FlexTrade Stock Trade Data Organizer C++ Program Test Plan

Hogyun Lim

May 30th, 2025

Current Version: Stock Trade Data Organizer v1.2

1. Introduction

1.1 Purpose

This document explains the test plan for the Stock_Trade_Data_Organizer v1.2 version of the Stock Trade Data Organizer command-line C++ program. This program is designed to read stock trade data from a file, compute the volume percentages needed to calculate the Volume Weighted Average Prices (VWAP), and determine daily maximum high as well as minimum low prices for all trade entries in the input file. The purpose of this test plan is to ensure that the Stock Trade Data Organizer functions as specified, is safe, and handles various scenarios as expected.

1.2 Program Specification

The following is the program specification included in the original Stock_Trade_Data_Organizer v1.1 C++ Program:

```
/**
Stock_Trade_Data_Organizer v1.1

// Read file in with the following format:
// [Stock],[Interval],[Volume Traded],[High],[Low]

// Calculate the total volume traded per Stock
// Calculate the total volume traded per Stock&Interval

// Write the total volume traded per Stock&Interval as a percentage of the total volume traded per Stock to stdout:
// [Stock],[Interval],[%Volume Traded]

// Write the delimiter '#' to stdout

// Write the maximum High and minimum Low for each Stock to stdout:
// [Stock],[Day High],[Day Low]

// example input:
VOD.L 1 100 184 183.7
BT.LN 1 300 449.4 448.2
VOD.L 2 25 184.1 182.4
BT.LN 2 900 449.8 449.5

// example output:
VOD.L,1,80
BT.LN,1,25
VOD.L,2,20
BT.LN,2,75
#
VOD.L,184.1,182.4
BT.LN,449.8,448.2

**/
```

1.3 Program Overview: Stock_Trade_Data_Organizer v1.2 reads data from an input file. Each line should contain Stock ID, Interval, Volume Traded, High Price, and Low Price. The command-line argument should be either version or filename. With the version argument the program should output the version name and terminate, and with the filename argument the program should output to the console according to the program specifications. In particular, the program should calculate the percentage of volume for each stock/interval against the total volume for that stock. It also identifies the maximum high and minimum low price for each stock across all intervals.

2. Scope of Testing

2.1 In Scope

The following are behaviors to be tested that are based on the intended use of the original Stock_Trade_Data_Organizer v1.1 version of the program as well as the program specifications:

1. Command-line argument processing (version command, filename input, argument count errors).
2. Input file reading and processing.
3. Correctness of calculations:
    a. Total volume per stock.
    b. Percentage of volume traded per stock in interval.
    c. Maximum high and minimum low prices per stock.
4. Accuracy of output formatting as per specifications.
5. Error handling for file access issues.
6. Behavior against invalid data.

2.2 Out of Scope

The following are behaviors that are deemed not worthwhile to be tested as they are are outside of the scope of the original Stock_Trade_Data_Organizer v1.1 version of the program and the program specifications:

1. Performance testing under extreme load (e.g., millions of entries).
2. Testing on different combinations of operating systems and compilers.
3. Memory leak detection.

3. Testing Approach & Strategy

3.1 Testing Method

This program should be tested by applying correct inputs to check for intended behavior as well as incorrect/invalid inputs to check for error handling. The correct inputs for intended behavior should include correct command-line input style, version checking, and checking correct filename and file content structure. The incorrect/invalid inputs should be incorrect command-line arguments, an empty input file, incorrect data lines in the file, a multiple-line file with no correct lines, and a file with negative values for prices. The test should ideally be done by a tester who has not seen the implementation of the program to reduce bias.

3.2 Test Environment

The test environment should be a command-line environment on a computer with a standard C++ compiler.

3.3 Pass/Fail Criteria

A test pass would be when all console outputs (stdout, stderr) exactly match the expected outputs for the given test case, and the program exits with the expected code (0 for success, -1 for error). A test fail would be when there is any deviation from the expected outputs or exit behavior.

4. Test Cases

4.1 Inputs to Check for Correct Behavior:

1. Standard Valid File:

   ```
   VOD.L 1 100 184 183.7
   BT.LN 1 300 449.4 448.2
   VOD.L 2 25 184.1 182.4
   BT.LN 2 900 449.8 449.5
   ```

   Purpose: Checks basic processing of multiple stocks, intervals, and high/low prices.

2. Single Stock, Multiple Intervals:

   ```
   AAPL 1 500 150.50 150.00
   AAPL 2 1000 151.00 150.60
   AAPL 3 200 150.80 150.20
   ```

   Purpose: Ensures correct outputs and calculations for a single stock.

3. File with Only One Data Line:

   ```
   MSFT 1 1000 200.00 199.50
   ```

   Purpose: Checks handling of a valid input on the smallest scale.

4. File with Zero Volume for an Interval:

   ```
   GOOG 1 100 300.0 299.0
   GOOG 2 0 300.5 300.1
   GOOG 3 50 301.0 298.0
   ```

   Purpose: Verifies correct percentage calculation (0%) for zero volume trade intervals.

5. File where a Stock's Total Volume is Zero:

   ```
   ZEROSTOCK 1 0 10.0 9.5
   ZEROSTOCK 2 0 10.1 9.6
   OTHERSTOCK 1 100 20.0 19.0
   ```

   Purpose: Checks how the percentage is calculated and outputted when Total Volume is zero.

4.2 Inputs to Check for Error Handling / Crashing

1. Command-Line Argument Errors:

   Run: ./your_program (no arguments)
   Run: ./your_program arg1 arg2 arg3 (too many arguments)
   Run: ./your_program nofile.txt (file does not exist)

Purpose: Checks if error messages are correctly displayed on the console and that the program exits as intended.

2. Empty Input File:

   Purpose: Ensures the program handles an empty file without crashing.

3. Incorrect Data Lines in a File:

   ```
   GOODSTOCK 1 100 10.5 10.0
   BADSTOCK1 A B C D (letters instead of numbers)
   BADSTOCK2 1 50 (too little information)
   GOODSTOCK 2 200 11.5 11.0
   ```

   Purpose: Checks that the program skips incorrect lines, prints the correct error message, and continues processing valid lines.

4. File With Every Line Being Incorrect:

   ```
   This is not data
   1 2 3 bad line
   ```

   Purpose: Checks that the program terminates as intended when no data can be processed.

5. Lines with Negative Values:

   ```
   STOCKNEG 1 -100 10.0 9.0
   ```

   Purpose: To see that the program processes values that are equal to or less than zero as intended.