Project TroBeta

Team Members:

1. Παναγιώτης Πλυτάς --> Αριθμός Μητρώου: 1115201500134
2. Δημήτριος Γούναρης --> Αριθμός Μητρώου: 1115201500032

Compilation:

To create the **executable file** use command:

```
g++ -o TroBeta TroBeta.cpp GeneralFunctions.cpp Bets.cpp History.cpp Hierarchy.cpp Users.cpp
```

Execution:

To login as an already existing user or as a guest use command:

```
./TroBeta
```

To register and login to the system as a new punter use command:

```
./TroBeta -R
```

Files:

- Bets.h
- GeneralFunctions.h
- Hierarchy.h
- HierarchyLibraries.h
- History.h
- HistoryLibraries.h
- UserLibraries.h
- Users.h

- Bets.cpp
- GeneralFunctions.cpp
- Hierarchy.cpp
- History.cpp
- TroBeta.cpp
- Users.cpp

- bets.csv
- users.csv
- audit.log
- hierarchy.dat

Task Approach:

First we categorized the project's tasks based on the classes that were necessary for each one of them. The categories were:

- History
- Hierarchy
- Bets
- Users
- General Functions (for various handling functions, login, registration etc.)

And then we decided which of these categories each of us had to implement:

- Δημήτρης Γούναρης → History, Hierarchy
- Παναγιώτης Πλυτάς → Bets, Users, General Functions, Main

Implementation Info

(For each team member)

1. Team Member: Παναγιώτης Πλυτάς (sdi1500134)

Tasks:

- BetFile class
- UserFile class
- User, Guest, Punter, Trader, Director, Wallet class
- General Functions and Main

Consensuses:

- The functions in the GeneralFunctions.cpp file are used to extract different information from the system files based on the delimiters (user_id, username, node_id, stake etc.) and to control the login, registration and hierarchy navigation.

- In order to handle and use the files users.csv and bets.csv I used two classes (UserFile and BetFile) to implement the various operations that had to be applied to them (change_balance, lock_unlock_user).

- The updating of the files users.csv and bets.csv is being done immediately after every operation that affects their contents and this process is achieved by opening the files when necessary, changing and putting their contents in a vector, deleting the old file, creating a new file and putting the new contents from the vector in it.

- What is being printed and what operations can be executed by each user is decided by the classes through virtual functions.

- Class hierarchy:
```
                  User
                    | (is a)
                  Guest
           (is a) /     \ (is a)
          Punter       Trader
                          \ (is a)
                          Director
```

- In the punter’s profile menu an operation X (Exit) has been added which allows the user to exit from this menu and return to the hierarchy.

- Whenever a trader settles a bet the system’s history is being updated so the system profits can be calculated from the information in the audit.log file.

- Directors can lock only punters and a reason is required only when locking a punter not when activating one.

2. Team Member: Δημήτρης Γούναρης (sdi1500032)

Tasks:

- HistoryAdapter class
- Hierarchy class
- Node class (and its subclasses)

Consensuses:

- Every function that has indexes as arguments (e.g. Hierarchy::toChild(int)), needs the index starting from 1, NOT from 0.

- The cost for dynamic pointers was too big, considering that we have a solid hierarchy to create. Thus, I have designed the code without taking into consideration the case of Categories having Events as children etc. Instead, the hierarchy and the Node class are designed in a way to provide a stable "framework" for whoever wants to implement this type of hierarchy.

- Node class, and all its subclasses, have only protected and private members. Node is not abstract, because I designed the hierarchy as a tree with one head, of type Node, having Categories as children. However, since Node functions (except from constructor) are only accessible through Hierarchy, there is no real danger by letting Node as common class.

- Hidden, voided, winning and losing nodes are flagged in the hierarchy file with {H}{V}{W}{L} (syntax as shown, no commas and no spaces).

- Hierarchy.dat is supposed to present the data in a way that every new element will not be more than one level below the previous. For example, we can't have a category declared, then a selection in this category declared and later declare the subcategory/event/market for this selection.

- History is in a different class for easy debugging and because it didn't really fit into another class (it's not a part of hierarchy by itself and is not a part of users either).

- In events, time and date are stored as strings, since the assignment does not suggest any operations on them and it was easier to split them as strings rather than in different int values. If there is no date on the event, the string remains empty and the hasTime boolean variable is set to false.

- When copying a node, the node and all its children are hidden and marked as " - Copy".

- When hiding a node, all the children are kept visible, but users still can't access them because they have no access to hidden nodes.

- Every node created is visible and is considered consistent. Thus, there is no check whether there are selections in an event etc.