*/\* eslint-disable max-len \*/*

**const** { expect } = require('chai')

**const** { describe, it } = require('mocha')

**const** reconcileOrder = require('./orderBook')

describe('Order Book', () **=>** {

describe('reconcileOrder', () **=>** {

it('adds an order to the book when the book is empty and thus cannot fulfill the order', () **=>** {

**const** existingBook = []

**const** incomingOrder = { type: 'sell', quantity: 10, price: 6150 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([{ type: 'sell', quantity: 10, price: 6150 }])

})

it('adds an order to the book when the book has orders of the corresponding type (i.e. a sell with no buys)', () **=>** {

**const** existingBook = [{ type: 'sell', quantity: 10, price: 6150 }]

**const** incomingOrder = { type: 'sell', quantity: 12, price: 6000 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([

{ type: 'sell', quantity: 10, price: 6150 },

{ type: 'sell', quantity: 12, price: 6000 }

])

})

it('adds an order to the book when the book has a corresponding order type but it does not match', () **=>** {

**const** existingBook = [{ type: 'buy', quantity: 10, price: 6000 }]

**const** incomingOrder = { type: 'sell', quantity: 12, price: 6150 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([

{ type: 'buy', quantity: 10, price: 6000 },

{ type: 'sell', quantity: 12, price: 6150 }

])

})

it('fulfills an order and removes the matching order when the book contains a matching order of the same quantity', () **=>** {

**const** existingBook = [{ type: 'buy', quantity: 10, price: 6150 }, { type: 'sell', quantity: 12, price: 6250 }]

**const** incomingOrder = { type: 'sell', quantity: 10, price: 6150 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([{ type: 'sell', quantity: 12, price: 6250 }])

})

it('fulfills an order and reduces the matching order when the book contains a matching order of a larger quantity', () **=>** {

**const** existingBook = [{ type: 'buy', quantity: 15, price: 6150 }, { type: 'sell', quantity: 12, price: 6950 }]

**const** incomingOrder = { type: 'sell', quantity: 10, price: 6150 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([{ type: 'sell', quantity: 12, price: 6950 }, { type: 'buy', quantity: 5, price: 6150 }])

})

it('partially fulfills an order, removes the matching order and adds the remainder of the order to the book when the book contains a matching order of a smaller quantity', () **=>** {

**const** existingBook = [{ type: 'buy', quantity: 10, price: 6150 }, { type: 'sell', quantity: 12, price: 5950 }]

**const** incomingOrder = { type: 'sell', quantity: 15, price: 6150 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([{ type: 'sell', quantity: 12, price: 5950 }, { type: 'sell', quantity: 5, price: 6150 }])

})

it('uses two existing orders to completely fulfill an order, removing the matching orders from the book', () **=>** {

**const** existingBook = [{ type: 'buy', quantity: 10, price: 6150 }, { type: 'buy', quantity: 5, price: 6150 }, { type: 'sell', quantity: 12, price: 5950 }]

**const** incomingOrder = { type: 'sell', quantity: 15, price: 6150 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([{ type: 'sell', quantity: 12, price: 5950 }])

})

it('uses two existing orders to completely fulfill an order, removing the first matching order from the book and reducing the second', () **=>** {

**const** existingBook = [{ type: 'buy', quantity: 10, price: 6150 }, { type: 'buy', quantity: 10, price: 6150 }, { type: 'sell', quantity: 12, price: 6950 }]

**const** incomingOrder = { type: 'sell', quantity: 15, price: 6150 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([{ type: 'sell', quantity: 12, price: 6950 }, { type: 'buy', quantity: 5, price: 6150 }])

})

it('uses two existing orders to partially fulfill an order, removing the matching orders from the book and reducing the incoming order before adding it to the book', () **=>** {

**const** existingBook = [{ type: 'buy', quantity: 10, price: 6150 }, { type: 'buy', quantity: 10, price: 6150 }, { type: 'sell', quantity: 12, price: 6950 }]

**const** incomingOrder = { type: 'sell', quantity: 25, price: 6150 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([{ type: 'sell', quantity: 12, price: 6950 }, { type: 'sell', quantity: 5, price: 6150 }])

})

it.skip('Extra Credit: it fulfills a mismatched order when both parties benefit', () **=>** {

**const** existingBook = [{ type: 'buy', quantity: 15, price: 6000 }, { type: 'sell', quantity: 12, price: 6950 }]

**const** incomingOrder = { type: 'sell', quantity: 15, price: 5900 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([{ type: 'sell', quantity: 12, price: 6950 }])

})

it.skip('Extra Credit: it does not fulfill a mismatched order when it does not benefit both parties', () **=>** {

**const** existingBook = [{ type: 'buy', quantity: 15, price: 5900 }, { type: 'sell', quantity: 12, price: 6950 }]

**const** incomingOrder = { type: 'sell', quantity: 15, price: 6000 }

**const** updatedBook = reconcileOrder(existingBook, incomingOrder)

expect(updatedBook).to.deep.equal([

{ type: 'buy', quantity: 15, price: 5900 },

{ type: 'sell', quantity: 12, price: 6950 },

{ type: 'sell', quantity: 15, price: 6000 },

])

})

})

})