REPORT

Introduction

The advisorbot program is designed to help a crypto-investor in the analysis of data available on the exchange. The program starts and runs on the command line allowing the user to enter commands and receive responses.

Implemented commands

Command	Description	Status
help	list all available commands	Implemented
help <cmd>	output help for the specified command <cmd>	Implemented
prod	list available products	Implemented
min <product> <bid|ask>	find minimum bid or ask for product in current time step	Implemented
max <product> <bid|ask>	find maximum bid or ask for product in current time step	Implemented
avg <product> <ask|bid> <timesteps>	compute average ask or bid for the sent product over the sent number of time steps	Implemented
predict <min|max> <product> <bid|ask>	predict max or min ask or bid for the sent product for the next time step	Implemented
time	state current time in dataset, i.e. which timeframe are we looking at	Implemented
step	move to next time step	Implemented
median	compute median ask or bid for the sent product over the sent number of time steps (own command)	Implemented

Description of parsing code

Two functions have been developed to handle user input:
1. trim() - removes whitespaces and end-of-line characters from the beginning and end of the string;
2. tokenise - splits a string into tokens using a space as a separator.
Both functions are implemented in StringUtils module.

The user input after splitting into tokens is a vector of words (std:string).
Next, the first word (from the vector) is compared with a set of commands, and if it matches, the corresponding processing function is called to the input of which is a vector of tokens.
Each command checks the correctness of the arguments on its own.

If the argument is an integer value, then the std::stoi() function is used to convert from a string to a number. After a successful conversion, the value is checked for a valid range. try/catch/throw are used to process the errors. String parameters such as min, max, bid, ask are checked for valid values by a simple string comparison.

Custom command

The median command has been implemented, which calculates the median value of the price for a user-defined product, bid/ask and time period.

Optimisation of the exchange code

The function of obtaining a list of exchange products (OderBook::getKnownProducts) has been optimized. In order not to scan the entire list of records every time when accessing, the search for available products is performed once when reading the database and is stored as a vector of strings.

Since most of the functions work at a certain point in time timestamp (usually back, towards history), the concept of a cursor was introduced - an iterator to the beginning of the current timeframe.
Thus, when executing commands, there is no repeated search for the current timestamp in the database (that is, the beginning of the current timeframe), which greatly reduces the execution time of commands and increases program performance.
The step command changes the position of the cursor moving it to the next time frame.