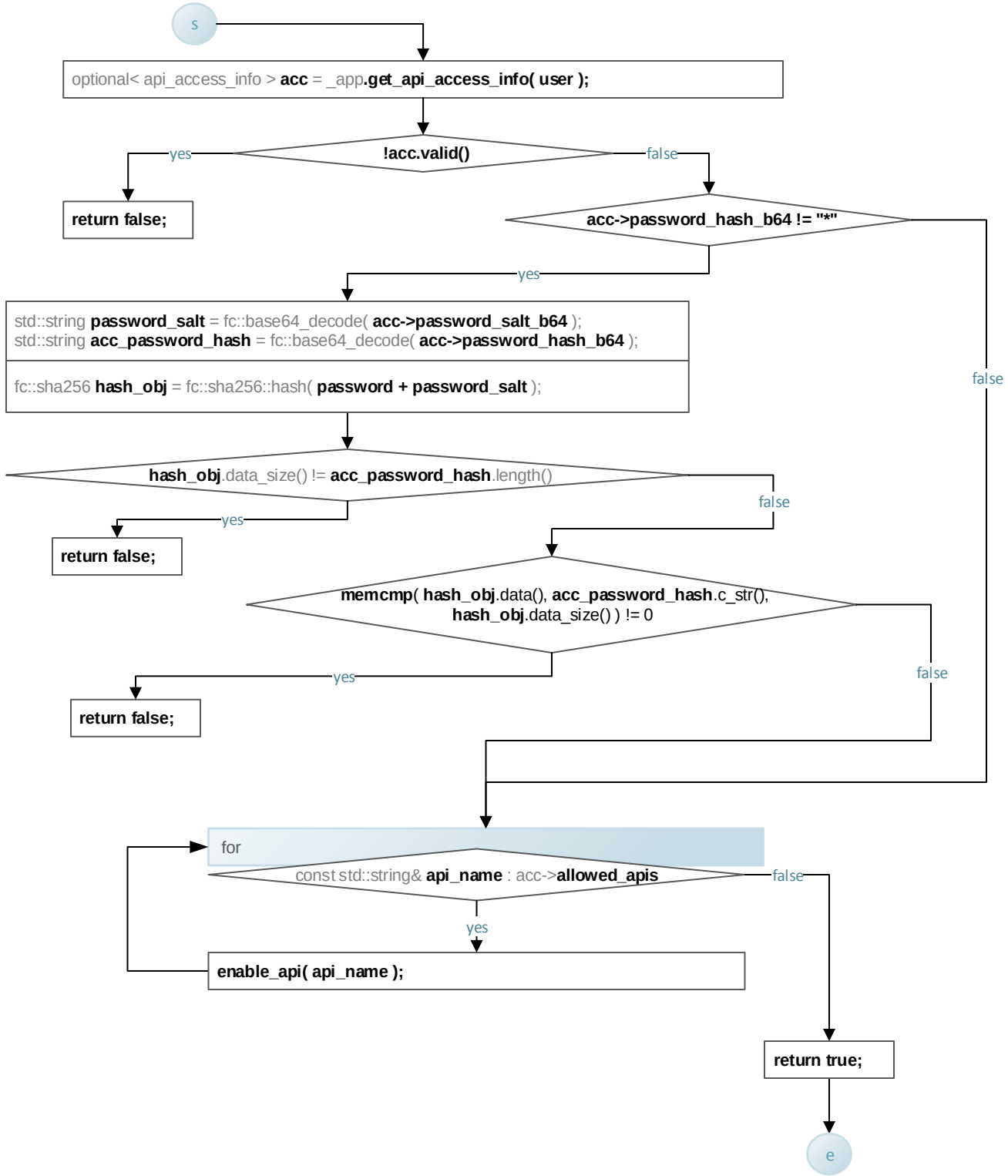
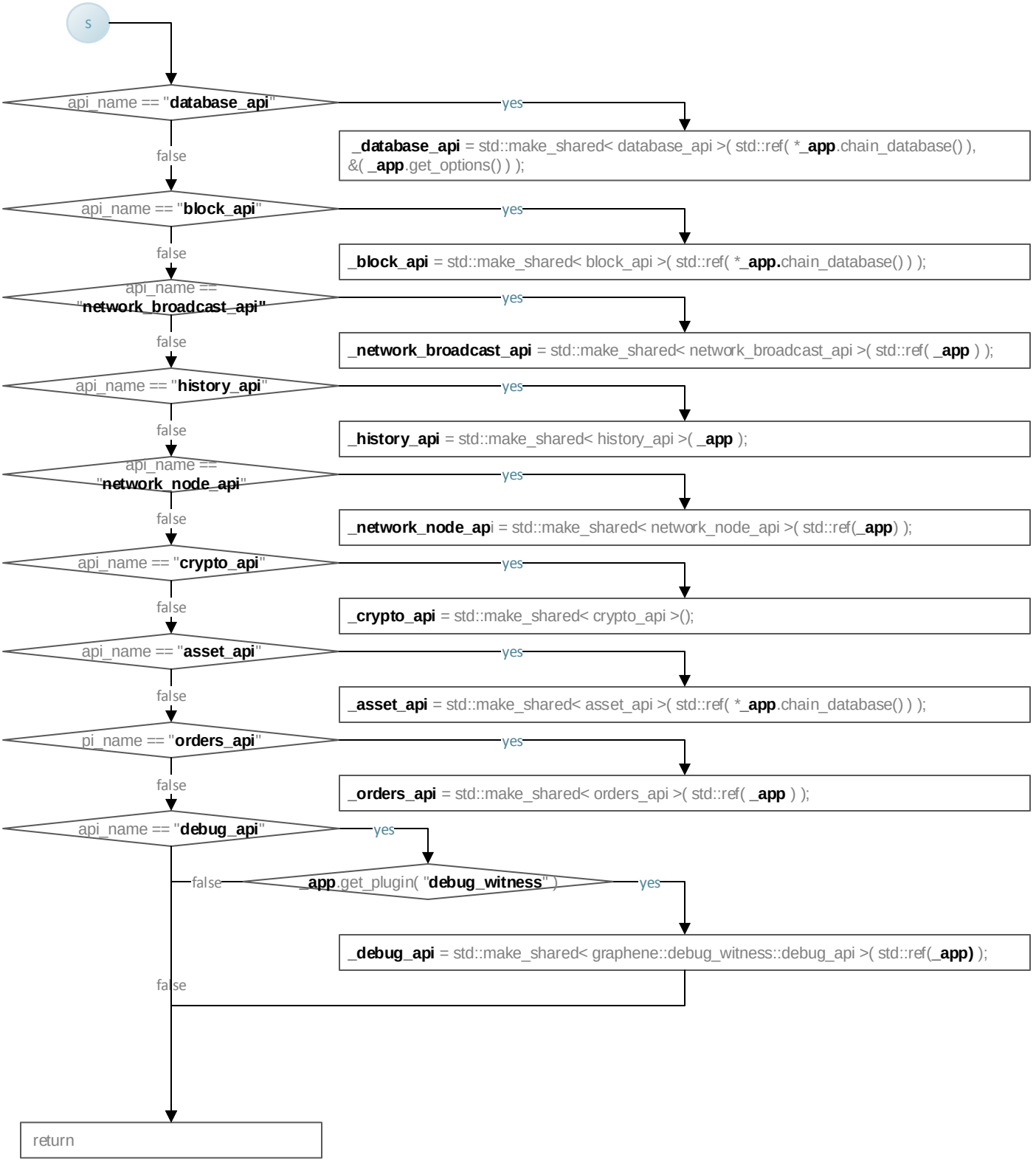
bool login_api::**login**(const string& **user**, const string& **password**)

api.cpp



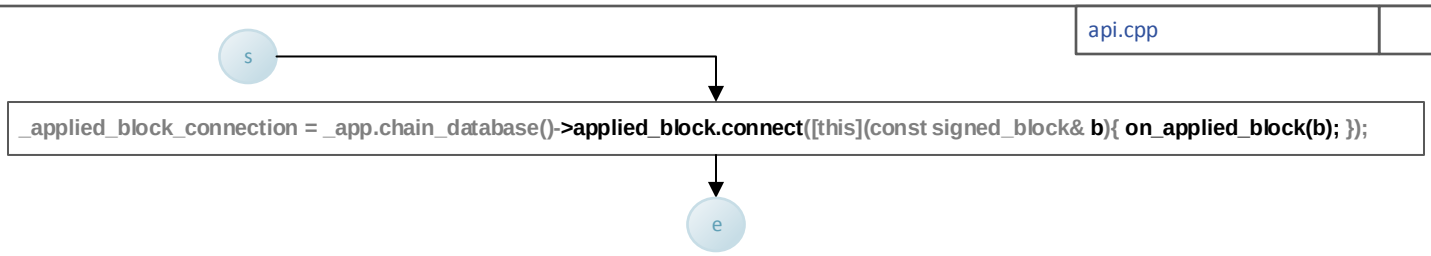
void login_api::**enable_api(** const std::string& **api_name**)

api.cpp



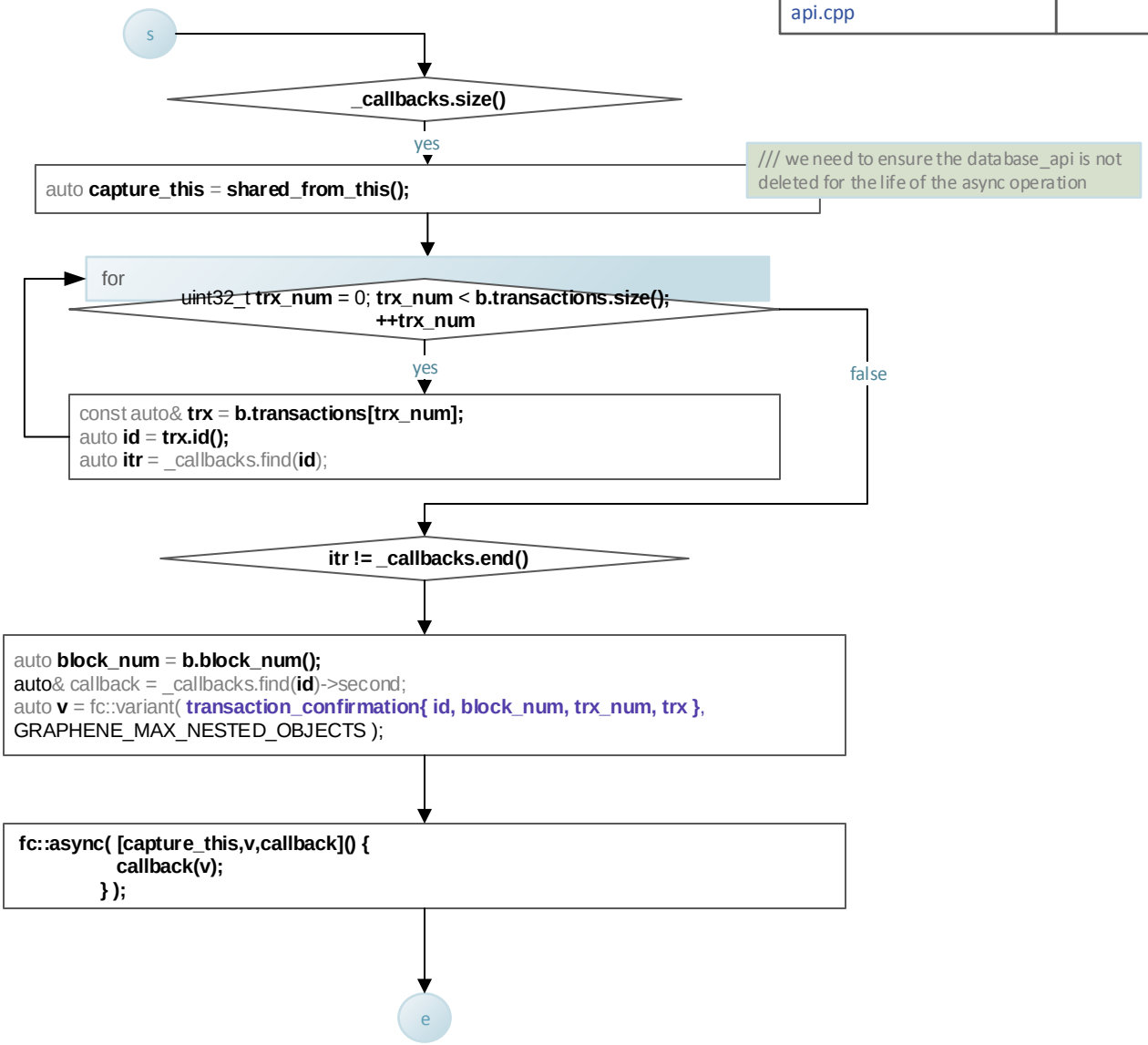
network_broadcast_api::network_broadcast_api(application& a):_app(a) (1/1)

api.cpp



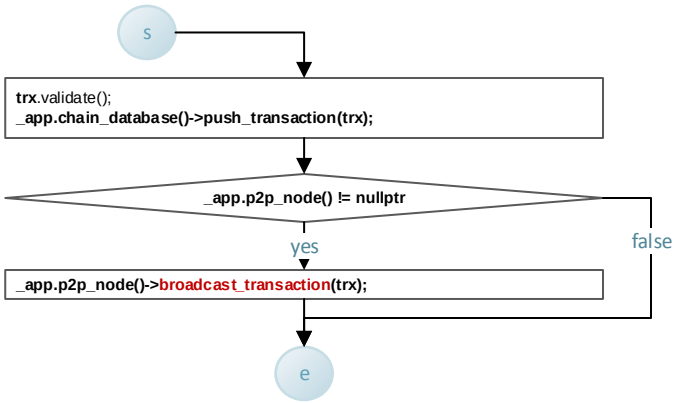
void network_broadcast_api::**on_applied_block**(const **signed_block& b**) (1/1)

api.cpp



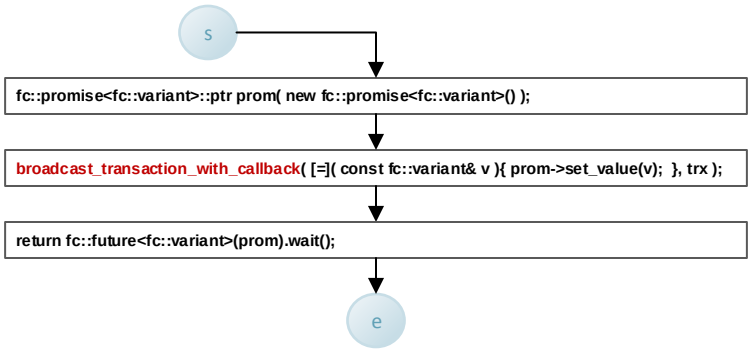
void network_broadcast_api::**broadcast_transaction**(const **signed_transaction& trx**) (1/1)

api.cpp



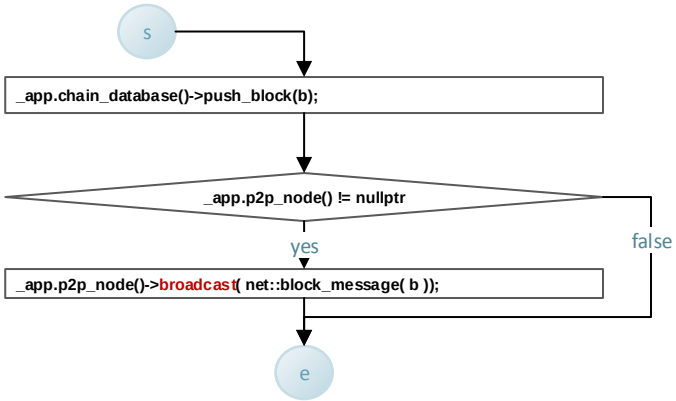
fc::variant network_broadcast_api::**broadcast_transaction_synchronous**(**c**onst **signed_transaction& trx**) (1/1)

api.cpp



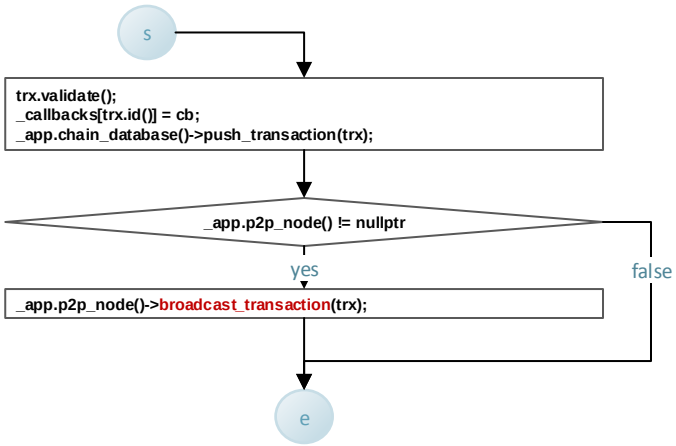
void network_broadcast_api::**broadcast_block**(const **signed_block& b**) (1/1)

api.cpp



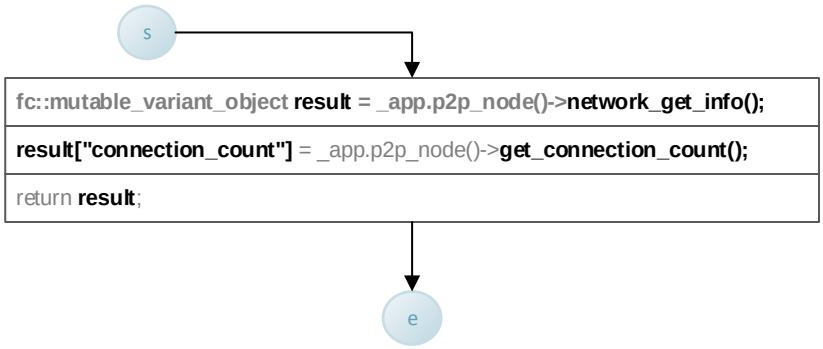
void network_broadcast_api::**broadcast_transaction_with_callback**(confirmation_callback **cb**, const **signed_transaction& trx**)

api.cpp



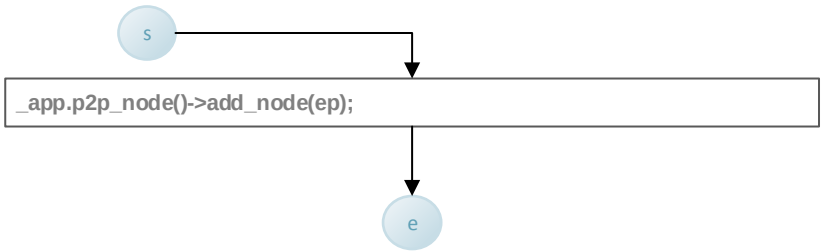
fc::variant_object network_node_api::**get_info()** const (1/1)

api.cpp



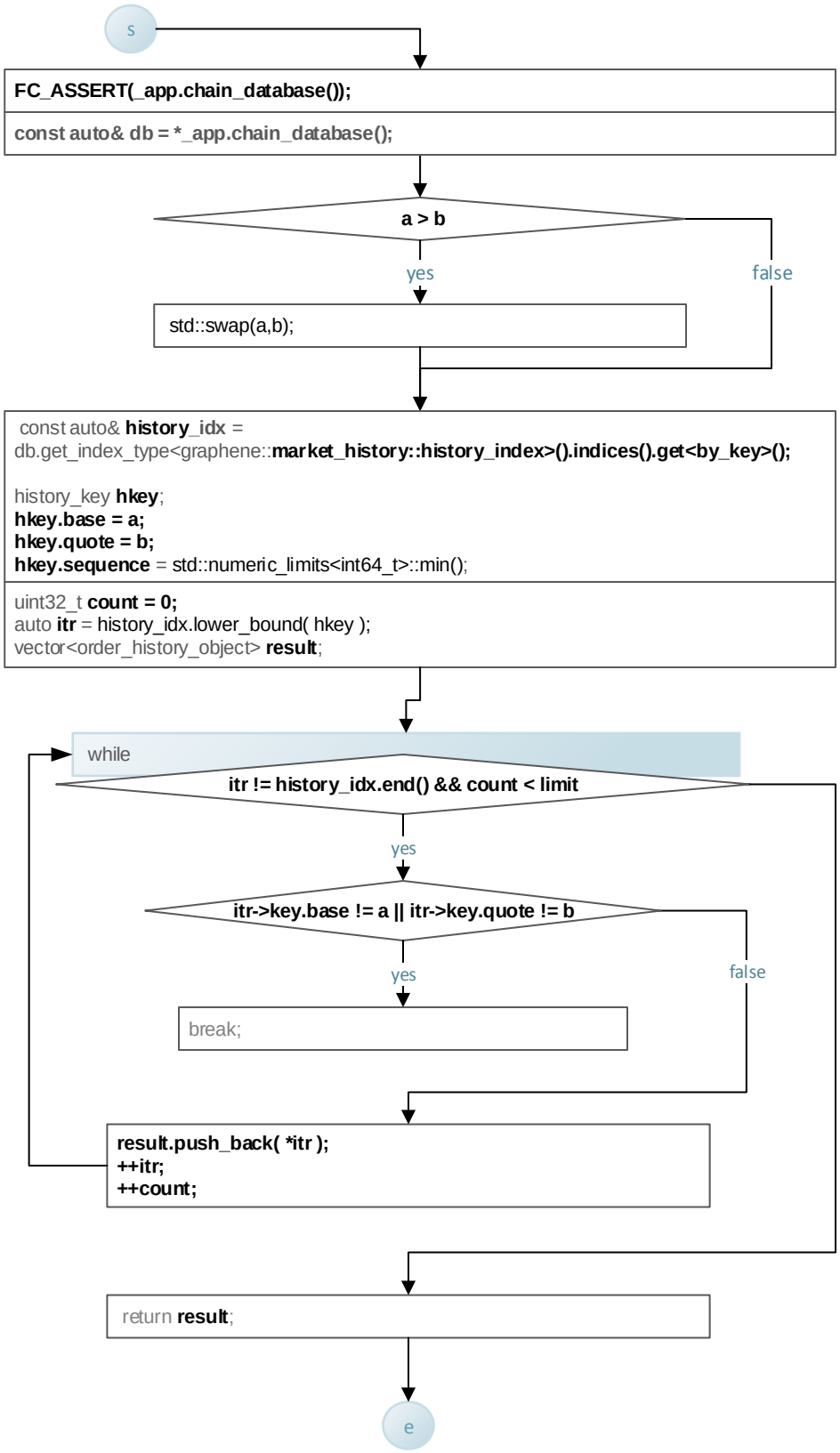
void network_node_api::**add_node**(const fc::ip::endpoint& **ep**) (1/1)

api.cpp



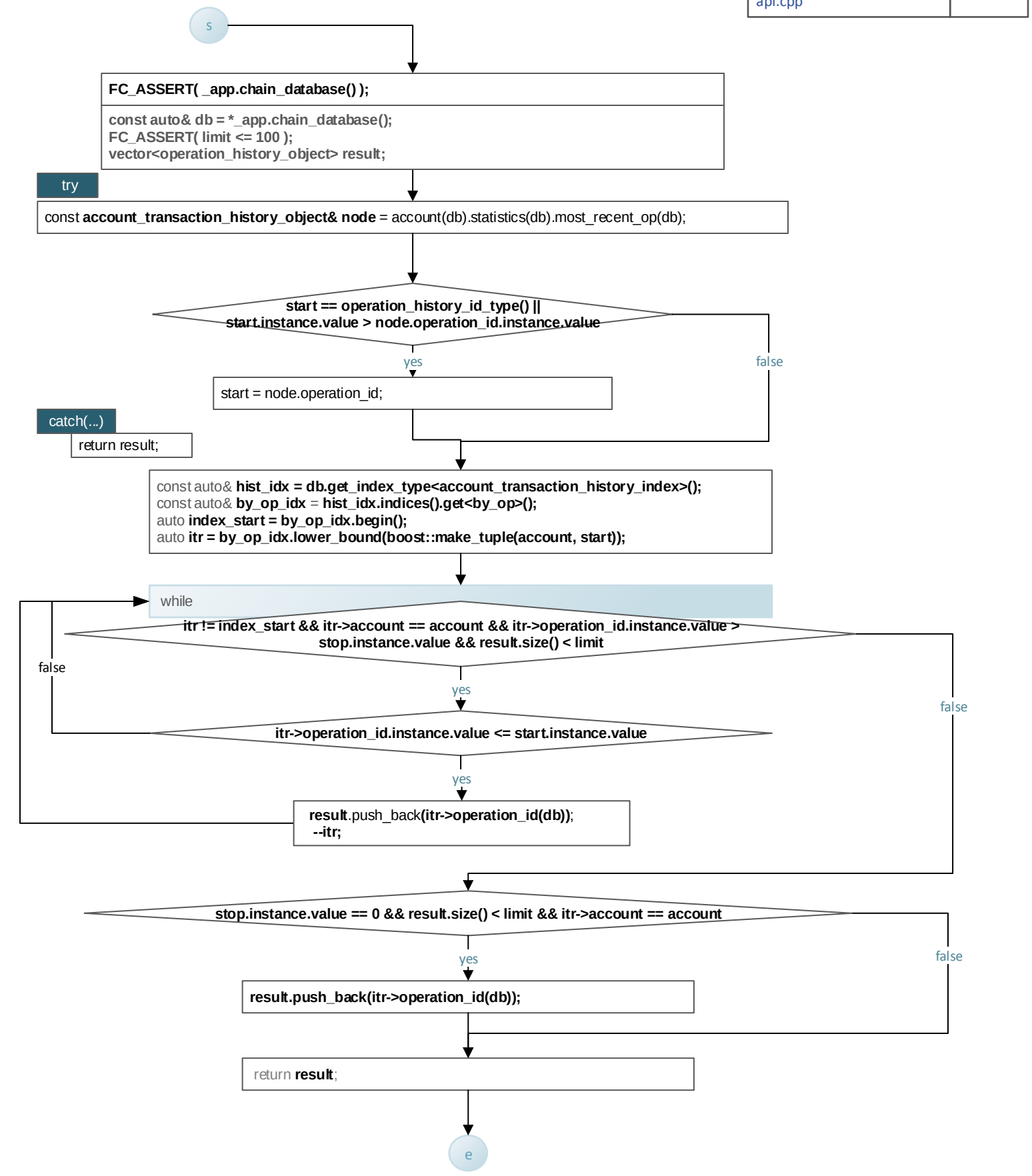
vector<order_history_object> history_api::**get_fill_order_history**(asset_id_type **a,** asset_id_type **b**, uint32_t **limit**)const

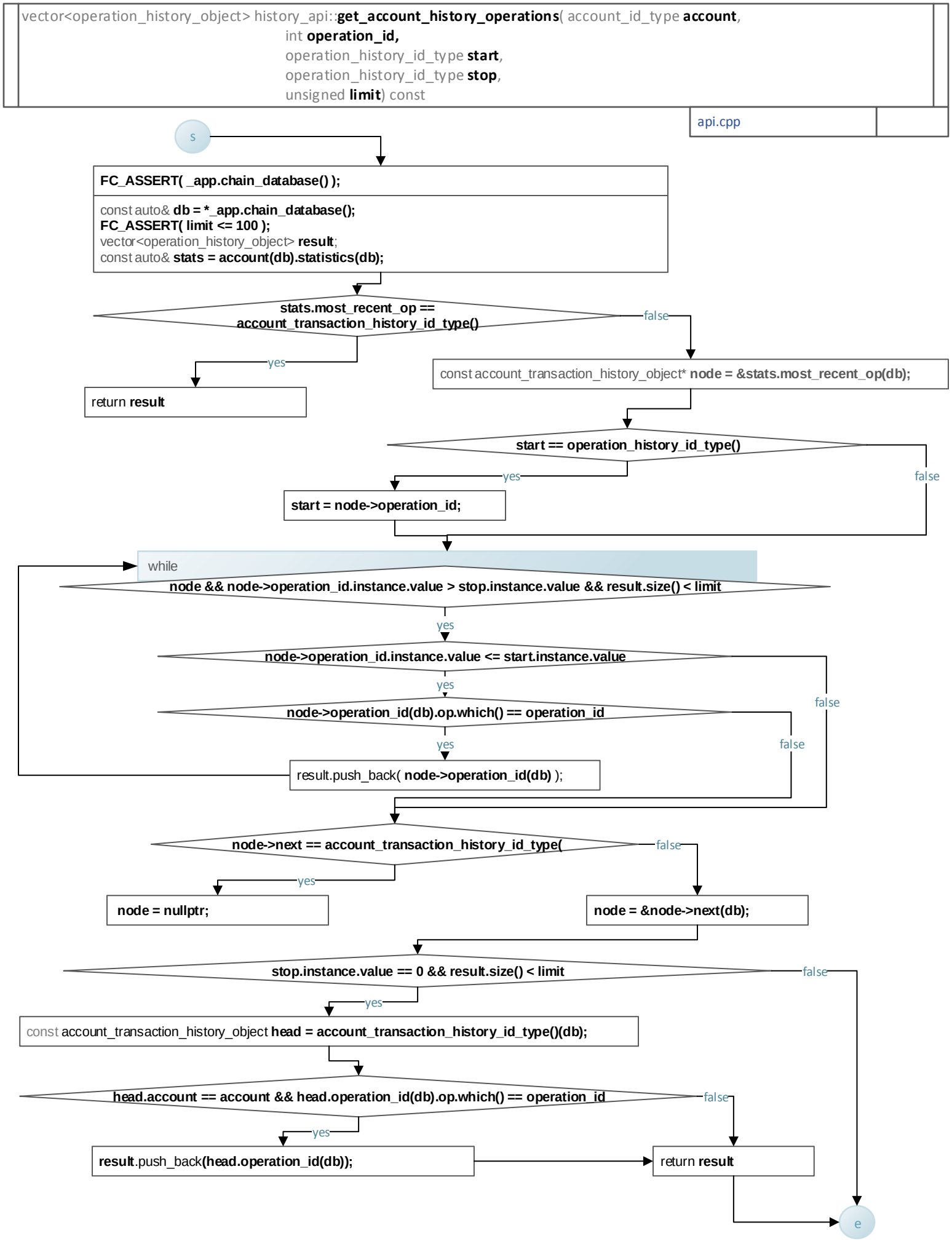
api.cpp



```
vector<operation_history_object> history_api::get_account_history( account_id_type account,
                                                                   operation_history_id_type stop,
                                                                   unsigned limit,
                                                                   operation_history_id_type start ) const
```

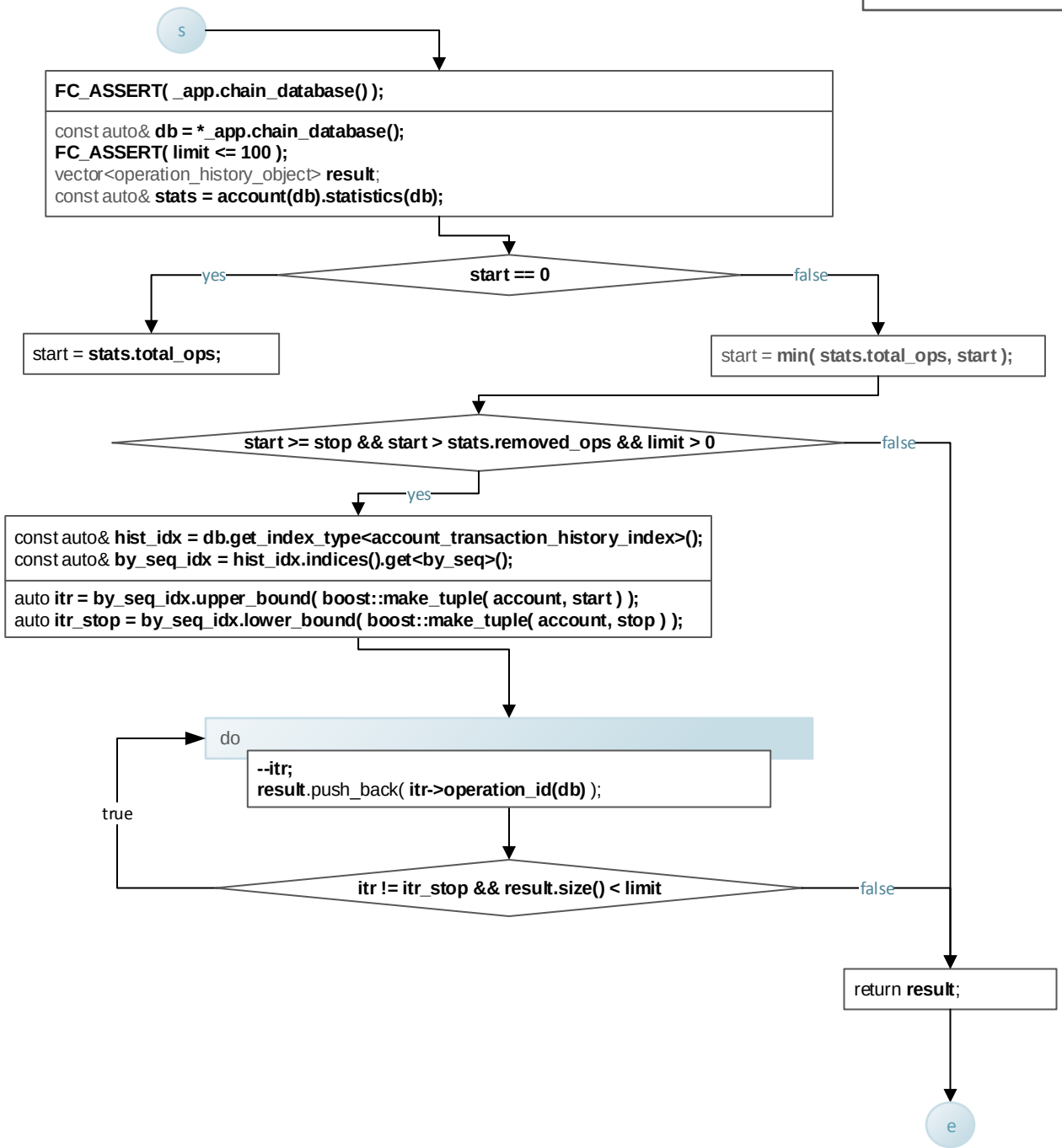
api.cpp

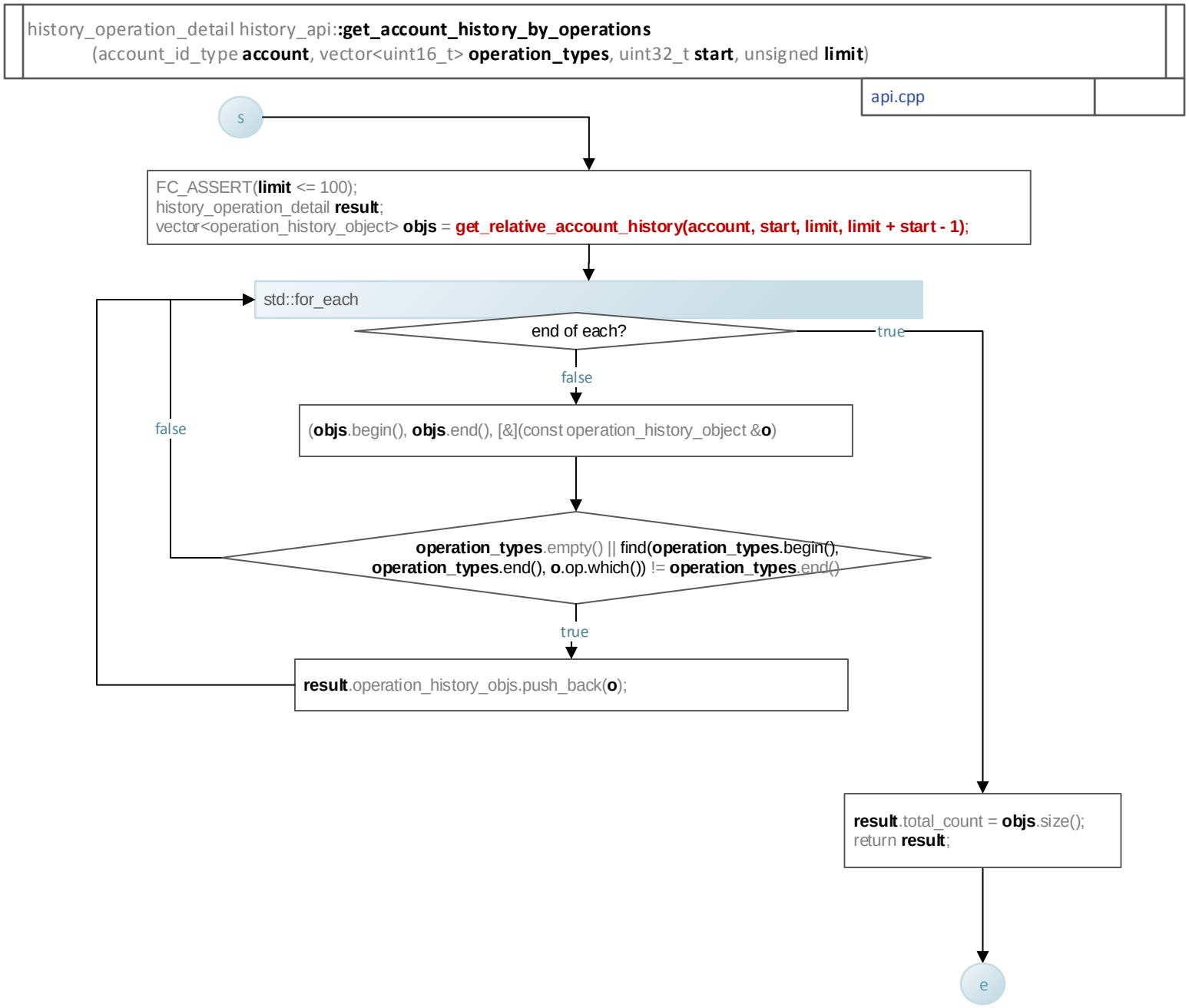



vector<operation_history_object> history_api::get_account_history_operations(account_id_type account,
int operation_id,
operation_history_id_type start,
operation_history_id_type stop,
unsigned limit) const
api.cpp
s
FC_ASSERT(_app.chain_database());
const auto& db = *_app.chain_database();
FC_ASSERT(limit <= 100);
vector<operation_history_object> result;
const auto& stats = account(db).statistics(db);
stats.most_recent_op == account_transaction_history_id_type()
false
yes
return result
const account_transaction_history_object* node = &stats.most_recent_op(db);
start == operation_history_id_type()
yes
false
start = node->operation_id;
while
node && node->operation_id.instance.value > stop.instance.value && result.size() < limit
yes
node->operation_id.instance.value <= start.instance.value
yes
false
node->operation_id(db).op.which() == operation_id
yes
false
result.push_back(node->operation_id(db));
node->next == account_transaction_history_id_type(
false
yes
node = nullptr;
node = &node->next(db);
stop.instance.value == 0 && result.size() < limit
false
yes
const account_transaction_history_object head = account_transaction_history_id_type()(db);
head.account == account && head.operation_id(db).op.which() == operation_id
false
yes
result.push_back(head.operation_id(db));
return result
e

```
vector<operation_history_object> history_api::get_relative_account_history( account_id_type account,
                                                                            uint32_t stop,
                                                                            unsigned limit,
                                                                            uint32_t start) const
```

api.cpp




history_operation_detail history_api::get_account_history_by_operations
(account_id_type account, vector<uint16_t> operation_types, uint32_t start, unsigned limit)
api.cpp
s
FC_ASSERT(limit <= 100);
history_operation_detail result;
vector<operation_history_object> objs = get_relative_account_history(account, start, limit, limit + start - 1);
std::for_each
end of each?
true
false
(objs.begin(), objs.end(), [&](const operation_history_object &o)
false
operation_types.empty() || find(operation_types.begin(), operation_types.end(), o.op.which()) != operation_types.end()
true
result.operation_history_objs.push_back(o);
result.total_count = objs.size();
return result;
e

vector<bucket_object> history_api::**get_market_history**(asset_id_type **a**, asset_id_type **b**,
uint32_t **bucket_seconds**, fc::time_point_sec **start**, fc::time_point_sec **end**)const

api.cpp