- # those refunds and thus block new higher bids from coming in.

```

```
131: create or replace function widraw_funds ( to_acct number )
132:     returns text
133:     as $$
134: DECLARE
135:     l_data          text;
136:     l_highest_bidder int;
137:     l_highest_bid    number;
138: BEGIN
139:
140:     select t2.bidder, t2.bid
141:         into l_highest_bidder, l_highest_bid
142:     from open_auction_bids as t2
143:    where t2.open_auction_bids_id = (
144:        select max(open_auction_bids_id)
145:        from open_auction_bids
146:    )
147:    ;
148:
149:     l_data = '{"status":"auction still in progress"}';
150:     select 'found' into l_data
151:     from open_auction
152:    where ended = true
153:    ;
154:     if found then
155:
156:         select transfer ( highest_bidder, l_highest_bid )
157:             into l_data;
158:         l_data = '{"status":"success"}';
159:
160:     end if;
161:
162:     RETURN l_data;
163: END;
164: $$ LANGUAGE plpgsql;
165:
166:
167:
168: -- # End the auction and send the highest bid
169: -- # to the beneficiary.
170: create or replace function end_auction ()
171:     returns text
172:     as $$
173: DECLARE
174:     l_data          text;
175: BEGIN
176:
177:     -- Pick out the high bid.  End the auction.
178:     update open_auction as t1
179:        set ended = true
180:        ( highest_bidder, higest_bid ) = (
181:            select t2.bidder, t2.bid
182:          from open_auction_bids as t2
183:         where t2.open_auciton_id = t1.open_auction_id
184:           and t2.open_auction_bids_id = (
185:               select max(open_auction_bids_id)
186:             from open_auction_bids
187:           )
188:        );
189:     where auction_end >= current_timestamp
190:    ;
191:
192:     -- return bids that were not the highest bid.
193:     select transfer ( t2.bidder, t2.bid )
194:     from open_auction_bids as t2
195:    where t2.open_auction_bids_id < (
196:        select max(open_auction_bids_id)
```

```
197:         from open_auction_bids
198:     )
199: ;
200:
201:     RETURN l_data;
202: END;
203: $$ LANGUAGE plpgsql;
204:
205:
206:
```
