Relatório Trabalho 2

Dupla:

Felipe Nazario Avelino e João Pedro Garcia Guedes

Objetivo:

O objetivo deste trabalho foi garantir a correção do software de um sistema bancário implementado em C++, escrito no arquivo bank.hpp, utilizando como referência um modelo formal em Quint. Para isso, utilizamos um esqueleto de testes fornecido em test.cpp, com o qual aplicamos testes baseados em modelos para comparar o comportamento do código com o comportamento esperado definido pelo modelo. A partir dessas comparações, identificamos divergências e realizamos as correções necessárias tanto no código do sistema (bank.hpp) quanto nos testes (test.cpp), com o objetivo de alinhar a implementação ao modelo e aumentar a confiança no sistema.

Funcionamento do Teste:

Alteramos o artefato test.cpp para imprimir informações importantes sobre o estado do banco, incluindo tanto o esperado (definido pelo modelo) quanto o estado que realmente temos.

Na main() temos o seguinte trecho:

```
BankState expected_bank_state = bank_state_from_json(state["bank_state"]);

cout << "----------------------Expected-----------------" << endl;
print_bank(expected_bank_state);
cout << "----------------------Actual-------------------" << endl;
print_bank(bank_state);
cout << endl;
```

Que utiliza a função a print_bank(), definida abaixo:

```
void print_bank(const BankState& bank_state) {
    cout << "Balances: ";
    for (const auto& balance : bank_state.balances) {
        cout << balance.first << ": " << balance.second << " ";
    }

    cout << "\nInvestments:" << endl;
    for (const auto& investment : bank_state.investments) {
        cout << "Investment ID: " << investment.first;
        cout << "  Owner: " << investment.second.owner;
        cout << "  Amount: " << investment.second.amount;
        cout << endl;
    }

    cout << "\nNext ID: " << bank_state.next_id << endl;
}
```

Imprimimos as principais informações do sistema, como o saldo de cada usuário e os dados dos investimentos, incluindo o proprietário e o valor investido.

Também há um print que vai representar uma action:

```
cout << "deposit(" << depositor << ", " << amount << ")" << endl;
error = deposit(bank_state, depositor, amount);
```

Para comparação de erros, utilizamos os seguintes trechos:

```
string expected_error = string(state["error"]["tag"]).compare("Some") == 0
                        ? state["error"]["value"]
                        : "";
cout << "Expected error: " << expected_error << endl;
cout << "Actual error: " << error << endl;
cout << endl;
```

```
assert(bank_state.balances == expected_bank_state.balances);
assert(error == expected_error);
```

As funções do banco (bank.hpp) retornam uma string indicando erro. Se a operação for bem-sucedida, a string é vazia. O teste compara esse retorno com o erro esperado definido no modelo e o saldo esperado com o saldo atual. Quando ambos são iguais, a verificação com assert é satisfeita. Isso garante que o código está se comportando conforme o modelo, tanto nos casos válidos quanto nos que devem gerar erro.

Abaixo um exemplo de saída.

```
----------------------Expected----------------
Balances: Alice: 143 Bob: 3 Charlie: 0
Investments:
Investment ID: 0  Owner: Bob  Amount: 75

Next ID: 1
----------------------Actual------------------
Balances: Alice: 143 Bob: 3 Charlie: 0
Investments:
Investment ID: 0  Owner: Bob  Amount: 75

Next ID: 1

Expected error:
Actual error:

sell_investment(Bob, 7)
----------------------Expected----------------
Balances: Alice: 143 Bob: 3 Charlie: 0
Investments:
Investment ID: 0  Owner: Bob  Amount: 75

Next ID: 1
----------------------Actual------------------
Balances: Alice: 143 Bob: 3 Charlie: 0
Investments:
Investment ID: 0  Owner: Bob  Amount: 75

Next ID: 1

Expected error: No investment with this id
Actual error: No investment with this id
```

Mudanças Ocasionadas pelos Testes:

A seguir, listamos as alterações que identificamos como necessárias nos arquivos

bank.hpp ou test.cpp para que o sistema funcionasse corretamente. Para cada modificação, apresentamos a saída no terminal que evidenciou o problema, junto com o trecho do código antes e depois da alteração.

Implementar action de BuyInvestiment

```
linuxpc@linuxpc:~/UDESC/MFO/bank$ g++ -I lib test.cpp && ./a.out
Trace #0
initializing
----------------------Expected----------------
Balances:
Alice: 0
Bob: 0
Charlie: 0

Investments:

Next ID: 0
----------------------Actual------------------
Balances:
Alice: 0
Bob: 0
Charlie: 0

Investments:

Next ID: 0

TODO: comparar o erro esperado com o erro obtido
TODO: fazer a conexão para as outras ações. Ação: buy_investment_action
```

Na primeira execução já há a necessidade de implementar a ação de comprar investimentos.

Assim ficou test.cpp:

```cpp
case Action::BuyInvestment: {
  string buyer = nondet_picks["buyer"]["value"];
  int amount = int_from_json(nondet_picks["amount"]["value"]);
  cout << "buy_investment(" << buyer << ", " << amount << ")" << endl;
  buy_investment(bank_state, buyer, amount);
  break;
}
```

Compra de quantidade negativa de investimentos

```
Trace #0
initializing
----------------------Expected-----------------
Balances:
Alice: 0
Bob: 0
Charlie: 0

Investments:

Next ID: 0
----------------------Actual-------------------
Balances:
Alice: 0
Bob: 0
Charlie: 0

Investments:

Next ID: 0

TODO: comparar o erro esperado com o erro obtido
buy_investment(Alice, -36)
----------------------Expected-----------------
Balances:
Alice: 0
Bob: 0
Charlie: 0

Investments:

Next ID: 0
----------------------Actual-------------------
Balances:
Alice: 36
Bob: 0
Charlie: 0

Investments:
Investment ID: 0
  Owner: Alice
  Amount: -36

Next ID: 1

TODO: comparar o erro esperado com o erro obtido
```

No nosso primeiro erro encontrado, a Alice compra uma quantidade negativa de investimentos e o sistema aceita, assim tendo uma diferença na quantidade de investimentos entre o esperado e o atual, logo precisamos ajustar buy_investment em bank.hpp para que isso não seja possível tal qual o esperado.

Antes da alteração:

```
string buy_investment(BankState &bank_state, string buyer, int amount) {
  bank_state.balances[buyer] -= amount;
  bank_state.investments[bank_state.next_id] = {buyer, amount};
  bank_state.next_id++;
  return "";
}
```

Após a alteração:

```
string buy_investment(BankState &bank_state, string buyer, int amount) {
  if(amount <= 0){
    return "Amount should be greater than zero";
  }
  bank_state.balances[buyer] -= amount;
  bank_state.investments[bank_state.next_id] = {buyer, amount};
  bank_state.next_id++;
  return "";
}
```

Implementar action de Withdraw

```
Trace #0
initializing
----------------------Expected-----------------
Balances: Alice: 0 Bob: 0 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 0 Charlie: 0
Investments:
Next ID: 0

TODO: comparar o erro esperado com o erro obtido
buy_investment(Alice, -36)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 0 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 0 Charlie: 0
Investments:
Next ID: 0

TODO: comparar o erro esperado com o erro obtido
TODO: fazer a conexão para as outras ações. Ação: withdraw_action
```

Agora há a necessidade de implementar a ação de saque.

Assim ficou test.cpp:

```
case Action::Withdraw: {
  string withdrawer = nondet_picks["withdrawer"]["value"];
  int amount = int_from_json(nondet_picks["amount"]["value"]);
  cout << "withdraw(" << withdrawer << ", " << amount << ")" << endl;
  withdraw(bank_state, withdrawer, amount);
  break;
}
```

Saque de dinheiro sem saldo

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 0 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 0 Charlie: 0
Investments:
Next ID: 0

TODO: comparar o erro esperado com o erro obtido
withdraw(Alice, 96)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 0 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: -96 Bob: 0 Charlie: 0
Investments:
Next ID: 0

TODO: comparar o erro esperado com o erro obtido
```

Aqui Alice conseguiu sacar dinheiro sem saldo, assim tendo uma diferença na quantidade de saldo entre o esperado e o atual, logo precisamos mudar withdraw em bank.hpp para que o sistema se comporte como esperado.

Antes da alteração:

```
string withdraw(BankState &bank_state, string withdrawer, int amount) {
  bank_state.balances[withdrawer] -= amount;
  return "";
}
```

Depois da alteração:

```
string withdraw(BankState &bank_state, string withdrawer, int amount) {
  if(bank_state.balances[withdrawer] < amount){
    return "Balance is too low";
  }
  bank_state.balances[withdrawer] -= amount;
  return "";
}
```

Necessidade de implementar action de Deposit

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 0 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 0 Charlie: 0
Investments:
Next ID: 0

TODO: comparar o erro esperado com o erro obtido
TODO: chamar a função correspondente
----------------------Expected-----------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 0 Charlie: 0
Investments:
Next ID: 0

TODO: comparar o erro esperado com o erro obtido
```

Agora há a necessidade de implementar a ação de depósito.

Assim ficou test.cpp:

```cpp
case Action::Deposit: {
  string depositor = nondet_picks["depositor"]["value"];
  int amount = int_from_json(nondet_picks["amount"]["value"]);
  cout << "deposit(" << depositor << ", " << amount << ")" << endl;
  deposit(bank_state, depositor, amount);
  break;
}
```

Depósito de quantidade negativa

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0

TODO: comparar o erro esperado com o erro obtido
deposit(Alice, -20)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: -20 Bob: 71 Charlie: 0
Investments:
Next ID: 0

TODO: comparar o erro esperado com o erro obtido
```

Alice conseguiu depositar uma quantidade negativa, assim tendo uma diferença na quantidade de saldo entre o esperado e o atual, logo precisamos ajustar deposit em bank.hpp para que o sistema se comporte como esperado.

Antes da alteração:

```
string deposit(BankState &bank_state, string depositor, int amount) {
  bank_state.balances[depositor] += amount;
  return "";
}
```

Após a alteração:

```
string deposit(BankState &bank_state, string depositor, int amount) {
  if(amount <= 0){
    return "Amount should be greater than zero";
  }
  bank_state.balances[depositor] += amount;
  return "";
}
```

Necessidade de implementar action de Transfer

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0

TODO: comparar o erro esperado com o erro obtido
deposit(Alice, -20)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0

TODO: comparar o erro esperado com o erro obtido
TODO: fazer a conexão para as outras ações. Ação: transfer_action
```

Agora há a necessidade de implementar a ação de transferência.

Assim ficou test.cpp:

```
case Action::Transfer: {
  string sender = nondet_picks["sender"]["value"];
  string receiver = nondet_picks["receiver"]["value"];
  int amount = int_from_json(nondet_picks["amount"]["value"]);
  cout << "transfer(" << sender << ", " << receiver << " , " << amount << ")" << endl;
  transfer(bank_state,sender,receiver,amount);
  break;
}
```

Transferência sem saldo

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0

transfer(Charlie, Alice , 45)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 71 Charlie: 0
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 45 Bob: 71 Charlie: -45
Investments:
Next ID: 0
```

Charlie conseguiu conseguiu fazer uma transferência mesmo sem saldo para Alice, assim tendo uma diferença na quantidade de saldo entre o esperado e o atual, logo precisamos modificar transfer em bank.hpp para que o sistema se comporte como esperado.

Antes da modificação:

```
string transfer(BankState &bank_state, string sender, string receiver,
                int amount) {
  bank_state.balances[sender] -= amount;
  bank_state.balances[receiver] += amount;
  return "";
}
```

Depois da modificação:

```
string transfer(BankState &bank_state, string sender, string receiver,
                int amount) {
  if(bank_state.balances[sender] < amount){
    return "Balance is too low";
  }
  bank_state.balances[sender] -= amount;
  bank_state.balances[receiver] += amount;
  return "";
}
```

Necessidade de implementar action de SellInvestment

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 43 Charlie: 28
Investments:
Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 43 Charlie: 28
Investments:
Next ID: 0

buy_investment(Bob, 32)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 11 Charlie: 28
Investments:Investment ID: 0  Owner: Bob  Amount: 32
Next ID: 1
----------------------Actual-------------------
Balances: Alice: 0 Bob: 11 Charlie: 28
Investments:Investment ID: 0  Owner: Bob  Amount: 32
Next ID: 1

TODO: fazer a conexão para as outras ações. Ação: sell_investment_action
```

Agora há a necessidade de implementar a ação de venda de investimentos.

Assim ficou test.cpp:

```
case Action::SellInvestment:{
    string seller = nondet_picks["seller"]["value"];
    int id = int_from_json(nondet_picks["id"]["value"]);
    cout << "sell_investment(" << seller << ", " << id << ")" << endl;
    sell_investment(bank_state,seller,id);
    break;
  }
```

Venda de investimento inexistente

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 11 Charlie: 28
Investments:
Investment ID: 0  Owner: Bob  Amount: 32

Next ID: 1
----------------------Actual-------------------
Balances: Alice: 0 Bob: 11 Charlie: 28
Investments:
Investment ID: 0  Owner: Bob  Amount: 32

Next ID: 1

sell_investment(Charlie, 4)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 11 Charlie: 28
Investments:
Investment ID: 0  Owner: Bob  Amount: 32

Next ID: 1
----------------------Actual-------------------
Balances: Alice: 0 Bob: 11 Charlie: 28
Investments:
Investment ID: 0  Owner: Bob  Amount: 32
Investment ID: 4  Owner:   Amount: 0

Next ID: 1

deposit(Alice, -45)
```

Nessa situação Charlie vendeu um investimento que não existia, assim tendo uma diferença na quantidade de investimentos entre o esperado e o atual, logo precisamos modificar sell_investment em bank.hpp para que o sistema se comporte como esperado.

Antes da modificação:

```
string sell_investment(BankState &bank_state, string seller,
                       int investment_id) {
  bank_state.balances[seller] += bank_state.investments[investment_id].amount;
  return "";
}
```

Depois da modificação:

```
string sell_investment(BankState &bank_state, string seller,
                       int investment_id) {
  if(bank_state.investments.find(investment_id) == bank_state.investments.end()){
    return "No investment with this id";
  }
  bank_state.balances[seller] += bank_state.investments[investment_id].amount;
  return "";
}
```

Saque de quantidade negativa

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 11 Charlie: 28
Investments:
Investment ID: 0  Owner: Bob  Amount: 32

Next ID: 1
----------------------Actual-------------------
Balances: Alice: 0 Bob: 11 Charlie: 28
Investments:
Investment ID: 0  Owner: Bob  Amount: 32

Next ID: 1

withdraw(Bob, -16)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 11 Charlie: 28
Investments:
Investment ID: 0  Owner: Bob  Amount: 32

Next ID: 1
----------------------Actual-------------------
Balances: Alice: 0 Bob: 27 Charlie: 28
Investments:
Investment ID: 0  Owner: Bob  Amount: 32

Next ID: 1
```

Aqui Bob conseguiu sacar uma quantidade negativa, assim tendo uma diferença na quantidade de saldo entre o esperado e o atual, logo precisamos modificar withdraw em bank.hpp para que o sistema se comporte como esperado.

Antes da modificação:

```
string withdraw(BankState &bank_state, string withdrawer, int amount) {
  if(bank_state.balances[withdrawer] < amount){
    return "Balance is too low";
  }
  bank_state.balances[withdrawer] -= amount;
  return "";
}
```

Depois da modificação:

```
string withdraw(BankState &bank_state, string withdrawer, int amount) {
  if(bank_state.balances[withdrawer] < amount){
    return "Balance is too low";
  }
  if(amount <= 0){
    return "Amount should be greater than zero";
  }
  bank_state.balances[withdrawer] -= amount;
  return "";
}
```

Compra de investimento maior que o saldo

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 0 Charlie: 67
Investments:

Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 0 Charlie: 67
Investments:

Next ID: 0

buy_investment(Charlie, 82)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 0 Charlie: 67
Investments:

Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 0 Charlie: -15
Investments:
Investment ID: 0  Owner: Charlie  Amount: 82

Next ID: 1
```

Aqui Charlie conseguiu comprar mais investimentos do que seu saldo permitia, assim tendo uma diferença na quantidade de saldo e investimentos entre o esperado e o atual, logo precisamos modificar buy_investment em bank.hpp para que o sistema se comporte como esperado.

Antes da modificação:

```
string buy_investment(BankState &bank_state, string buyer, int amount) {
  if(amount <= 0){
    return "Amount should be greater than zero";
  }
  bank_state.balances[buyer] -= amount;
  bank_state.investments[bank_state.next_id] = {buyer, amount};
  bank_state.next_id++;
  return "";
}
```

Depois da modificação:

```
string buy_investment(BankState &bank_state, string buyer, int amount) {
  if(amount <= 0){
    return "Amount should be greater than zero";
  }
  if(bank_state.balances[buyer] - amount < 0){
    return "Balance is too low";
  }
  bank_state.balances[buyer] -= amount;
  bank_state.investments[bank_state.next_id] = {buyer, amount};
  bank_state.next_id++;
  return "";
}
```

Transferência de quantidade negativa

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 0 Charlie: 67
Investments:

Next ID: 0
----------------------Actual-------------------
Balances: Alice: 0 Bob: 0 Charlie: 67
Investments:

Next ID: 0

transfer(Alice, Bob , -50)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 0 Charlie: 67
Investments:

Next ID: 0
----------------------Actual-------------------
Balances: Alice: 50 Bob: -50 Charlie: 67
Investments:

Next ID: 0
```

Nessa situação Alice fez uma transferência negativa para Bob, assim tendo uma diferença na quantidade de saldo entre o esperado e o atual, logo precisamos modificar transfer em bank.hpp para que o sistema se comporte como esperado.

Antes da modificação:

```
string transfer(BankState &bank_state, string sender, string receiver,
                int amount) {
  if(bank_state.balances[sender] < amount){
    return "Balance is too low";
  }
  bank_state.balances[sender] -= amount;
  bank_state.balances[receiver] += amount;
  return "";
}
```

Depois da modificação:

```
string transfer(BankState &bank_state, string sender, string receiver,
                int amount) {
  if(bank_state.balances[sender] < amount){
    return "Balance is too low";
  }
  if(amount <= 0){
    return "Amount should be greater than zero";
  }
  bank_state.balances[sender] -= amount;
  bank_state.balances[receiver] += amount;
  return "";
}
```

Venda de investimento por usuário incorreto

```
----------------------Expected-----------------
Balances: Alice: 4 Bob: 0 Charlie: 0
Investments:
Investment ID: 0  Owner: Alice  Amount: 43
Investment ID: 1  Owner: Alice  Amount: 25

Next ID: 2
----------------------Actual-------------------
Balances: Alice: 4 Bob: 0 Charlie: 0
Investments:
Investment ID: 0  Owner: Alice  Amount: 43
Investment ID: 1  Owner: Alice  Amount: 25

Next ID: 2

sell_investment(Bob, 1)
----------------------Expected-----------------
Balances: Alice: 4 Bob: 0 Charlie: 0
Investments:
Investment ID: 0  Owner: Alice  Amount: 43
Investment ID: 1  Owner: Alice  Amount: 25

Next ID: 2
----------------------Actual-------------------
Balances: Alice: 4 Bob: 25 Charlie: 0
Investments:
Investment ID: 0  Owner: Alice  Amount: 43
Investment ID: 1  Owner: Alice  Amount: 25

Next ID: 2
```

Nessa situação Bob vendeu um investimento que não era dele, assim tendo uma diferença na quantidade de saldo entre o esperado e o atual, logo precisamos modificar sell_investment em

bank.hpp para que o sistema se comporte como esperado.

Antes da modificação:

```
string sell_investment(BankState &bank_state, string seller,
                       int investment_id) {
  if(bank_state.investments.find(investment_id) == bank_state.investments.end()){
    return "No investment with this id";
  }
  bank_state.balances[seller] += bank_state.investments[investment_id].amount;
  return "";
}
```

Depois da modificação:

```
string sell_investment(BankState &bank_state, string seller,
                       int investment_id) {
  if(bank_state.investments.find(investment_id) == bank_state.investments.end()){
    return "No investment with this id";
  }
  if(bank_state.investments[investment_id].owner != seller){
    return "Seller can't sell an investment they don't own";
  }
  bank_state.balances[seller] += bank_state.investments[investment_id].amount;
  return "";
}
```

Atualização de id do investimento

```
----------------------Expected-----------------
Balances: Alice: 0 Bob: 48 Charlie: 0
Investments:
Investment ID: 1  Owner: Bob  Amount: 14

Next ID: 2
----------------------Actual-------------------
Balances: Alice: 0 Bob: 48 Charlie: 0
Investments:
Investment ID: 0  Owner: Bob  Amount: 57
Investment ID: 1  Owner: Bob  Amount: 14

Next ID: 2

sell_investment(Bob, 0)
----------------------Expected-----------------
Balances: Alice: 0 Bob: 48 Charlie: 0
Investments:
Investment ID: 1  Owner: Bob  Amount: 14

Next ID: 2
----------------------Actual-------------------
Balances: Alice: 0 Bob: 105 Charlie: 0
Investments:
Investment ID: 0  Owner: Bob  Amount: 57
Investment ID: 1  Owner: Bob  Amount: 14

Next ID: 2
```

Nessa situação o sistema não está apagando o id do investimento ao realizar a venda, assim tendo uma diferença na quantidade de saldo e investimento entre o esperado e o atual. Com isso, precisamos modificar sell_investment em bank.hpp para que o sistema se comporte como esperado.

Antes da modificação:

```
string sell_investment(BankState &bank_state, string seller,
                       int investment_id) {
  if(bank_state.investments.find(investment_id) == bank_state.investments.end()){
    return "No investment with this id";
  }
  if(bank_state.investments[investment_id].owner != seller){
    return "Seller can't sell an investment they don't own";
  }
  bank_state.balances[seller] += bank_state.investments[investment_id].amount;
  return "";
}
```

Depois da modificação:

```
string sell_investment(BankState &bank_state, string seller,
                       int investment_id) {
  if(bank_state.investments.find(investment_id) == bank_state.investments.end()){
    return "No investment with this id";
  }
  if(bank_state.investments[investment_id].owner != seller){
    return "Seller can't sell an investment they don't own";
  }
  bank_state.balances[seller] += bank_state.investments[investment_id].amount;
  bank_state.investments.erase(investment_id);
  return "";
}
```