Work on projects

To be concrete, the instructions below are given for the case of `quant1` and for the `Windows-WSL-Ubuntu` setup. The other projects in the course are `quant2` and `quantathon`. The same instructions also hold for the `MacOS-Homebrew` setup after the changes:

Command Palette: `(Ctrl+Shift+P) -> (Cmd+Shift+P)`.

Build a project: `(Shift+F7) -> (Shift+Fn+F7)`.

Run a project: `(Shift+F5) -> (Shift+Fn+F5)`.

Debug a project: `(Ctrl+F5) -> (Ctrl+Fn+F5)`.

Installation of project `quant1`

1. Download file quant1.zip.

2. Extract the contents of the file to directory Vega\VegaQ. Check that you get the directory tree as Vega:\VegaQ\quant1 (not as Vega:\VegaQ\VegaQ\quant1).

3. Open folder \VegaQ with VS Code. Open file \VegaQ\CMakeLists.txt. This is the same file where you wrote your "YOUR_ID" while installing the course package. Uncomment line

   ```
   # add_subdirectory(quant1)
   ```

   that is, remove `#`.

4. Configure the project with `(Ctrl+Shift+P)` and `(CMake:Configure)` and then build it with `(Shift+F7)`. You will see the error messages like:

   ```
   ...  Linking CXX executable quant1
   ...
   [build] Build finished with exit code 2
   ```

   These errors occur because the functions declared in header file \VegaQ\quant1\quant1.hpp have not been implemented yet.

5. The documentation for these functions is provided in two places:

   (a) in file quant1.pdf;

   (b) in directory \build\doc\quant1\html created as part of the previous step. Click on any *.html file.

6. Create *.cpp files (one per problem) in directory \VegaQ\quant1\Src and implement the requested functions. To make your code to look nice, it is a good idea to run for each of your *.cpp files the format commands:

   `(Ctrl+Shift+I)`: Windows-WSL or Ubuntu.

   `(Shift+Alt+F)`: Mac OS.

7. Configure with `(Ctrl+Shift+P)` and `(CMake:Configure)`, compile with `(Shift+F7)`, and run the project with `(Ctrl+F5)` or `(Shift+F5)`.

8. If everything works fine, then file quant1.txt will be created in directory Vega\build\output\quant1. Check that "YOUR_ID" appears on the first line.

Hint. A good way to start your work is to return the default constructor of `cfl::MultiFunction` for every function in the project. For instance, the initial implementation for `prb::put` looks like

```
cfl::MultiFunction
prb::put(double dStrike, double dMaturity, AssetModel &rModel)
{
  return cfl::MultiFunction();
}
```

Do such implementation for *every* function in the project. You will be able to run the project and get the output file `quant1.txt`. The column of the results will contain only zeros, because the default constructor of `cfl::MultiFunction` builds function with constant value zero. Now you need to start thinking about algorithms ....