Part 3: Secure programming

Introduction

The chosen program is from the Object-Oriented Programming module teaching C++, the midterm project is to program an Advisorbot that user interacts with. The user type in commands like `help`, `help <cmd>`, `<cmd>`…… and the program reacts accordingly. Since C++ is a strongly typed language, a lot of data type checking is included, which, if not done properly, may cause some serious secure problems.

Recommendation 1: use `size_t` in `for` loops

```
std::vector<float> Advisorbot::stepThroughTimestep(ProductType currency, OrderBookType orderType, int timeStep) {
    std::vector<float> average;
    for (int i = 0; i < timeStep; ++i) {
        currentTime = orderBook.getPrevTime(currentTime);
        std::vector<OrderBookEntry> entries = orderBook.getOrders(orderType, currency, currentTime);
        for (int j = 0; j < entries.size(); ++j) {
            average.push_back(entries[j].price);
        }
    }
    return average;
};
```

There are a total of 6 `for` loops in the main cpp file for the program. 5 of them include an index, which should be replaced by `size_t` to prevent possible buffer overflow. In fact, any variable that represents the size of an object, i.e. size, index, length, loop counter, should use `size_t` type, which is an unsigned integer type.

Recommendation 2: default to use `const`

```
    double minValue(ProductType currency, OrderBookType orderType, std::string
    currentTime);
    /** Print out maximum bid or ask product in current time step*/
    void maxProduct(std::vector<std::string> inputCommand);
    /** Find maximum bid or ask for product in current time step */
    double maxValue(ProductType currency, OrderBookType orderType, std::string
    currentTime);
    /** Compute average ask or bid for the sent product over the sent number of
    time steps */
    void avgProduct(std::vector<std::string> inputCommand);
    /** Calculate the average price through a given number of timestamps */
    double calculateAvg(ProductType currency, OrderBookType orderType, int
    timeStep);
```

```
    double min_value(const ProductType currency, const OrderBookType order_type, std::string
    current_time);
    /** Print out maximum bid or ask product in current time step*/
    void max_product(const std::vector<std::string>& input_command);
    /** Find maximum bid or ask for product in current time step */
    double max_value(const ProductType currency, const OrderBookType order_type, std::string
    current_time);
    /** Compute average ask or bid for the sent product over the sent number of time steps */
    void avg_product(std::vector<std::string> inputCommand);
    /** Calculate the average price through a given number of timestamps */
    double calculate_avg(const ProductType currency, const OrderBookType order_type, const int
    time_step);
```

When going through the program for secure programming checking, I found out that a lot of variables I used inside a function (locally) is better off as a constant. The above two images are examples of before and after the change. Since those inputs are important keys to what the function will output (as flags), it is crucial to keep it as a constant. Also, I learned from some folks in the industry talking about one of the best practices is programing is to default to `const` unless found out the variable is in need of a change, the declaration will then be changed.

Recommendation 3: type conversion (narrowing conversion)

```
float Advisorbot::prediction(const std::string& min_max, const OrderBookType order_type, const ProductType currency) {
    std::vector<float> X;
    std::vector<float> Y;  // max/min output

    std::vector<std::vector<float>> Z(2);

    /** Insert the past 3 records into data */
    for (int i = 0; i < 3; ++i) {  // 3 timestamps
        X.push_back(i);
        if (min_max == "  double Advisorbot::min_value(ProductType currency, OrderBookType order_type, std::string current_time)
            Y.push_back(
        } else if (min_m Find minimum bid or ask for product in current time step
            Y.push_back(min_value(currency, order_type, current_time_));
        } else {
            std::cout << "The second parameter is either 'max' or 'min'." << std::endl;
            break;
        }
        current_time_ = order_book_.get_prev_time(current_time_);
    }
```

Since C++ is a strongly typed programming language, it is important to make sure the data types matched or is converted properly to prevent buffer overflow.

In this case, the `min_value()` returns a `double` while the vector it was pushed into is typed `float`. Because `double` has bigger size than `float`, such usage will cause a **narrowing conversion**, which might be troublesome if the value went over the range for `float`.

The solution can simply be modifying one of the types to be the same as the other. To be on the safe side, all the `float` should be changed to `double` since from looking at the data that's been processed in the program, there are quite some small numbers involved.