c++ ДЗ

Петрова Елизавета

31 октября 2025 г.

orderbook.hpp

Что было не так

Парсер CSV был чувствителен к регистру. Заголовки колонок (`SECCODE`, `BUYSELL`, `TRADEPRICE` и т. д.) ожидались строго в верхнем регистре. Значения полей `ACTION`/`BUYSELL` сравнивались по регистру частично. Это ломает импорт при приходе файлов с `camelCase`/`TitleCase`/`lowercase`.

Что сделано и зачем

Добавлен утилитарный метод `Upper(std::string_view)` для нормализации строк к верхнему регистру (ASCII). Метод объявлен в `orderbook.hpp` (реализация в `orderbook.cpp`). Остальной публичный API не менялся. Цель — сделать парсинг регистронезависимым и устойчивым к вариативным CSV.

git diff

```
@@ -1,12 +1,14 @@
 #ifndef BACKTESTER_ORDERBOOK_L3_H_
 #define BACKTESTER_ORDERBOOK_L3_H_

 #include <cstdint>
 #include <functional>
 #include <list>
 #include <map>
 #include <optional>
+#include <string>
 #include <string_view>
 #include <unordered_map>
 #include <unordered_set>
 #include <vector>

 namespace backtester {
@@ -98,12 +100,16 @@  class OrderBookL3 {
   static std::string_view Trim(std::string_view s);
   static std::string NormalizeId(std::string_view s);  // trim + strip leading zeros
   static std::vector<std::string_view> SplitLine(std::string_view line, char delim);
   static std::unordered_map<std::string, int>
       BuildHeaderIndex(const std::vector<std::string_view>& hdrs);
+  // Helper: ASCII upper-case copy (для сопоставления без учёта регистра).
+  // Упрощённо (ORDLOG - ASCII), без локали/Unicode.
```

```
+  static std::string Upper(std::string_view s);

   // Book ops
   void AddOrder(const OrderKey& rec);
   void CancelOrder(const std::string& order_id);
   void ReduceOrder(const std::string& order_id, std::int64_t qty);

   // Queries
   std::optional<Level> BestFromBook(const std::map<double, PriceLevel>& book,
                                     bool highest) const;
   void AccumulateLevel(const PriceLevel& lvl, Level& out) const;
 };

 } // namespace backtester

 #endif // BACKTESTER_ORDERBOOK_L3_H_
```

Итог

Заголовок подготовлен для регистронезависимого парсинга, без изменения публичного интерфейса.

orderbook.cpp

Что было не так

Чувствительность к регистру заголовков/значений. Нет защиты от «коротких» строк CSV (меньше колонок, чем в заголовке) — риск `out-of-range`. Сторона `BUYSELL`: всё «не BUY» шло в `kAsk`. Лучше явно распознавать `SELL/S`.

Что сделано и зачем

Регистронезависимость: используем `Upper()` в `BuildHeaderIndex` и в парсерах `ACTION`/`BUYSELL`. Валидация длины строки: добавлен лямбда-хелпер `has(idx)` и ранний `continue`, если строка короче, чем нужно (защита от `f[i]` вне диапазона). **BUYSELL**: теперь явно поддержаны `S`/`SELL` (остальное — `kAsk` по умолчанию).

git diff

```
@@ -1,10 +1,13 @@
 #include "libbacktester/orderbook.hpp"

 #include <charconv>
 #include <fstream>
 #include <stdexcept>
 #include <string>
+#include <algorithm>
+#include <cctype>
+#include <iterator>

 namespace backtester {

 //------------------ ctor/dtor ------------------
```

```
@@ -90,23 +93,30 @@ void OrderBookL3::LoadCsv(const std::string& filepath) {
   const std::string colTradeId    = "TRADENO";
   const std::string colTradePrice = "TRADEPRICE";

-  auto parseAction = [](std::string_view sv) -> Action {
-    sv = Trim(sv);
-    if (sv == "1" || sv == "ADD" || sv == "Add")       return Action::kAdd;
-    if (sv == "2" || sv == "TRADE" || sv == "Trade")   return Action::kTrade;
-    if (sv == "0" || sv == "CANCEL" || sv == "DEL")    return Action::kCancel;
+  auto parseAction = [](std::string_view sv) -> Action {
+    sv = OrderBookL3::Trim(sv);
+    const std::string up = OrderBookL3::Upper(sv);
+    if (up == "1" || up == "ADD")              return Action::kAdd;
+    if (up == "2" || up == "TRADE")            return Action::kTrade;
+    if (up == "0" || up == "CANCEL" || up == "DEL") return Action::kCancel;
     return Action::kUnknown;
   };
-  auto parseSide = [](std::string_view sv) -> Side {
-    sv = Trim(sv);
-    if (sv == "B" || sv == "BUY" || sv == "Buy" || sv == "1") return Side::kBid;
-    return Side::kAsk;
+  auto parseSide = [](std::string_view sv) -> Side {
+    sv = OrderBookL3::Trim(sv);
+    const std::string up = OrderBookL3::Upper(sv);
+    if (up == "B" || up == "BUY" || up == "1") return Side::kBid;
+    if (up == "S" || up == "SELL"|| up == "2") return Side::kAsk;
+    return Side::kAsk; // по умолчанию оставляем Ask (как ранее)
   };
   auto to_i64 = [](std::string_view sv, std::int64_t& out) -> bool {
-    sv = Trim(sv);
+    sv = OrderBookL3::Trim(sv);
     auto r = std::from_chars(sv.data(), sv.data()+sv.size(), out);
     return r.ec == std::errc();
   };
   auto to_f64 = [](std::string_view sv, double& out) -> bool {
-    sv = Trim(sv);
+    sv = OrderBookL3::Trim(sv);
     auto r = std::from_chars(sv.data(), sv.data()+sv.size(), out);
     return r.ec == std::errc();
   };
@@ -129,6 +139,13 @@ void OrderBookL3::LoadCsv(const std::string& filepath) {
   const int i_tradeId    = getOpt(colTradeId);
   const int i_tradePrice = getOpt(colTradePrice);

+  auto has = [&](int idx, const std::vector<std::string_view>& row) -> bool {
+    return idx >= 0 && static_cast<std::size_t>(idx) < row.size();
+  };
+
   std::string line;
   while (std::getline(in, line)) {
     if (line.empty()) continue;
     auto f = SplitLine(line, kDelim);
+    // защита от «коротких» строк (иначе f[i] может быть OOB)
+    if (!has(i_time,f) || !has(i_action,f) || !has(i_orderno,f) ||
+        !has(i_side,f) || !has(i_price,f)  || !has(i_qty,f)) continue;

     if (i_seccode >= 0) {
       if (Trim(f[i_seccode]) != std::string_view{ticker_}) continue;
```

```
@@ -203,10 +220,22 @@ std::unordered_map<std::string, int>
 OrderBookL3::BuildHeaderIndex(const std::vector<std::string_view>& hdrs) {
   std::unordered_map<std::string, int> m;
   m.reserve(hdrs.size());
   for (int i = 0; i < static_cast<int>(hdrs.size()); ++i) {
-    std::string_view k = Trim(hdrs[i]);
-    m.emplace(std::string(k), i);
+    std::string k = Upper(Trim(hdrs[i]));
+    m.emplace(std::move(k), i);
   }
   return m;
 }

+// static
+std::string OrderBookL3::Upper(std::string_view s) {
+  std::string r;
+  r.reserve(s.size());
+  for (unsigned char c : std::string(s)) {
+    r.push_back(static_cast<char>(std::toupper(c)));
+  }
+  return r;
+}
+
```

Итог

Парсер стал: регистронезависимым, безопасным к коротким строкам и теперь явно понимает SELL/S.

test_orderbook.cpp

Что было не так

В тесте `CancelRemovesAndPrunesLevel` отмена ссылалась на `ORDERNO=0`, хотя по смыслу должна отменять ордер #1.

Что сделано и зачем

Тесты на регистронезависимость заголовков и значений (camelCase/lower/UPPER). Тест на короткую строку CSV (меньше столбцов) — строка корректно пропускается, а не валит парсер. Оставленные ранее проверки агрегации уровня/порядка обхода — совместимы.

git diff

```
@@ -31,18 +31,18 @@ TEST(OrderBookL3Test, AddAndBest) {
   EXPECT_EQ(ba->agg_qty, 200);
 }

-// ----- Cancel removes order and prunes empty level -----
+// ----- Cancel removes order and prunes empty level (fix ORDERNO -> 1) -----
 TEST(OrderBookL3Test, CancelRemovesAndPrunesLevel) {
   const std::string csv =
       "NO,SECCODE,BUYSELL,TIME,ORDERNO,ACTION,PRICE,VOLUME,TRADENO,TRADEPRICE\n"
```

```
       "1,FOO,B,1000,1,1,10.00,100,,\n"
-      "2,FOO,B,1001,2,9.50, 50,,,\n" // spurious 'trade' row w/o TRADENO will be
↪  ignored (unknown), harmless
-      "3,FOO,B,1002,2,0,0,,,\n"      // also ignored
-      "4,FOO,B,1003,2,0,0,,,\n"      // ignored
-      "5,FOO,B,1004,0,0,0,0,,\n";    // ACTION=0 cancel order #1
+      "2,FOO,B,1001,2,9.50, 50,,,\n"
+      "3,FOO,B,1002,2,0,0,,,\n"
+      "4,FOO,B,1003,2,0,0,,,\n"
+      "5,FOO,B,1004,1,0,0,0,,\n";    // FIX: cancel ORDERNO=1

   backtester::OrderBookL3 ob("FOO");
   ob.LoadCsv(WriteCsv("ob_cancel.csv", csv));
   EXPECT_FALSE(ob.bestBid().has_value());
 }
@@ -152,6 +152,69 @@ TEST(OrderBookL3Test, CancelUnknownIsNoop) {
   EXPECT_DOUBLE_EQ(bb->price, 10.0);
   EXPECT_EQ(bb->agg_qty, 100);
 }

+// ----- Headers are case-insensitive; camelCase works -----
+TEST(OrderBookL3Test, HeaderCaseInsensitiveAndCamelCase) {
+  const std::string csv =
+      "No,SecCode,BuySell,Time,OrderNo,Action,Price,Volume,TradeNo,TradePrice\n"
+      "1,FOO,b,1000,1,add,10.00,100,,\n"
+      "2,FOO,S,1000,2,Add,11.00,200,,\n"
+      "3,FOO,B,1001,1,cancel,0,0,,,\n";
+  backtester::OrderBookL3 ob("FOO");
+  ob.LoadCsv(WriteCsv("ob_hdr_camel.csv", csv));
+  auto ba = ob.bestAsk();
+  ASSERT_TRUE(ba.has_value());
+  EXPECT_DOUBLE_EQ(ba->price, 11.0);
+  EXPECT_EQ(ba->agg_qty, 200);
+  EXPECT_FALSE(ob.bestBid().has_value());   // #1 was canceled
+}
+
+// ----- Action/Side values are case-insensitive ('buy/add/trade/cancel') -----
+TEST(OrderBookL3Test, ActionAndSideCaseInsensitive) {
+  const std::string csv =
+      "NO,SECCODE,BUYSELL,TIME,ORDERNO,ACTION,PRICE,VOLUME,TRADENO,TRADEPRICE\n"
+      "1,FOO,buy,1000,1,add,10.00,120,,\n"          // add bid 120
+      "2,FOO,SELL,1000,2,ADD,10.00, 80,,\n"         // add ask 80
+      "3,FOO,buy,1001,1,TRADE,10.00, 80,tx1,10.00\n"
+      "4,FOO,SeLl,1001,2,trade,10.00, 80,tx1,10.00\n"
+      "5,FOO,BUY,1002,1,CaNcEl,0,0,,,\n";           // cancel remainder
+  backtester::OrderBookL3 ob("FOO");
+  ob.LoadCsv(WriteCsv("ob_case_insensitive.csv", csv));
+  EXPECT_EQ(ob.openQty("1"), 0);
+  EXPECT_FALSE(ob.bestBid().has_value());
+  EXPECT_FALSE(ob.bestAsk().has_value());
+}
+
+// ----- Short CSV row is safely skipped (no out-of-range) -----
+TEST(OrderBookL3Test, ShortRowIsSkippedSafely) {
+  const std::string csv =
+      "NO,SECCODE,BUYSELL,TIME,ORDERNO,ACTION,PRICE,VOLUME,TRADENO,TRADEPRICE\n"
+      "1,FOO,B,1000,1,1,10.00,100,,\n"
+      "2,FOO,S,1000,2,1,11.00,200,,\n"
+      "3,FOO,B,1001,3,1,9.50, 50\n"; // <-- укороченная строка (нет всех запятых)
```

```
+  backtester::OrderBookL3 ob("FOO");
+  ob.LoadCsv(WriteCsv("ob_short_row.csv", csv));
+  auto bb = ob.bestBid();
+  auto ba = ob.bestAsk();
+  ASSERT_TRUE(bb.has_value());
+  ASSERT_TRUE(ba.has_value());
+  EXPECT_DOUBLE_EQ(bb->price, 10.00);
+  EXPECT_DOUBLE_EQ(ba->price, 11.00);
+}
```

Итог

Исправлен неверный `ORDERNO` в тесте отмены; добавлены проверки на case-insensitive парсинг и безопасную обработку коротких строк CSV.